Non-Interactive Batch Arguments for NP from Standard Assumptions

Arka Rai Choudhuri Abhishek Jain Zhengzhong Jin

Johns Hopkins University

Abstract

We study the problem of designing *non-interactive batch arguments* for NP. Such an argument system allows an efficient prover to prove multiple NP statements, with size smaller than the combined witness length.

We provide the first construction of such an argument system for NP in the common reference string model based on standard cryptographic assumptions. Prior works either require non-standard assumptions (or the random oracle model) or can only support private verification.

At the heart of our result is a new *dual mode* interactive batch argument system for NP. We show how to apply the correlation-intractability framework for Fiat-Shamir – that has primarily been applied to proof systems – to such interactive arguments.

Contents

1 Introduction

Consider the following scenario: Alice wants to convince Bob of the veracity of k statements $(x_1, \ldots, x_k)$ in an NP language. A naïve solution is for Alice to send a witness w_i for each of the k instances and for Bob to verify each pair (x_i, w_i). This proof is *non-interactive* (i.e., consists of a single message) as well as *publicly verifiable* (i.e., anyone can verify its correctness). However, it is quite expensive, requiring communication that grows linearly with the combined length of the witnesses.

Can we do better? That is, can we non-interactively prove k NP statements with communication much smaller than $k \cdot m$, where $m = m(|x|)$ is the witness length? Addressing this question is the main focus of this work.

Batch Arguments. We study the problem of designing *batch arguments* (BARG) for NP in the common reference string (CRS) model. Such an argument system allows an efficient prover to compute a non-interactive and publicly verifiable "batch proof" of k NP instances, with size much smaller than $k \cdot m$. If any of the k instances is false, then no polynomial-time cheating prover must be able to produce an accepting proof.

In the *interactive* setting, the problem of batch proofs was first studied by Reingold, Rothblum and Rothblum [RRR16] and more recently in [RRR18, RR20]. The focus of these works is on achieving statistical soundness, and we refer the reader to Section 1.2 for a discussion. In this work, we focus on the (harder) non-interactive setting but settle for the weaker notion of computational soundness.

Since verifying the membership of k NP instances is itself an NP problem, BARGs with poly-logarithmic communication can be obtained from succinct non-interactive arguments (SNARGs) for NP [Mic94, BCCT12, DFH12, GLR11, BCCT13]. However, SNARGs for NP are presently only known to exist under strong, non-falsifiable assumptions [Nao03, GW11] (or the random oracle model). In the *designated-verifier* setting, Brakerski, Holmgren and Kalai [BHK17] constructed two-message batch arguments for NP with communication proportional to the size of a single witness, assuming the existence of a computational private information retrieval scheme [KO97, CGKS95]. The main drawback of their solution is that it requires *private* verification. Recently, Kalai, Paneth and Yang [KPY19] constructed the first non-interactive *publicly verifiable* batch arguments for NP, but rely on a new non-standard (but falsifiable) assumption on groups with bilinear maps.

This state of affairs motivates the following basic question:

Do there exist BARGs for NP *based on standard assumptions?*

1.1 Our Results

We provide the first construction of a publicly verifiable non-interactive batch argument system for NP in the CRS model from standard computational assumptions. Our scheme achieves non-adaptive computational soundness.

Theorem 1 (Informal)**.** *Let* C-SAT *be the circuit satisfiability language defined by a boolean circuit $C : \{0,1\}^{|x|} \times \{0,1\}^{|y|} \mapsto \{0,1\}$. Assuming standard computational assumptions, there exists a BARG for* C-SAT *in the CRS model with non-adaptive soundness. The proof size for k statements is $\tilde{O}((|C| + \sqrt{k|C|}) \cdot \lambda)$, where λ is the security parameter.*

When the number of statements k is smaller than $|C|$, the size of the proof only grows with $|C|$; otherwise, it essentially only grows with k.

On our assumptions. Our construction relies on two essential cryptographic components:

– **Somewhere-Extractable Linearly Homomorphic Commitment.** The first building block for achieving our result is a new notion of *somewhere-extractable linearly homomorphic commitment* (SE-LHC) schemes (Section 4). We show an instantiation of SE-LHC assuming the hardness of the quadratic residuosity (QR) assumption.

– **Correlation-Intractable Hash Functions for TC^0.** Our second cryptographic building block is a correlation-intractable hash function (CIH) [CGH04] for TC^0 circuits. CIH for bounded-depth polynomial-size circuits are known from the learning with errors (LWE) assumption [PS19, CCH+19]. Very recently, CIH for TC^0 circuits were constructed based on the sub-exponential hardness of the Decisional Diffie-Hellman (DDH) assumption against polynomial-time adversaries [JJ21].

Putting together the above, Theorem 1 can be instantiated based on QR *and* either LWE or sub-exponential DDH.

We refer the reader to Section 2 for an overview of our construction.

On adaptive soundness. Our construction in Theorem 1 achieves non-adaptive (computational) soundness. This seems inherent, as there are known barriers to constructing BARGs with adaptive soundness based on falsifiable assumptions. Specifically, Brakerski, Holmgren and Kalai [BHK17] showed a transformation from *adaptively-sound* BARGs (with argument of knowledge property[1]) to adaptively-sound SNARGs using RAM delegation schemes. This in turn allows for using the Gentry-Wichs [GW11] black-box lower bound for SNARGs. We refer the reader to Section 7.4 for more details.

1.2 Related Works

Batch verification is an interesting question for various cryptographic primitives, and can lead to practical benefits in some settings (see, e.g., [CHP12]).

In the setting of interactive *proofs*, the problem of batch verification of NP has been recently studied in a sequence of works [RRR16, RRR18, RR20]. These works consider the class UP, a subset of NP, where each statement in the language has a unique witness of membership. To the best of our knowledge, no positive results are known in this regime for NP. It should be noted that while there are lower bounds on the communication complexity of interactive proofs for languages in NP [GVW02, GH98], the lower bounds do not appear to directly extend to the NP batch language $L^{\otimes k}$ due to the additional structure inherent to $L^{\otimes k}$. We refer the reader to [RRR16] for a detailed discussion on this topic.

If we consider computational soundness, where security holds only for computationally bounded cheating provers, Killian's protocol [Kil92] gives us an interactive batch argument based on collision resistance of hash functions. In the non-interactive setting, Brakerski, Holmgren and Kalai [BHK17] construct *privately-verifiable* non-adaptive batch arguments (of knowledge) assuming computational private information retrieval schemes. Kalai, Paneth and Yang [KPY19] construct a *publicly-verifiable* non-adaptive batch argument, but rely on a new (falsifiable) decisional assumption on groups with bilinear pairings. One can also generically use SNARGs to construct non-interactive batch arguments, but constructions of SNARGs are only known based on strong non-falsifiable assumptions (or in the random oracle mode).

Very recently, there have been works that consider the problem of batch verification for statistical zero-knowledge (SZK) proofs [KRR+20, KRV21]. The specific goals in these works are orthogonal to the problem we consider: the prover in these works is no longer required to be efficient, but it is imperative that the resultant batch protocol is also an SZK proof system.

[1]Our construction in Theorem 1 achieves (non-adaptive) argument of knowledge property.

2 Technical Overview

As established in the introduction, we want to design publicly verifiable non-interactive batch arguments for NP. To this end, there exists a well-studied general paradigm one could follow: (i) First, construct an interactive public-coin proof system (P, V) for NP; (ii) Next, apply the Fiat-Shamir (FS) round-collapsing transform[FS87] on (P, V) with respect to some hash function family $\mathcal{H}$ to obtain a non-interactive proof.

Originally, the soundness of the FS transformation was only established when modeling the hash family as a random oracle. But the transformation has garnered a lot of recent attention with an exciting line of work that demonstrate the soundness of the transformation when the hash function family is *correlation intractable* [CGH04]. In particular, this idea has been used with much success in the context of non-interactive zero-knowledge arguments [KRR17, CCRR18, HL18, CCH+19, PS19, BKM20, CPV20, CKU20, JJ21], (publicly verifiable) succinct non-interactive arguments of knowledge for log-space uniform computation [CCH+19, JK20, KZ20, JKKZ21] and establishing the hardness of the complexity class PPAD [CHK+19, LV20, JK20, KZ20, JKKZ21].

Since this paradigm is central to our work as well, we start by describing the transformation, correlation intractability (CI), and the role it plays in the soundness of the transformation.

2.1 Background

Fiat-Shamir Transformation and CI. The Fiat-Shamir transform with respect to some hash family $\mathcal{H}$, utilizes a sampled hash function $h \leftarrow \mathcal{H}$ as the common reference string (CRS) to convert a public-coin interactive protocol into a non-interactive proof in the CRS model, where the verfier's messages are derived (non-interactively) by the prover applying the hash function h to the transcript. For instance, consider the following flow of messages between the prover and the verifier, where the verifier's message β is a uniformly random string:

$$\begin{array}{ccc} \underline{\mathsf{P}}(x) & & \underline{\mathsf{V}}(x) \\ & \xrightarrow{\quad\alpha\quad} & \\ & \xleftarrow{\quad\beta\quad} & \\ & \xrightarrow{\quad\gamma\quad} & \end{array}$$

The prover computes the verifier's message as $\beta := h(x, \alpha)$, and the resultant non-interactive proof is the tuple (α, β, γ) - the verifier can recompute β and check if the prover did indeed compute it correctly. Unlike soundness in the interactive setting, where a cheating prover P^* has no control over the verifier's message β, in the transformed non-interactive protocol, P^* has *some* control over β. Specifically, P^* can try different values of α to input into the hash function until it gets a β that it considers favorable. Let's formalize what we mean. For a statement $x \notin L$, when the prover evaluates the hash function h on (x, α), it wants to find an element from the following set of *bad challenges*,

$$\mathcal{B}_{x,\alpha} := \left\{\beta \,\middle|\, \exists \gamma \text{ s.t. } \mathsf{V}(x, \alpha, \beta, \gamma) = 1\right\}. \tag{1}$$

Can we hope to enforce some restrictions on the hash family $\mathcal{H}$, such that it is intractable to find an α such that $h(x, \alpha) \in \mathcal{B}_{x,\alpha}$, i.e. the hash evaluation doesn't result in a *bad challenge*? This is exactly where the correlation intractability of the hash family $\mathcal{H}$ helps. Intuitively, $\mathcal{H}$ is a correlation intractable hash family (CIH) for a function f, if the following holds for all probabilistic polynomial time adversary (PPT) $\mathcal{A}$,

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

This yields the following idea - define a function $\mathsf{BAD}(\cdot)$, that on input (x, α), outputs an element in $\mathcal{B}_{x,\alpha}$. Let us for the moment assume that $\mathcal{B}_{x,\alpha}$ for all α and $x \notin L$ has at most a single element. If $\mathcal{H}$ is a CIH for $f(\cdot) := \mathsf{BAD}(\cdot)$, then any cheating prover that outputs an accepting transcript (α, β, γ) for $x \notin L$ must break the correlation intractability of $\mathcal{H}$ since $\beta \in \mathcal{B}_{x,\alpha}$ by definition.

But what about when $\mathcal{B}_{x,\alpha}$ consists of multiple elements? We want to argue that the cheating prover doesn't output *any* element from $\mathcal{B}_{x,\alpha}$. If $|\mathcal{B}_{x,\alpha}|$ is *polynomially bounded*, we can argue this via a simple application of the union bound: modify $\mathsf{BAD}(\cdot, \cdot)$ to additionally take in as input an index i, and output the i-th element of $\mathcal{B}_{x,\alpha}$ (for some ordering of the elements). Let $f_i(\cdot) := \mathsf{BAD}(\cdot, i)$, then by the union bound we have for any PPT adversary $\mathcal{A}$,

$$\Pr_{h \leftarrow \mathcal{H}}\left[h(x) \in \{f_1(x), \cdots, f_{|\mathcal{B}|}(x)\} \mid \mathcal{A}(h) = x\right] \leq |\mathcal{B}| \cdot \mathsf{negl}(\lambda).$$[2]

While our description above is for a protocol with a single verifier message, this can be extended to multi-round protocols by further constraining the interactive protocol to satisfy additional properties such as *round-by-round soundness* [CCH+19]. We will elaborate on these properties soon, once we discuss our specific approach.

Clearly, the BAD functions we can support using the above methodology are constrained by the functions for which we can construct CIH. The known CIH from standard assumptions are: bounded-depth polynomial size circuits from LWE [CCH+19, PS19], linear approximable relations from trapdoor hash functions [BKM20, DGI+19], and TC^0 circuits from sub-exponential DDH [JJ21]. At the very least, we thus require BAD to be efficiently computable.

Putting together the above, we obtain the following design principles for constructing an interactive protocol which is "compatible" with the CIH framework for Fiat-Shamir (w.r.t. known constructions of CIH):

1. The BAD function is efficiently computable.
2. For every $x \notin L$ and every α, the size of $\mathcal{B}$ is polynomially bounded.

In this work, we follow the Fiat-Shamir paradigm as well to obtain our main result. In the following, we start by discussing potential choices for an interactive protocol that meets our desired efficiency goals while still being compatible with the CIH framework.

Considerations for the interactive protocol. Since the Fiat-Shamir transformation does not reduce the communication complexity of the interactive protocol, our starting point needs to be a protocol where the total communication between the prover and verifier is much smaller than $O(km)$, where k denotes the number of instances, and m the length of a single witness. A natural candidate that satisfies our requirements is Killian's protocol for languages in NP [Kil92]. Specifically, it is a public-coin interactive protocol where the total communication between the prover and verifier is significantly smaller than the length of the witness. Thus by defining the following NP language, $L^{\otimes k} = \{(x_1, \ldots, x_k) : \forall i \in [k], x_i \in L\}$, Killian's protocol gives us a public coin interactive argument with total communication significantly smaller than $O(km)$. Unfortunately, a recent work of [BBH+19] established non-trivial barriers towards instantiating the hash function in the Fiat-Shamir transformation applied to Killian's protocol.

There is in fact a broader point to consider: Kilian's protocol is an *argument*, i.e. its soundness holds only against computationally bounded cheating provers. In general, successful applications of the Fiat-Shamir paradigm when used in conjunction with CIH, have been largely restricted to interactive *proofs*,

[2]For notation convenience, we drop the subscript for $\mathcal{B}$.

where the soundness holds against computationally unbounded cheating provers. Intuitively, this is because $\mathcal{B}$, as defined in Equation 1, does not capture the computational resource bounds of a cheating prover. Specifically, $\mathcal{B}$ may contain exponentially many elements but does not capture the fact that for a computationally bounded cheating prover, finding the γ corresponding to $\beta \in \mathcal{B}$ is intractable. And as we have already outlined above, we need $\mathcal{B}$ to be of polynomial size. In fact, there are examples of certain interactive arguments that are not sound on the application of the Fiat-Shamir transformation (see e.g. [Bar01, GK03]).

Given the above state of affairs, the natural approach is to consider public coin interactive batch *proofs* for NP that achieve the same succinctness properties as (non-interactive) BARGs. Presently, however, interactive batch proofs are only known for the class UP, a subset of NP for which there is exactly one witness of membership for each statement [RRR16, RRR18, RR20]. Indeed, constructing such proofs for NP is an open problem.

2.2 Dual-mode interactive batch arguments

We therefore deviate from the above approach and instead define and construct a primitive we call *dual-mode interactive batch arguments*. Intuitively, these are interactive *arguments* in the common reference string (CRS) model, where the CRS can be generated in two computationally indistinguishable modes - (1) *normal mode*; and (2) *trapdoor mode*. We require that in the trapdoor mode, the protocol is sound against all (possibly unbounded) cheating provers; however, in the normal mode, it only achieves computational soundness.

This gives us the best of both worlds – we bypass the problem of constructing interactive batch proofs, but still retain the possibility of applying the Fiat-Shamir transform to the protocol when it is executed in the trapdoor mode (without running into the issues that arise for arguments). In order to apply the Fiat-Shamir transform, we require some additional properties from dual mode interactive batch arguments: specifically, we require such protocols to be *Fiat-Shamir friendly*, a notion we will elaborate on shortly.

We present a dual-mode interactive batch argument system for proving multiple instances of the NP-complete problem R1CS. An R1CS instance $\mathbb{x}$ is defined to be the tuple $\mathbb{x} := (A, B, C, \mathrm{io}, m)$, where io denotes the public input and output of the instance, and $A, B, C \in \{0,1\}^{m\times m}$ are matrices. We say that a vector $w \in \{0,1\}^{m-|\mathrm{io}|-1}$ is a witness for $\mathbb{x}$ if $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$, where $z = (\mathrm{io}, 1, w)$, $\cdot$ is the matrix-vector product, and $\circ$ is the Hadamard (entry-wise) product.[3]

Background: Spartan Protocol. Our starting point is the Spartan protocol [Set20] which proves the satisfiability of a *single* R1CS instance $\mathbb{x}$ with total communication sub-linear in the witness size $|w|$, i.e. the protocol is succinct. The Spartan protocol is defined over a field $\mathbb{F}$, such that $\log|\mathbb{F}| \approx \lambda$, and follows roughly the structure described below:

1. The prover first computes a commitment c to the witness w, that it sends to the verifier. In order to achieve communication succinctness, $|c|$ must be sub-linear in m. (We shall see below that the commitment scheme needs to satisfy some additional properties.)

2. The verifier then sends a random element element $\tau \in \mathbb{F}^s$, where s is such that $m = 2^s$.

3. It was shown in [Set20] that with probability $s/|\mathbb{F}|$ over the choice of τ, any R1CS instances can then be reduced to the following check:

$$\sum_{x\in\{0,1\}^s} \mathcal{G}_{\mathrm{io},w,\tau}(x) = 0, \tag{2}$$

[3]R1CS instances are more generally defined over a field, but for this overview we will consider them over $\mathbb{F}_2$ (or $\{0,1\}$). An instance of Boolean circuit satisfiability (C-SAT), defined by a circuit C can be transformed to an R1CS instance where $m \approx |C|$. See Section A.2 for details on the transformation.

where $\mathcal{G}_{\text{io},w,\tau} : \mathbb{F}^s \mapsto \mathbb{F}$ is a polynomial with degree 3 in each variable, and is determined entirely by $\mathbb{x}$, the witness w and τ. For the purpose of our discussion, the exact form of the polynomial is not immediately relevant. Note that without the witness w, the verifier does not have a representation of $\mathcal{G}_{\text{io},w,\tau}$, but we shall see shortly that it doesn't matter.

The above check is precisely the scenario where the *sumcheck protocol* [LFKN92, Sha92], an interactive protocol between a prover and verifier, is useful. In the sumcheck protocol, the prover is attempting to convince the verifier of the claim $\sum_{b_1,\cdots,b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell) = v$, where $g : \mathbb{F}^s \mapsto \mathbb{F}$ is an s variate polynomial of degree at most d in each variable, and $v \in \mathbb{F}$ is a publicly known value. The resultant interactive protocol is an s round public coin *proof* where the prover sends $O(d \cdot s)$ field elements. Importantly the verifier is only required to evaluate g at a *single point* $r^* \in \mathbb{F}^s$ at the end of the protocol, where r^* determined solely by the verifier's randomness in sumcheck protocol.

4. The prover and verifier run the sumcheck protocol for Equation 2, at the end of which verifier needs to evaluate $\mathcal{G}_{\text{io},w,\tau}(\cdot)$ at the point r^* (and compare against some value determined by the sumcheck protocol). But since the verifier does not have access to $\mathcal{G}_{\text{io},w,\tau}(\cdot)$, it asks the prover to send relevant information so that it can complete the check. Since the Spartan protocol requires this message from the prover to be succinct, the prover cannot send w in the clear.

   Fortunately, it turns out that the value that the prover needs to send is simply a linear combination of the bits of w (see Section A.3 for details), where the linear coefficients are determined entirely by A, B, C, τ and r^* i.e. let $\sum_{i \in [m]} \sigma_i \cdot w_i$ be the corresponding linear combination where the coefficients σ_i are known to both the prover and verifier, and are even independent of io.[4]

5. The prover now opens the commitment c to $\sum_{i \in [m]} \sigma_i \cdot w_i$ such that the opening is succinct. This allows the verifier to complete its check.

Spartan provides various instantiations for the commitment scheme satisfying the above properties, where the commitment opening is an interactive protocol. The resulting protocol is computationally sound.

The Spartan protocol does not satisfy our desired properties from a dual-mode interactive batch argument. However, it serves as a useful starting point for us. In Spartan, the goal was to have the total communication be sub-linear in m, while in the batch setting, we are fine with total communication proportional to a *single witness*. This in turn means that we can consider commitment schemes where the commitment size is proportional to a single witness. Let us now see how we can use this insight to adapt the Spartan protocol to both make it suitable for batch verification, and achieve the notion of dual-mode batch arguments.

Our Construction. We now discuss the main steps in our interactive protocol, while highlighting the differences from the above discussion. We want to batch prove k instances $\{\mathbb{x}^{(j)}\}_{j \in [k]}$ where the matrices A, B and C are the same across all instances, and only the public input-output io varies across the instances. The reader may view this as multiple instances with the same relation circuit, but different statements. The description of the protocol now follows.

1. To commit to a batch of witnesses $\{w^{(j)}\}_{j \in [k]}$, we follow the batch commitment strategy in [RRR16]: arrange the witnesses as rows of a $k \times m$ matrix, and commit to the column of each matrix, i.e. $\forall i \in [m]$, $c_i \leftarrow \mathsf{Com}(w_i^{(1)}, \ldots, w_i^{(k)})$. If the k-tuple commitment has size $O(\lambda)$, then the total commitment is of size $\widetilde{O}(m)$, ignoring polynomial factors in $O(\lambda)$.

   This indicates that our *commitment scheme must allow us to commit to the k-tuple succinctly.*

[4]Strictly speaking, the prover needs to send 3 separate linear combinations of the witness, but we ignore this here for simplicity.

2. Given that each instance has a different statement io and witness w, each of the k instances define a different polynomial, giving rise to the k polynomials $\{\mathcal{G}^{(j)}_{\mathrm{io},w,\tau}\}_{j\in[k]}$. The prover and verifier then run k sumcheck protocols in parallel with the *same verifier randomness*. As discussed earlier, at the end of the sumcheck protocols, the verifier needs to evaluate each of these polynomials at points $r^{*(j)}$ determined solely by the verifier's randomness in the sumcheck protocol.

   Since the verifier uses the same randomness across all instances of the sumcheck protocol execution, the polynomials need to be evaluated at the *same point* r^*. Additionally, since the linear coefficients depend only on A, B, C, τ and r^*, this in turn implies that the linear coefficients for all the witnesses $w^{(j)}$ are the same: $(\sigma_1, \cdots, \sigma_m)$.

3. As in Spartan, the prover now needs to send $\sum_{i\in[m]} \sigma_i w_i^{(j)}$ to the verifier. For convenience, this can be be re-written as sending the k-tuple, $\sum_{i\in[m]} \sigma_i \cdot (w_i^{(1)}, \cdots, w_i^{(k)})$, where $\cdot$ indicates component-wise multiplication.

   If our commitment scheme satisfies linear homomorphism, i.e.

$$\mathsf{Com}(\sum_{i\in[m]} \sigma_i \cdot (w_i^{(1)}, \cdots, w_i^{(k)})) = \sum_{i\in[m]} \sigma_i \mathsf{Com}(w_i^{(1)}, \ldots, w_i^{(k)}),$$

   then it suffices for the prover to open to the commitment $\sum_{i\in[m]} \sigma_i c_i$.

   Thus our *commitment scheme must satisfy linear homomorphism (as described above), with the size of the opening proportional to the size of the underlying message.*

Let us go back to our requirement from dual-mode interactive batch arguments. For the protocol to achieve statistical soundness in the trapdoor mode, we need at the very least, the commitment to be statistically binding. However, this seems at odds with our succinctness requirements since we want the total number of bits sent to be significantly smaller than the size of the message committed.

Key Tool: Somewhere-Extractable Linearly Homomorphic Commitments. We resolve this issue by utilizing a commitment scheme in the CRS model, where the CRS is generated in one of two *computationally indistinguishable ways* - (1) *normal mode*; or (2) *extraction mode*. In the *extraction mode*, the CRS generation algorithm takes as input an index i^*, and additionally outputs an *extraction trapdoor* td that is not a part of the CRS. We require that the commitment of the k-tuple in the *extraction mode* for index i^*, is statistically binding at the i^*-th index of the commitment. Further, there is an efficient algorithm Ext such that given the the trapdoor td, Ext extracts the underlying message at the i^*-th index, and this holds even if the commitment was "malformed". Additionally, the extraction also satisfies *linear homomorphism*, i.e. $\sigma_1 \cdot \mathsf{Ext}(c_1, \mathsf{td}) + \sigma_2 \cdot \mathsf{Ext}(c_2, \mathsf{td}) = \mathsf{Ext}(\sigma_1 \cdot c_1 + \sigma_2 \cdot c_2, \mathsf{td})$. The linear homomorphism property of extraction ensures that once we extract from the commitments, the opening of the linear homomorphic evaluation can be computed solely from the linear coefficients - ensuring that the committer is bound to opening of the linear homomorphism.

If all m commitments are committed via the *extraction mode* CRS for index i^*, then the prover is statistically bound to $w^{(i^*)}$. Then intuitively, in the extraction mode, the security can be reduced to the soundness of the other components of the protocol for the i^*-th instance. The reduction to the check for polynomial $\mathcal{G}^{(i^*)}_{\mathrm{io},w,\tau}$ (via [Set20]), and the sumcheck protocol are both statistically sound, thereby satisfying overall statistical soundness. Thus, by setting the *trapdoor mode* (resp., normal model) CRS to be the extraction mode (resp., normal mode) CRS of the commitment scheme, we obtain a dual-mode interactive batch argument. Note the added syntax for the *trapdoor mode* of the dual-mode interactive batch argument - it takes in as input an index i^*, and generates a trapdoor td (not be included in the CRS).

We now summarize our requirements of the commitment scheme from the above discussion:

1. Commitment scheme for k-tuples in the CRS model, with indistinguishable methods of generating the CRS - *normal mode* or *extraction mode* such that the commitment is statistically binding at the i^*-th index when the CRS is generated in the extraction mode on input i^*.

2. Efficient extraction of the message at i^*-th index in the extraction mode, given the trapdoor td.

3. The commitment should allow for linear homomorphism (even over the extracted values).

4. The commitment should be succinct, while the opening should depend only on the size of the committed message.

We refer to such commitments as *somewhere-extractable linearly homomorphic commitments*. Our notion is similar to the notion of somewhere statistically-binding hash functions [HW15], but requires some additional properties. Later, in Section 2.4, we describe our construction of such commitment schemes based on the quadratic residuosity assumption. For now, we will simply assume that such commitment schemes exist.

One point of note is that since the CRS of the commitment scheme requires the index of the statement we want to prove soundness for, we can only achieve *non-adaptive security*. We will later show in the technical sections that this is in some sense the best that one can hope for.

Costs. From the description of the protocol, the communication cost for the commitment (and its opening) is $\widetilde{O}(m)$, while the communication cost from k sumcheck protocols is $\widetilde{O}(ks) = \widetilde{O}(k \log m)$, giving us a total communication cost of $\widetilde{O}(m + k \log m)$.

2.3 Fiat-Shamir compatibility

As we have alluded to before, constructing a dual-mode interactive batch argument is an important first step towards a non-interactive protocol. But by itself, it is not enough. We need to show that our constructed protocol in Fiat-Shamir friendly. This has been recently formalized by [JKKZ21] as the notion of *Fiat-Shamir* (FS) *compatibility*, that extends our earlier discussion in Section 2.1 on the relationship between CIH and the Fiat-Shamir transform.

Let the prover's i-th message in the protocol be denoted by α_i, while the corresponding verifier message by β_i. The protocol transcript trans_i is defined to be $\mathsf{trans}_i := (\alpha_1, \beta_1, \cdots, \alpha_i, \beta_i)$, which collects all messages up to (and including) the i-th round messages. An interactive proof is said to be FS compatible if it follows the following two properties:

Round-by-round soundness: There is a function State that takes as input the statement x, and a transcript prefix $\mathsf{trans}_i := (\alpha_1, \beta_1, \cdots, \alpha_i, \beta_i)$, and outputs Accept or Reject. We require some additional properties from State: for every $x \notin L$, $\mathsf{State}(x, \emptyset) = \mathsf{Reject}$, and for every full transcript trans the verifier rejects if $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$. Perhaps, most importantly, we require that if $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$, then for *any* prover message α, $\mathsf{State}(x, \mathsf{trans}|\alpha|\beta) = \mathsf{Reject}$ with overwhelming probability over the choice of β.

Efficient BAD **function:** For every $x \notin L$, when $\mathsf{State}(x, \mathsf{trans}_i) = \mathsf{Reject}$, we require an efficiently computable function BAD[5] that outputs the "bad" verifier challenges β that will result in State switching output to Accept, i.e. if $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$, then $\mathsf{BAD}(x, \mathsf{trans}|\alpha)$ outputs a uniformly random element from the set $\mathcal{B}$ defined as

$$\mathcal{B} := \left\{\beta \,\middle|\, \mathsf{State}(x, \mathsf{trans}|\alpha|\beta) = \mathsf{Accept}\right\}.$$

[5]Unlike the definition in [JKKZ21], we will require any non-uniform advice to the BAD function to also be *efficiently* computable.

From our earlier design principles, we require the size of the set $\mathcal{B}$ to be polynomially bounded.

Before we proceed, let's recall our discussion from Section 2.1 on Fiat-Shamir and CI-hash functions. It is easy to see that the discussion there also applies here - as long as we can construct a CIH $\mathcal{H}$ that is CI for the circuits computing BAD, then the Fiat-Shamir transformed protocol with respect to $\mathcal{H}$ is sound.

We know of the following CI-hash functions based on standard assumptions[6]:

– CI for all a priori polynomially bounded circuits assuming LWE [PS19]; and

– CI for all of TC^0 assuming sub-exponential security of DDH [JJ21].

We want to be able to leverage both of these constructions for our final non-interactive batch argument. Since the size (and depth) of the circuit computing BAD directly corresponds to the functions for which we need CI, to achieve a result based on sub-exponential security of DDH, we need to show that the function BAD can be computed in TC^0. We call such protocols to be *strongly* FS *compatible*.

Let us now demonstrate that our dual-mode protocol in the trapdoor mode is FS-compatible. Recall that in the trapdoor mode an index i^* is specified, and we shall prove FS compatibility when $\mathbb{x}^{(i^*)} \notin L_{\mathsf{R1CS}}$. This is sufficient, since for a batch instance to be false, there is at least one index j such that $\mathbb{x}^{(j)} \notin L$. In particular, this allows us to ignore the other sumcheck executions while establishing FS compatibility. We will further show that BAD can also be computed in TC^0. Since we only focus on a single instance $\mathbb{x}^{(i^*)}$, in what follows, we skip the index i^* for the instance to simplify notation.

Round-by-Round soundness. The verifier messages can be split into two cases: (a) $\tau \in \mathbb{F}^s$; (b) verifer messages inside the sumcheck protocol. We only sketch here the main ideas and refer the reader to the technical sections for more details as our primary focus will be on the construction of the BAD function.

For the sumcheck, we rely on [JKKZ21] that already establishes the sumcheck protocol to be round-by-round sound. The main difference is that [JKKZ21] requires full knowledge of the polynomial over which the sumcheck is computed. In our setting, however, the polynomial $\mathcal{G}_{\mathrm{io},w,\tau}$ is (partially) determined by the witness, which is sent within the commitment. We resolve this issue by using the trapdoor td to extract the i^*-th witness and compute the polynomial, since the CRS was generated in the trapdoor mode for i^*. For the verifier message τ, we can rely on the Theorem underlying Spartan [Set20] that shows that any R1CS instance $\mathbb{x}$ can be reduced to the sum $\sum_{x\in\{0,1\}^s} \mathcal{G}_{\mathrm{io},w,\tau}(x) = 0$ other than with probability $s/|\mathbb{F}|$ over the choice of τ. The actual State computation for τ will be elaborated upon in the BAD function computation below.

Efficient BAD function. As described above, verifier messages can be split into two cases. From the definition of the BAD function, it suffices to build two separate functions, one for each cases. Let's start with the simpler case of the sumcheck verifier messages.

Sumcheck BAD *function:* In the sumcheck protocol, for each round $i \in [s]$, the prover sends a univariate polynomial $g_i^* : \mathbb{F} \mapsto \mathbb{F}$ of degree 3 to the verifier. If computed correctly, it should correspond to the polynomial g_i, defined as

$$g_i(x) := \sum_{x_{i+1},\cdots,x_s\in\{0,1\}} \mathcal{G}_{\mathrm{io},w,\tau}(\beta_1,\cdots,\beta_{i-1},x,x_{i+1},\cdots,x_s).$$

The set of bad challenges in the i-th round are the verifier challenges β_i such that both polynomials g_i and g_i^* evaluate to the same value on β_i, i.e. $\mathcal{B} := \{\beta_i \mid g_i(\beta_i) = g_i^*(\beta_i)\}$. Alternatively $\mathcal{B}$ consists of the roots of the polynomial $g_i - g_i^*$. Since $\mathcal{G}_{\mathrm{io},w,\tau}$ is a polynomial that is degree 3 in each variable, $|\mathcal{B}| \le 3$.

[6]We note that the CI-hash function constructed in [BKM20] is also based on standard assumptions, but the class of functions that it supports (i.e. class it is CI for) is very small, and therefore limits its applicability.

Unlike [JKKZ21], which demonstrate BAD function for the general sumcheck, we focus on the setting where the *true* polynomial g_i can be computed in polynomial time (e.g. $s = O(\log \lambda)$). Thus on input, $(\mathbb{x}, \mathsf{trans}_{i-1}|\alpha_i)$, BAD (i) parses α_i as the polynomial g_i^*; (ii) computes the *true* polynomial g_i, using the trapdoor first to extract w and determine $\mathcal{G}_{\mathrm{io},w,\tau}$; and (iii) use a polynomial time algorithm like Cantor-Zassenhaus to compute the (three) roots of $g_i - g_i^*$, and output one at random.

τ BAD *function:* To describe the BAD function corresponding to τ, we need to look at the polynomial $\mathcal{G}_{\mathrm{io},w,\tau}$ implied by [Set20] (Theorem 2). So far we have focused on $\mathcal{G}_{\mathrm{io},w,\tau}(x)$ as a polynomial over the variables x, with $\tau \in \mathbb{F}^s$ fixed. Let us now focus on the same polynomial over both x and τ, i.e. for every τ, $\mathcal{G}_{\mathrm{io},w,\tau}(x) = \mathcal{G}'_{\mathrm{io},w}(x,\tau)$. In fact [Set20] showed that $\mathcal{G}'_{\mathrm{io},w}(x,\tau)$ is a polynomial over $x_1, \cdots, x_s$ and $\tau_1, \cdots, \tau_s$ that has degree 1 in each τ_i (see Section A.3 for details). Thus, we can rewrite $\sum_{x\in\{0,1\}^s} \mathcal{G}_{\mathrm{io},w,\tau}(x)$ as a polynomial over $\tau_1, \cdots, \tau_s$. Specifically, let

$$Q(\tau) := \sum_{x\in\{0,1\}^s} \mathcal{G}'_{\mathrm{io},w}(x,\tau),$$

where Q is a polynomial over s variables $\tau_1, \cdots, \tau_s$, with degree 1 in each τ_i. Note that as in the case of $\mathcal{G}_{\mathrm{io},w,\tau}$, Q is determined by the witness w that only the prover has access to. For $\mathbb{x}$ and w such that $\mathcal{R}_{\mathsf{R1CS}}(\mathbb{x}, w) = 1$, the correctly computed polynomial $Q_{\mathrm{io},w}$ is the zero polynomial, i.e. $Q_{\mathrm{io},w} \equiv 0$. The random $\tau \in \mathbb{F}^s$, sent by the verifier is to test whether $Q(\tau) = 0$. If $Q \not\equiv 0$, then by the Schwartz-Zippel lemma, $Q(\tau) = 0$ with probability at most $s/|\mathbb{F}|$ over the choice of τ, which is negligible in λ for our choice of $\mathbb{F}$. This suggests the following strategy for BAD, when $Q \not\equiv 0$, let $\mathcal{B} := \{\tau \in \mathbb{F}^s \mid Q(\tau) = 0\}$. BAD then works as follows: (i) uses the trapdoor td to first extract w and determine Q; and (ii) solve for τ from $\mathcal{B}$ and output a random such τ.

While this appears to work on the surface, on closer inspection it can be observed that while the Schwartz-Zippel lemma guarantees the probability to be at most $s/|\mathbb{F}|$, the size of the set $\mathcal{B}$ can be exponential ($|\mathcal{B}| \approx |\mathbb{F}^{s-1}|$). As indicated by our design goals at the start, this is undesirable and something we do not know how to work around.

We take an alternate approach. Instead of using a single hash function that outputs the vector $\tau \in \mathbb{F}^s$, we consider a sequence of hash functions $(h_1, \cdots, h_s)$ that each output a single τ_i. Specifically, for every i, $\tau_i := h_i(\mathbb{x}, \tau_1, \cdots, \tau_{i-1})$.

Let $Q|_{\tau_1^*,\cdots,\tau_{i-1}^*}$ be the polynomial Q with the first $i-1$ variables fixed to be values $\tau_1^*, \cdots, \tau_{i-1}^*$. If $Q \not\equiv 0$, then we want it to continue to be the case that for the prefix $\tau_1^*, \cdots, \tau_{i-1}^*$, $Q|_{\tau_1^*,\cdots,\tau_{i-1}^*} \not\equiv 0$. This then lets us define the i-th bad set $\mathcal{B}_i$ when $Q|_{\tau_1,\cdots,\tau_{i-1}} \not\equiv 0$,

$$\mathcal{B}_i := \left\{\tau \in \mathbb{F} \mid \; Q|_{\tau_1,\cdots,\tau_{i-1},\tau} \equiv 0\right\}.$$

Before we describe the BAD function, let us take a moment to see how one determines whether $Q|_{\tau_1,\cdots,\tau_{i-1},\tau} \equiv 0$. This corresponds to all coefficients of the said polynomial to be 0. At a high level, from the description of Q, the coefficients are determined by the sum over $m = 2^s$ values, which in turn is computable in polynomial time as $m = \mathsf{poly}(\lambda)$. Then a bad τ simply corresponds to those elements in $\mathbb{F}$ that result in the coefficients becoming 0. Since the polynomial is *linear* in each variable, solving for such a τ corresponds to solving a linear system in $\mathbb{F}$. Correspondingly, for all i, the set $\mathcal{B}_i$ is of bounded polynomial size. We refer the reader to Section 6 for more details on these steps.

We are finally in a position to describe the BAD function, which on input $(\mathbb{x}, \tau_1, \cdots, \tau_{i-1})$ (note that the prover message is empty) does the following: (i) use the trapdoor td to first extract w and determine Q, and then correspondingly $Q|_{\tau_1,\cdots,\tau_{i-1},\tau}$; and (ii) solve the linear equation in τ such that $Q|_{\tau_1,\cdots,\tau_{i-1},\tau} \equiv 0$, and output such a τ if it exists.

From our discussions above, BAD is in fact efficiently computable, and thus satisfies our requirement.

BAD **has low depth.** To base our non-interactive protocol on CIH for TC^0, we need to demonstrate that the BAD function for both cases can be computed in TC^0. In contrast to when we established that BAD was efficient, here, the simpler case is the BAD function for τ. But before we proceed, we note that in both cases, we require trapdoor extraction, and thus we additionally require *low-depth extraction* property from our commitment scheme. We proceed with our discussion assuming this to be the case, and will provide more details when discussing out construction of the commitment scheme in Section 2.4.

τ BAD *function:* In the above description, we are only solving linear equations in $\mathbb{F}$, which can be computed in TC^0, thus trivially giving us the required property.

Sumcheck BAD *function:* Unfortunately, things are not so simple for the BAD function in the sumcheck case. The BAD function as described, needs to compute a root of a degree 3 polynomial in $\mathbb{F}$. While we do know how to do this in polynomial time, for computing roots in low depth, we are only aware of root finding for degree 2 polynomials in $\mathbb{F}$ to be in TC^0.

To circumvent this issue, we take a closer look at the polynomial $\mathcal{G}_{\text{io},w,\tau}$[7]. As described in Section A.3, it turns out that $\mathcal{G}_{\text{io},w,\tau}$ is of a special form, where we compute a sumcheck protocol for,

$$\sum_{x\in\{0,1\}^s}\mathcal{G}_{\text{io},w,\tau}(x)=\sum_{x\in\{0,1\}^s}f_{\text{io},w,\tau}(x)\left(\prod_{j=1}^{s}h_{j,\tau}(x_j)\right)=0,$$

where f is a polynomial with individual degree 2, and each $h_{j,\tau}$ is a univariate polynomial in x_j with degree 1. Moreover, the coefficients of $h_{i,\tau}$ are determined only by τ, and therefore known to the verifier once it samples τ. This suggests a slight modification of the sumcheck polynomial, where the prover in the i-th round sends the degree 2 polynomial $g^{*'}$ to the verifier which it has purportedly computed as,

$$g_i'(x)=\sum_{x_{i+1},\cdots,x_s\in\{0,1\}}f_{\text{io},w,\tau}(\beta_1,\cdots,\beta_{i-1},x,x_{i+1},\cdots,x_s)\left(\prod_{j=1}^{i-1}h_{j,\tau}(\beta_j)\right)\left(\prod_{j=i+1}^{s}h_{j,\tau}(x_j)\right).$$

The verifier then locally computes $h_{i,\tau}$, and computes g_i, Clearly, the three roots of the polynomial $g_i^*-g_i$ consist of the two roots of $g_i^{*'}-g_i'$ and the root of $h_{i,\tau}$. Thus, by modifying the sumcheck protocol as suggested above, we can then reduce the root computation in BAD to computation of roots for a degree 2 polynomial, and a degree 1 polynomial, both of which we can compute in TC^0.

This establishes that our dual-mode protocol in the trapdoor mode is strongly FS-compatible. [JKKZ21] demonstrate that the Fiat-Shamir transformation with respect to $\mathcal{H}$ for any FS-compatible protocol is sound as long as $\mathcal{H}$ is CI for polynomial size functions (larger than BAD). We extend their proof to demonstrate that if we strengthen FS compatibility to strong FS compatibility, it suffices for $\mathcal{H}$ to be CI for TC^0.

Next, we show how to leverage our protocol to construct a non-interactive batch argument (BARG).

Going from FS-Compatibility to BARGs. In this final step, we finally construct our non-interactive arguments. We apply the Fiat-Shamir transform to the dual-mode interactive batch argument to achieve a publicly verifiable non-adaptive BARG in the CRS model. For soundness of the transform we rely on (i) *mode indistinguishability* property of the protocol to switch to the trapdoor mode; and (ii) then in the *trapdoor mode*, we rely on the FS-compatibility that we have discussed above.

Communication sub-linear in k. The above construction has an additive term that is linear in k (recall that the communication cost is $\widetilde{O}(m+k\log m)$). We describe how one can generically make this sub-linear by using fairly standard techniques. The idea is to simply batch k_1 instances into a larger instance of the

[7]For simplicity, we focus on a single polynomial here as our explanation extends to the batch setting too.

language $L^{\otimes k_1} := \{(x_1, \ldots, x_{k_1}) : \forall i \in [k_1], x_i \in L\}$ that has a relation circuit of size $k_1|C| + k_1$, where $|C|$ is the size of the underlying relation circuit. Then we apply our dual-mode batch argument for k/k_1 instances of $L^{\otimes k_1}$. By setting $k_1 \approx O(\sqrt{k})$, we get communication that is sub-linear in k. Note that from our earlier discussion $m \approx k_1|C| + k_1$.

2.4 Somewhere-Extractable Linearly Homomorphic Commitment

We now finally describe our construction of the somewhere-extractable linearly homomorphic commitment scheme. Over the course of the above discussion, we have accumulated various requirements that our commitment scheme must satisfy. We describe a construction that achieves these properties based on the *quadratic residuosity* (QR) assumption.

We start by focusing on the simpler goal of constructing a somewhere statistically binding commitment scheme building on ideas from the recent work on trapdoor hash functions [DGI+19].[8] We will discuss how to achieve the extraction and linear homomorphism properties later.

Recall that, for any Blum integer $N = p \cdot q$, where p, q are primes such that $p \pmod 4 = q \pmod 4 = 3$, we denote $\mathbb{Z}_N^*$ as the multiplicative group modulo N, and $\mathbb{J}_N$ as the subgroup of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$, and $\mathbb{QR}_N$ be the subgroup of quadratic residues. Let $\mathbb{H} = \{-1, +1\}$ also be a multiplicative group, then $\mathbb{J}_N = \mathbb{H} \times \mathbb{QR}_N$. We now describe the commitment scheme:

- The *trapdoor* mode commitment key for the coordinate i^* consists of two arrays of group elements.

$$\begin{bmatrix} \mathbf{g} \\ \mathbf{h} \end{bmatrix} = \begin{bmatrix} g_1 & g_2 & \cdots & g_{i^*} & \cdots & g_k \\ g_1^s & g_2^s & \cdots & -g_{i^*}^s & \cdots & g_k^s \end{bmatrix},$$

  where $s \leftarrow \lfloor (N-1)/2 \rceil$ is sampled uniformly at random, and the elements of the second row are the corresponding first row elements raised to the exponent s, except that we flip the sign on the i^*-th coordinate.

  In the *normal* mode, we do not flip the sign, i.e. let $\mathbf{h} = (\mathbf{g})^s$. The mode indistinguishability relies on the quadratic residuosity assumption[9].

- To commit to a vector $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ of length k, we compute $(c_g = \prod_{i=1}^k g_i^{x_i}, c_h = \prod_{i=1}^k h_i^{x_i})$. Then $c_h = c_g^s \cdot (-1)^{x_{i^*}}$. Hence, x_{i^*} is statistical binding. Furthermore, the commitment size is compact, since it only contains two group elements.

Linear Homomorphism and Extraction. We now discuss how to achieve the desired extraction and the linear homomorphism properties. We observe that the commitment described above is essentially an encryption of x_{i^*}. Hence, one can use the trapdoor $\mathsf{td} = (p, q, s)$ to extract x_{i^*}. The linear homomorphism works as follows: if we denote the commitment of $\mathbf{x}$ under the key $(\mathbf{g}, \mathbf{h})$ as $(\mathbf{g}^{\mathbf{x}}, \mathbf{h}^{\mathbf{x}})$, then for any two commitments $(\mathbf{g}^{\mathbf{x}}, \mathbf{h}^{\mathbf{x}})$, $(\mathbf{g}^{\mathbf{y}}, \mathbf{h}^{\mathbf{y}})$, and any integers $a, b \in \mathbb{Z}$, we can compute

$$\left((\mathbf{g}^{\mathbf{x}})^a \cdot (\mathbf{g}^{\mathbf{y}})^b = \mathbf{g}^{a \cdot \mathbf{x} + b \cdot \mathbf{y}}, \quad (\mathbf{h}^{\mathbf{x}})^a \cdot (\mathbf{h}^{\mathbf{y}})^b = \mathbf{h}^{a \cdot \mathbf{x} + b \cdot \mathbf{y}} \right),$$

which is exactly the commitment for $a \cdot \mathbf{x} + b \cdot \mathbf{y}$.

However, if we use the above commitment scheme for our application to batch arguments, we face the following challenge: the field operation needs modulo 2 computation, but the honest prover can not hope to perform such computation, since the homomorphic operation is over $\mathbb{Z}$.

[8] Similar ideas have also been used in the constructions of somewhere statistically-binding hash functions [HW15, KLW15, OPWW15] and hash encryption schemes [DG17, DGHM18, BLSV18, GH18].

[9] The mode indistinguishability follows from the Theorem 4 from [BG10], which relies on the quadratic residuosity assumption.

To overcome this issue, we have the honest prover do all the operations over the polynomial ring $\mathbb{Z}[\alpha]$, instead of the field $\mathbb{F}$. Note that this modification does not affect completeness since the honest prover is essentially proving some polynomial identities (e.g. the R1CS instance $(A \cdot z) \circ (B \cdot z) = C \cdot z$ reduced from circuit satisfiability), and such identities hold regardless of whether the underlying variables are taken from a field or a ring. For soundness, we make the following observation: if a proof is accepted over the ring $\mathbb{Z}[\alpha]$, then if we further perform modulo 2 operation, the proof must still be accepted. Hence the soundness can be reduced to the case when operations are over $\mathbb{F}$. See Section 5 for a more detailed discussion.

(Linearly Homomorphic) Extraction from *any* Commitment. The aforementioned extraction and linear homomorphism only works for "well-formed" commitments. In order to prove round-by-round soundness of our dual-mode interactive batch argument protocol, however, we need the extraction works for *any* (possibly not well-formed) commitment. Moreover, the linear homomorphism property must also hold over the extracted values.

To achieve such a property, we observe that for any (possibly not well-formed) commitments $c = (c_g, c_h) \in \mathbb{J}_N \times \mathbb{J}_N$, we can still compute c_h/c_g^s, which is also a group element in $\mathbb{J}_N$. From the decomposition $\mathbb{J}_N = \mathbb{H} \times \mathbb{QR}_N$, there exists a unique $m \in \mathbb{Z}_2$ and $g \in \mathbb{QR}_N$ such that $c_h/c_g^s = (-1)^m \cdot g$. Hence, we define the extracted message for c as m. Since N is a Blum integer, $|\mathbb{QR}_N| = (p-1)/2 \cdot (q-1)/2$ is an odd number. We let n denote $|\mathbb{QR}_N|$. Then, we extract m by computing

$$(c_h/c_g^s)^n = (-1)^m \cdot g^n = (-1)^m.$$

We show that this extraction can be decomposed to an off-line pre-precompution phase and an online extraction phase, where the online extraction can be computed in TC^0. We allow the off-line pre-precomputation to be deeper than TC^0 circuits, since in our protocol, the pre-computation is always performed honestly by the prover and the verifier.

We now show that the linear homomorphism property also holds for the above extraction algorithm. For any two commitments $c = (c_g, c_h), d = (d_g, d_h)$, the extraction is a "linear operation" over c_g, c_h, i.e. if $\mathsf{Ext}(c, \mathsf{td}) = m_c$, then $(-1)^{m_c} = c_h^n c_g^{-sn}$. Similarly, if $\mathsf{Ext}(d, \mathsf{td}) = m_d$, then $(-1)^{m_d} = d_h^n d_g^{-sn}$. Now for any linear combination $a, b \in \mathbb{Z}$, when we extract from $a \cdot c + b \cdot d$, we compute $(c_h^a \cdot d_h^b)^n \cdot (c_g^a \cdot d_g^b)^{-sn} = (-1)^{a \cdot m_1 + b \cdot m_2}$. Hence, the extracted value for $a \cdot c + b \cdot d$ is $a \cdot m_1 + b \cdot m_2 \pmod 2$, which establishes the linear homomorphism property.

For more details, see Section 4.3.

2.5 Roadmap to the Paper

We start by defining necessary preliminaries in Section 3 before defining, and constructing, somewhere-extractable linearly homomorphic commitment in Section 4. Next in Section 5 we define and construct dual mode interactive batch arguments before demonstrating that our construction is Fiat-Shamir friendly in Section 6. Finally in Section 7 we put it all together and provide a construction of non-interactive batch arguments (BARGs).

3 Preliminaries

In this section we provide definitions and notations that will be used throughout the paper. Due to a lack of space, additional preliminaries have been deferred to Section A.

We start with some basic notation: For any n length string a, we denote by a_i the i-th position of the string. Often we will see i represented in the binary form, i.e. $i \in \{0,1\}^{\lceil |x| \rceil}$, in such a scenario we simply convert i to its integer representation to index into the string a. To concatenate two strings a and b, we denote it as (a, b). Lastly, we will consider matrices of the form $A \in \mathbb{F}^{m\times n}$, which we shall view as functions $A : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^{\lceil \log n \rceil} \mapsto \mathbb{F}$, where $A(i, j)$ corresponds to the element in A along the i-th row, and j-th column.

3.1 Complexity Problems

We define below the two relevant complexity problems, *Boolean circuit satisfiability* (C-SAT) and *satisfiability of systems of rank-1 quadratic equations over a finite field* $\mathbb{F}$ (R1CS). Our starting point will be C-SAT instances, but our protocol will be designed for R1CS instances.

Definition 1 (Circuit-C-SAT). *A circuit satisfiability instance* C-SAT *is a tuple* (C, x)*, defined by a Boolean circuit* $C : \{0,1\}^{|x|} \times \{0,1\}^{|y|} \mapsto \{0,1\}$ *and a string* $x \in \{0,1\}^{|x|}$.

A C-SAT *instance is said to be* satisfiable *if there exists a string* $y \in \{0,1\}^{|y|}$ *such that* $C(x, y) = 1$*. We denote this as* $\mathcal{R}_{\text{C-SAT}}((C, x), y) = 1$.

Definition 2 (R1CS). *An* R1CS *instance is a tuple* $\mathbb{x} = (\mathbb{F}, A, B, C, io, m, n)$ *where (a) io denotes the public input and output of the instance; (b)* $A, B, C \in \mathbb{F}^{m\times m}$ *with* $m \geq |io| + 1$*; and (c) there are at most n non-zero entries in each matrix.*

An R1CS *instance is said to be* satisfiable *if there exists a witness* $w \in \mathbb{F}^{m-|io|-1}$ *such that* $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$*, where* $z = (io, 1, w)$*,* $\cdot$ *is the matrix-vector product and* $\circ$ *is the Hadamard (entry-wise) product. We denote this as* $\mathcal{R}_{\text{R1CS}}(\mathbb{x}, w) = 1$.

Circuit-SAT to R1CS. As discussed above, while our definition is for general R1CS instances, we shall consider instances generated via a reduction from C-SAT. Given a C-SAT instance, one can convert it into an R1CS instance over $\mathbb{F}$ where $A, B, C \in \mathbb{F}^{m\times m}$ for $m = O(|C|)$ and $n = O(|C|)$, i.e. the matrices A, B and C are sparse. Furthermore, io $\in \{0,1\}^{|x|}$ and the witness $w \in \{0,1\}^{|C|-|x|}$. For completeness, we present the transformation in Section A.2.

We will use the following theorem from [Set20] that shows that any R1CS instance can be represented by sum over the Boolean hypercube (i.e. over $\{0,1\}^{\ell}$ for some ℓ) of a low degree polynomial.

Theorem 2 ([Set20]). *For any* R1CS *instance* $\mathbb{x} = (\mathbb{F}, A, B, C, io, m, n)$ *there exists a degree 3,* $\log m$*-variate polynomial* $\mathcal{G}$ *such that*

$$\sum_{x\in\{0,1\}^{\log m}} \mathcal{G}(x) = 0$$

if and only if there exists a witness w such that, except with soundness error negligible in λ*,* $\mathcal{R}_{\text{R1CS}}(\mathbb{x}, w) = 1$*. Here* $|\mathbb{F}|$ *is exponential in* λ *and* $m = O(\lambda)$.

The construction of the above polynomial $\mathcal{G}$ will be important to our work, and we present the overview of the construction along with relevant notation in Section A.3.

3.2 Sumcheck

Fix any finite field $\mathbb{F}$, and consider the subset of $\mathbb{F}$, $\{0,1\}$. A (possibly inefficient) prover P_{SC} with input $g : \mathbb{F}^{\ell} \mapsto \mathbb{F}$, an ℓ variate polynomial of degree at most d in each variable, attempts to convince the verifier V_{SC} of the following claim

$$\sum_{b_1,\cdots,b_\ell\in\{0,1\}} g(b_1, \cdots, b_\ell) = v,$$

where $v \in \mathbb{F}$ is a common input to both P_{SC} and V_{SC}. The verifier is only given oracle access to g, denoted by $\mathsf{V}_{\mathsf{SC}}^{g}$, and is in fact allowed only a single query to g. The protocol to enable this is referred to as the sumcheck protocol, and is described in Figure 1.

Sumcheck Protocol: $(\mathsf{P}_{\mathsf{SC}}, \mathsf{V}^{g}_{\mathsf{SC}})$ **for** $\sum_{b_1,\cdots,b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell) = v$

Common input: $v \in \mathbb{F}$.

P's auxiliary input: polynomial g

1. Set $i := 1$, $v_0 := v$.
2. Prover P_{SC} computes the univariate polynomial $g_i : \mathbb{F} \mapsto \mathbb{F}$ of degree d

   $$g_i(x) := \sum_{b_{i+1},\cdots,b_\ell \in \{0,1\}} g(t_1, \cdots, t_{i-1}, x, b_{i+1}, \cdots, b_\ell)$$

   send g_i to the verifier $\mathsf{V}^{g}_{\mathsf{SC}}$.
3. Verifier $\mathsf{V}^{g}_{\mathsf{SC}}$ does the following:
   (a) Check if g_i is a univariate polynomial of degree at most d, and that $g_i(0)+g_i(1) = v_{i-1}$. If not, $\mathsf{V}^{g}_{\mathsf{SC}}$ rejects.
   (b) Choose a random element $t_i \xleftarrow{\$} \mathbb{F}$
   (c) Set $v_i := g_i(t_i)$.
   (d) if $i < \ell$, set $i := i + 1$, send t_i to prover P_{SC} and go back to Step 2.
   (e) if $i = \ell$, check if $v_\ell = g(t_1, \cdots, t_\ell)$ by querying the oracle g at the point $(t_1, \cdots, t_\ell)$.

Figure 1: Sumcheck protocol $(\mathsf{P}_{\mathsf{SC}}(g), \mathsf{V}^{g}_{\mathsf{SC}}(v))$ [LFKN92, Sha92]

For our applications, it will be convenient to think of the output of the sumcheck protocol to both parties to be the vector $(t_1, \cdots, t_\ell)$ and v_ℓ such that the final check of the verifier can be performed using the outputs of the sumcheck protocol by a *single query* to g.

Theorem 3 ([LFKN92, Sha92]). *Let $g : \mathbb{F}^\ell \mapsto \mathbb{F}$ be an ℓ-variate polynomial of degree d in each variable. The sumcheck protocol $(\mathsf{P}_{\mathsf{SC}}, \mathsf{V}^{g}_{\mathsf{SC}})$ described in Figure 1 satisfies the following properties.*

Completeness *If $\sum_{b_1,\cdots,b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell) = v$, then*

$$\Pr_{t_1,\cdots,t_\ell}\left[(\mathsf{P}_{\mathsf{SC}}(g), \mathsf{V}^{g}_{\mathsf{SC}}(v)) = 1\right] = 1$$

Soundness *If $\sum_{b_1,\cdots,b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell) \neq v$, then for every (possibly unbounded) prover P^**

$$\Pr_{t_1,\cdots,t_\ell}\left[(\mathsf{P}_{\mathsf{SC}}(g), \mathsf{V}^{g}_{\mathsf{SC}}(v)) = 1\right] \leq \frac{d \cdot \ell}{|\mathbb{F}|}$$

Efficiciency *The prover runs in time $\leq \mathsf{poly}(2^\ell, T_g)$ where T_g is the time taken to evaluate g. The verifier runs in time $\leq \ell \cdot \mathsf{poly}(d, \log|\mathbb{F}|)$ and the total communication complexity is $\leq \mathsf{poly}(d, \ell, \log|\mathbb{F}|)$. The number of bits sent by the prover to verifier is $O(d \cdot \ell \cdot \log|\mathbb{F}|)$.*

The sumcheck protocol is also public-coin.

Remark 1. *We assume that all field operations in $\mathbb{F}$ take time* $\mathsf{polylog}|\mathbb{F}|$.

3.3 Cryptographic Definitions

We start by defining the standard notion of computational indistinguishability.

Definition 3 (Computational Indistinguishability). *Two ensembles $X = \{X_\alpha\}_{\alpha \in S}$ and $Y = \{Y_\alpha\}_{\alpha \in S}$ are said to be computationally indistinguishable, denoted by $X \approx_c Y$, if for every non-uniform* PPT *distinguisher $\mathcal{D}$, every polynomial* p*, all sufficiently large λ and every $\alpha \in \{0,1\}^{\mathsf{poly}(\lambda)} \cap S$*

$$\left|\Pr\left[\mathcal{D}(1^\lambda, X_\alpha) = 1\right] - \Pr\left[\mathcal{D}(1^\lambda, Y_\alpha) = 1\right]\right| < \frac{1}{\mathsf{p}(\lambda)} \ ,$$

where the probability are taken over the samples of X_α, Y_α and coin tosses of $\mathcal{D}$.

3.3.1 Learning with Error Assumption

We state below the learning with errors (LWE) assumption.

Definition 4 (Learning with Error Assumption). *For any positive integers n, q, any $\mathbf{s} \in \mathbb{Z}^n$, and any error distribution χ over $\mathbb{Z}$, the LWE (Learning with Error) distribution $A_{\mathbf{s},\chi}$ is defined by uniformly sampling a vector $\mathbf{a}$, and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $e \leftarrow \chi$.*

The $\mathsf{LWE}_{n,q,\chi}$ assumption states that, any n.u. PPT adversary can not distinguish, with non-negligible probability, between the distribution $A_{\mathbf{s},\chi}$ and the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

3.3.2 Decisional Diffie-Hellman Assumption

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

Definition 5 (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 $\nu(\lambda)$ such that

$$\Big|\Pr\left[(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda), a, b \leftarrow \mathbb{Z}_p : \mathcal{D}(1^\lambda, \mathbb{G}, p, g, g^a, g^b, g^{ab}) = 1\right] -$$
$$\Pr\left[(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda), a, b, c \leftarrow \mathbb{Z}_p : \mathcal{D}(1^\lambda, \mathbb{G}, p, g, g^a, g^b, g^c) = 1\right]\Big| \le \nu(\lambda)$$

Sub-exponential DDH. We say that $\mathcal{G}$ satisfies the sub-exponential DDH assumption, if there exists a constant $0 < c < 1$ such that for any n.u. PPT distinguisher, the advantage $\nu(\lambda)$ is bounded by $2^{-\lambda^c}$ for any sufficiently large λ.

3.3.3 Quadratic Residuosity.

For any integer N, let $\mathbb{Z}_N^*$ be the multiplication group of all elements that is co-prime with N. Let $\mathbb{J}_N$ be the subgroup of $\mathbb{Z}_N^*$ that consists of all elements with Jacobi symbol 1. Given N, and any element $x \in \mathbb{Z}_N^*$, there exists an efficient algorithm to test whether $x \in \mathbb{J}_N$. Let $\mathbb{QR}_N$ be the subgroup of $\mathbb{Z}_N^*$ that consists of all the quadratic residues. We say that N is a Blum integer, if $N = p \cdot q$, where p, q are distinct primes such that $p \pmod 4 = q \pmod 4 = 3$. Let $\mathbb{H} = \{-1, +1\}$ be a multiplicative group, then we have $\mathbb{QR}_N = \mathbb{H} \times \mathbb{QR}_N$.

In this work, we assume without loss of generality that the bit length of N is $O(\lambda)$.

Definition 6 (Quadratic Residuosity Assumption). *Let N be an uniformly sampled Blum integer, then for any non-uniform PPT adversary $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that*

$$\left|\Pr_N\left[x \leftarrow \mathbb{J}_N : \mathcal{D}(1^\lambda, N, x) = 1\right] - \Pr_N\left[x \leftarrow \mathbb{QR}_N : \mathcal{D}(1^\lambda, N, x) = 1\right]\right| < \nu(\lambda).$$

We will use the following theorem from [BG10].

Theorem 4 ([BG10]). *Let N be a Blum integer, and $\mathbb{QR}_N$ be the quadratic residuosity group with generator g, and let $k = k(\lambda)$ be a polynomial, then for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\lambda)$ such that*

$$\left|\Pr_N\left[\mathbf{a} \leftarrow \mathcal{A}(1^\lambda, N, g) : \mathcal{A}((-1)^{\mathbf{a}} \cdot (\mathbf{g})^s) = 1\right] - \Pr_N\left[\mathbf{a} \leftarrow \mathcal{A}(1^\lambda, N, g) : \mathcal{A}((\mathbf{g})^s) = 1\right]\right| < \nu(\lambda),$$

where $s \leftarrow \lfloor (N-1)/2 \rceil$, and $\mathbf{g} \leftarrow \mathbb{QR}_N^k$.

3.3.4 Correlation Intractable Hash

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

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

Hash: a *deterministic* polynomial algorithm 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 7 ([CGH04]). *A hash family $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ is said to be correlation intractable* (CI) *for a function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ if the following two properties hold:*

- *For every $\lambda \in \mathbb{N}$, every $f \in \mathcal{F}_\lambda$, every $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$, the functions f and $\mathcal{H}.\mathsf{Hash}(k, \cdot)$ have the same domain and co-domain.*
- *For every* PPT *adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $f \in \mathcal{F}_\lambda$,*

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

For our work it will suffice to consider constructions of CI hash families for (a) the class of all size circuits of size $\rho(\lambda)$ for some polynomial function ρ assuming the Learning with Errors (LWE) assumption; and (b) the class TC^0 assuming sub-exponential hardness of the Decisional Diffie-Hellman (DDH) assumption. We state the formal theorems below.

Theorem 5 (CIH from LWE [PS19]). *Let $L = L(\lambda), S = S(\lambda)$ and $d = d(\lambda)$ be polynomials in λ. Let the function family $\mathcal{F} = \{\mathcal{F}_{\lambda,L,S,d}\}_\lambda$, where $\mathcal{F}_{\lambda,L,S,d}$ contains all functions that can be computed by a circuit of output length $L(\lambda)$, size $S(\lambda)$, and depth $d(\lambda)$. Assuming the hardness of learning with error (Definition 4), there exists a construction of correlation intractable hash function for $\mathcal{F}$.*

Theorem 6 (CIH for TC^0 from sub-exponential DDH [JJ21]). *For any fixed depth L, let $\mathsf{TC}^0_L = \{\mathsf{TC}^0_{L,\lambda}\}_\lambda$ be the circuit family consists of all threshold circuits of depth L, and output length $\Omega(\lambda)$. Assuming the sub-exponential security of DDH (Definition 5), there exists a construction of correlation intractable hash function for TC^0_L.*

4 Somewhere-Extractable Linearly Homomorphic Commitments

In this section, we introduce the notion of somewhere-extractable linearly homomorphic commitment. In such a commitment scheme, one commits to a vector of values using a commitment key that can be generated using one of two indistinguishable modes: *normal mode* or *extraction mode*. Before going into the details, we give an overview of the desired properties from such a scheme:

Somewhere Extraction: When generating the commitment key K in the *extraction mode*, a coordinate i^* is chosen such that any commitment under the key K binds the least significant bit of the i^*-th coordinate of the committed message. Further, alongside the commitment key, the key generation algorithm in the *extraction mode* also outputs a trapdoor td, which allows one to extract the least significant bit of the i^*-th coordinate of the message (i.e. extraction $\pmod 2$). We denote this extraction algorithm as $\mathsf{Ext}(\cdot, \mathsf{td})$.

Linear Homomorphism: Consider two commitments $c_1 = \mathsf{Com}(K, \mathbf{m}_1; r_1)$ and $c_2 = \mathsf{Com}(K, \mathbf{m}_2; r_2)$ under the same commitment key (in any mode) for messages $\mathbf{m}_1, \mathbf{m}_2$ with corresponding randomness r_1, r_2. For any integers a and b, given c_1 and c_2, there is a way to homomorphically obtain the commitment $\mathsf{Com}(K, a \cdot \mathbf{m}_1 + b \cdot \mathbf{m}_2; a \cdot r_1 + b \cdot r_2)$.

Linearly Homomorphic Extraction: The aforementioned linear homomorphism only concerns well-formed commitments. However, we also need the linear homomorphism properties to hold for commitments that may not be well-formed. To this end, we introduce an extractable space $\mathcal{E}$, such that for any $c \in \mathcal{E}$, as the name suggests, we can use the extraction algorithm Ext to extract a message. Additionally, we require $\mathcal{E}$ to satisfy the following properties:

Public Verifiability: Given any c, it can be publicly verified if $c \in \mathcal{E}$.

Close Under Linear Homomorphism: The linear homomorphic operation is closed in $\mathcal{E}$, i.e. for any two elements in $\mathcal{E}$, the linear homomorphic evaluated commitment is also in $\mathcal{E}$.

Extraction is Linear Homomorphic: Most importantly, the extraction operation is linear homomorphic in $\mathcal{E}$, i.e. for any two elements $c_1, c_2 \in \mathcal{E}$, and any two integers a_1, a_2, we have

$$\mathsf{Ext}(a_1 \cdot c_1 + a_2 \cdot c_2, \mathsf{td}) = a_1 \cdot \mathsf{Ext}(c_1, \mathsf{td}) + a_2 \cdot \mathsf{Ext}(c_2, \mathsf{td}) \pmod 2$$

Low-Depth Extraction: For our applications, we ideally want the extraction algorithm Ext be computed in low depth, specifically TC^0. However, this is hard to achieve. Hence, we decompose the extraction algorithm to an offline pre-computation phase $\mathsf{PreComp}$, where $\mathsf{PreComp}$ is not allowed to use td but can be of polynomial depth, and a *low-depth* online-phase $\mathsf{OnlineExt}(\cdot, \mathsf{td})$. We only require that the online-phase of extraction $\mathsf{OnlineExt}(\cdot, \mathsf{td})$ be computed in TC^0.

In Section 4.1 we formally define such a commitment scheme. In Section 4.2, we show an extension of the definition to a more general setting, where the commitments are over polynomials. Lastly, in Section 4.3, we construct such a commitment scheme from the quadratic residuosity assumption.

4.1 Definition

A somewhere-extractable linearly homomorphic commitment scheme is a tuple of algorithms $\mathsf{LHC} = (\mathsf{Gen}, \mathsf{ExtGen}, \mathsf{Com}, \mathsf{Ext}, \mathsf{Samp})$ described below, where Samp is a randomness sampling algorithm for the commitment.[10]

[10]We use an explicit randomness sampling algorithm because in our construction from QR, the randomness is sampled from a space that depends on the commitment key.

- $\mathsf{Gen}(1^\lambda, 1^k)$: On input the security parameter λ and input length k, outputs a commitment key K.
- $\mathsf{ExtGen}(1^\lambda, 1^k, i^*)$: On input the security parameter λ, input length k and an index $i^* \in [k]$, outputs an *extractable* commitment key K, and a trapdoor td.
- $\mathsf{Com}(K, (x_1, x_2 \ldots, x_k); r)$: On input a commitment key K, k integers $(x_1, x_2, \ldots, x_k) \in \mathbb{Z}^k$ and randomness $r \leftarrow \mathsf{Samp}(K)$ as input, outputs a commitment c.
- $\mathsf{Ext}(c, \mathsf{td})$: On input a commitment c and a trapdoor td, output a message m. Further, this can be decomposed into two algorithms $\mathsf{PreComp}$ and $\mathsf{OnlineExt}$ described as follows:
  - $\mathsf{PreComp}(1^\lambda, c)$: On input the security parameter λ and a commitment c, output a *pre-processed* value c' that is to be used for *online extraction.*
  - $\mathsf{OnlineExt}(c', \mathsf{td})$: On input the pre-processed commitment c' and a trapdoor td, output a message $m \in \mathbb{F}_2$.

  For correctness, we require that $\mathsf{Ext}(c, \mathsf{td}) = \mathsf{OnlineExt}(\mathsf{PreComp}(1^\lambda, c), \mathsf{td})$. We also emphasize that $\mathsf{PreComp}$ does not take the trapdoor as input.

We require the algorithms to satisfy the following properties.

Compactness: The size of the commitment is bounded by some fixed polynomial $\mathsf{poly}(\lambda)$ in the security parameter.

Key Indistinguishability: For any integer $i^* \in [k]$, and any non-uniform PPT adversary $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that

$$|\Pr\left[K \leftarrow \mathsf{Gen}(1^\lambda, 1^k) : \mathcal{D}(1^\lambda, K) = 1\right] - \Pr\left[K \leftarrow \mathsf{ExtGen}(1^\lambda, 1^k, i^*) : \mathcal{D}(1^\lambda, K) = 1\right]| < \nu(\lambda).$$

Linear Homomorphism: There exists a binary operation "+" over the commitments such that, for any key K, $a, b \in \mathbb{Z}$, and any integers vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^k$, and randomness $r, u \in \mathbb{Z}$, we have

$$a \cdot \mathsf{Com}(K, \mathbf{x}; r) + b \cdot \mathsf{Com}(K, \mathbf{y}; u) = \mathsf{Com}(K, a \cdot \mathbf{x} + b \cdot \mathbf{y}; a \cdot r + b \cdot u).$$

Extraction: The extraction algorithm Ext satisfies the following properties:

Somewhere $\mathbb{F}_2$-Extraction: For any $\mathbf{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{Z}^k$, any $i^* \in [k]$, and any randomness $r \in \mathbb{Z}$,

$$\Pr[(K, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, 1^k, i^*), c = \mathsf{Com}(K, \mathbf{x}; r) : \\ \mathsf{Ext}(c', \mathsf{td}) = x_{i^*} \bmod 2] = 1.$$

Linearly Homomorphic Extraction: There exists an extractable space $\mathcal{E}$ and a polynomial time algorithm $\mathcal{E}\mathsf{Ver}$ such that, for any $c \in \{0, 1\}^*$,

$$\Pr\left[c \in \mathcal{E} \iff \mathcal{E}\mathsf{Ver}(1^\lambda, c) = 1\right] = 1.$$

Furthermore, $\mathcal{E}$ is closed under linear combination, i.e. for any $a_1, a_2 \in \mathbb{Z}, c_1, c_2 \in \mathcal{E}$, we have $a_1 \cdot c_1 + a_2 \cdot c_2 \in \mathcal{E}$, and

$$\Pr\left[\mathsf{Ext}(a_1 \cdot c_1 + a_2 \cdot c_2, \mathsf{td}) = a_1 \cdot \mathsf{Ext}(c_1, \mathsf{td}) + a_2 \cdot \mathsf{Ext}(c_2, \mathsf{td}) \pmod 2\right] = 1.$$

In addition, every "well-formed" commitment is in $\mathcal{E}$ .i.e. for any key K, input $\mathbf{x} \in \mathbb{Z}^k$, and randomness $r \in \mathbb{Z}$, we have $\mathsf{Com}(K, \mathbf{x}; r) \in \mathcal{E}$.

Low-Depth Online Extraction: The algorithm $\mathsf{OnlineExt}$ can be computed by TC^0 circuits.

4.2 Extension to Homomorphism w.r.t. to Polynomial.

In this section, we extend the commitment scheme in Section 4.1 to allow commitment for polynomials. Specifically, we define the commitment for a *vector of polynomials* $\mathbf{f}(\alpha) = \mathbf{f}_0 + \mathbf{f}_1 \cdot \alpha + \ldots + \mathbf{f}_{t-1} \cdot \alpha^{t-1} \in \mathbb{Z}^k[\alpha]$ as the commitments for each of its coefficient vectors $(\mathsf{Com}(K, \mathbf{f}_0), \mathsf{Com}(K, \mathbf{f}_1), \ldots, \mathsf{Com}(K, \mathbf{f}_{t-1}))$ (for simplicity, we omit the randomness). The linear homomorphic property, somewhere extraction property and linear homomorphic extraction can also be extended similarly to polynomials.

Let $v(\alpha) \in \mathbb{Z}[\alpha]$ be a monic polynomial over $\mathbb{Z}$ with degree t. Let $R = \mathbb{Z}[\alpha]/(v(\alpha))$ be the polynomial ring over the integers modulo $v(\alpha)$. Let $R/2R$ be the quotient ring of R divided by the ideal $2R$. We define the following operations.

- **Commitment w.r.t. Polynomial:** For any polynomials $f^{(i)} = \sum_{j=0}^{t-1} f_j^{(i)} \cdot \alpha^j$ where $i \in [k]$, and $r = \sum_{j=0}^{t-1} r_j \cdot \alpha^j \in R$, where $f_j, r_j \in \mathbb{Z}$, we define the notation $\mathsf{Com}(K, f^{(1)}, f^{(2)}, \ldots, f^{(k)}; r)$ as

$$\mathsf{Com}(K, f^{(1)}, f^{(2)}, \ldots, f^{(k)}; r) = (\mathsf{Com}(K, f_0^{(1)}, \ldots, f_0^{(k)}; r_0), \mathsf{Com}(K, f_1^{(1)}, \ldots, f_1^{(k)}; r_1), \ldots, \mathsf{Com}(K, f_{t-1}^{(1)}, \ldots, f_{t-1}^{(k)}; r_{t-1})).$$

- **Extraction w.r.t. Polynomial:** For any $c = (c_0, c_1, \ldots, c_{t-1}) \in \mathcal{E}^t$, we define $\mathsf{Ext}(c, \mathsf{td})$ as

$$\mathsf{Ext}(c, \mathsf{td}) = \mathsf{Ext}(c_0, \mathsf{td}) + \mathsf{Ext}(c_1, \mathsf{td}) \cdot \alpha + \ldots + \mathsf{Ext}(c_{t-1}, \mathsf{td}) \cdot \alpha^{t-1} \in R/2R.$$

- **Somewhere $\mathbb{F}_2$-Extraction w.r.t. Polynomial:** For any polynomial $f^{(1)}, f^{(2)}, \ldots, f^{(k)} \in R$, and $r \in R$, we have

$$\Pr[(K, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, 1^k, i^*) : \mathsf{Ext}\left(\mathsf{Com}(K, f^{(1)}, f^{(2)}, \ldots, f^{(k)}; r), \mathsf{td}\right) = f^{(i^*)} \pmod 2 \in R/2R] = 1$$

- **Homomorphism w.r.t. to Polynomial:** For any commitment $c \in \mathcal{E}$, we define $f \cdot c$ as

$$f \cdot c = (f_0 \cdot c, f_1 \cdot c, \ldots, f_{t-1} \cdot c).$$

  Furthermore, for any two tuples of commitments $\mathbf{c} = (c_0, c_1, \ldots, c_{t-1}), \mathbf{c}' = (c_0', c_1', \ldots, c_{t-1}') \in \mathcal{E}^t$, we define the addition as

$$\mathbf{c} + \mathbf{c}' = (c_0 + c_0', c_1 + c_1', \ldots, c_{t-1} + c_{t-1}').$$

- **Linear Homomorphic Extraction w.r.t. Polynomial:** For any integer n, and n commitments $c_i \in \mathcal{E}, i = 1, 2, \ldots, n$ in the extractable space $\mathcal{E}$, and n polynomials $f_1, f_2, \ldots, f_n \in R$, we have

$$\Pr[\mathsf{Ext}(f_1 \cdot c_1 + f_2 \cdot c_2 + \ldots + f_n \cdot c_n, \mathsf{td}) = f_1 \cdot \mathsf{Ext}(c_1, \mathsf{td}) + f_2 \cdot \mathsf{Ext}(c_2, \mathsf{td}) + \ldots + f_n \cdot \mathsf{Ext}(c_n, \mathsf{td}) \pmod 2] = 1$$

It is easy to see, given the definitions above, that the above is a generic transformation from *somewhere-extractable linearly homomorphic commitment* to support extension to homomorphism with respect to polynomials.

Since the homomorphism, as described, results in the growth of the values inside the commitment, we need to ensure that the openings do not become "too large". The following lemma bounds the size of the opening.

Lemma 1 (Bound on the size of the opening)**.** *For any integer n and B, let $f_1, f_2, \ldots, f_n$ be n polynomials in the ring $R = \mathbb{Z}[\alpha]/(v(\alpha))$ with coefficients in the range $[-B, B]$, and let $c_i = \mathsf{Com}(K, \mathbf{x}_i; r_i)$, $i = 1, 2, \ldots, n$ be n commitments, then the coefficients of the openings $(\sum_{i=1}^n f_i \mathbf{x}_i, \sum_{i=1}^n f_i r_i)$ is in $[-B', B']$, where*

$$B' = nB \cdot \max_{i \in [n]} \{\|\mathbf{x}_i\|_\infty, |r_i|\}.$$

Proof. For coefficients of $\sum_{i=1}^n f_i \mathbf{x}_i$, they can be bounded by $\sum_{i=1}^n B \|\mathbf{x}_i\|_\infty \le nB \max_{i \in [n]} \|\mathbf{x}_i\|_\infty$. The term $\sum_{i \in [n]} f_i r_i$ can be bounded similarly. □

4.3 Construction

We present our construction of somewhere-extractable linearly homomorphic commitments in Figure 2.

The reader may note that we have not split the extraction algorithm Ext in Figure 2 as necessitated by the definition. Instead we defer the decomposition into PreComp and OnlineExt to Figure 3. We state below the theorem, and then prove each of the properties subsequently.

Theorem 7. *The construction in Figure 2 is a somewhere-extractable linearly homomorphic commitment based on the Quadratic Residuosity assumption (Definition 6), where the decomposed extraction algorithm is described in Figure 3.*

4.4 Proof of Theorem 7

We now prove each the required properties in a sequence of lemmas below.

Lemma 2 (Compactness)**.** *The construction in Figure 2 satisfies compactness property.*

The compactness follows directly from the construction.

Lemma 3 (Key Indistinguishability)**.** *The construction in Figure 2 satisfies key indistinguishability.*

Proof. For any integer $k = k(\lambda)$, any index $i^* \in [k]$, and any adversary $\mathcal{D}$, we build the following adversary $\mathcal{A}$ for Theorem 4. Upon input of (N, g'), $\mathcal{A}$ output the vector $\mathbf{a} = \mathbb{1}_{i^*}$, where $\mathbb{1}_{i^*} \in \{0,1\}^k$ is the indicator vector with 1 on the i^*-th coordinate, and 0 elsewhere. Next, the challenger samples a random s and sends to $\mathcal{A}$ the tuple $(\mathbf{h}, h)$, which is either $(-1)^{\mathbf{a}} \cdot (\mathbf{g}, g)^s$ or $(\mathbf{g}, g)^s$. When $\mathcal{A}$ receives $(\mathbf{h}, h)$, it invokes the adversary $\mathcal{D}$ on the input $K = (N, g, h, \mathbf{g}, \mathbf{h})$, and passes the output of $\mathcal{D}$ as its own. By Theorem 4, the adversary $\mathcal{A}$ cannot distinguish whether $(\mathbf{h}, h) = (-1)^{\mathbf{a}} \cdot (\mathbf{g}, g)^s$ or $(\mathbf{h}, h) = (\mathbf{g}, g)$. However, the adversary $\mathcal{A}$ simulates the correct input distribution for $\mathcal{D}$. This completes the proof. □

Lemma 4 (Linear Homomorphism)**.** *The construction in Figure 2 satisfies the linear homomorphism property.*

Proof. We first define the linear homomorphic operation over the commitment. For any integer $a, b \in \mathbb{Z}$, and any commitment $c = (c_g, c_h), d = (d_g, d_h) \in \mathbb{J}_N^2$, we define the linear homomorphic operation as

$$a \cdot c + b \cdot d = (c_g^a \cdot d_g^b \bmod N, c_h^a \cdot d_h^b \bmod N).$$

If $c = \mathsf{Com}(K, \mathbf{x}; r)$ and $d = \mathsf{Com}(K, \mathbf{y}; u)$, where $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_k)$, then we have

$$\begin{aligned} a \cdot c + b \cdot d &= \left((g^r \prod_{i=1}^k g_i^{x_i})^a \cdot (g^u \prod_{i=1}^k g_i^{y_i})^b, (h^r \prod_{i=1}^k h_i^{x_i})^a \cdot (h^u \prod_{i=1}^k h_i^{y_i})^b \right) \\ &= \left(g^{ar+bu} \prod_{i=1}^k g_i^{ax_i + by_i}, h^{ar+bu} \prod_{i=1}^k h_i^{ax_i + by_i} \right) = \mathsf{Com}(K, a\mathbf{x} + b\mathbf{y}; ar + bu). \end{aligned}$$

□

Somewhere-Extractable Linearly Homomorphic Commitment

- **Key Generation** $\mathsf{Gen}(1^\lambda, 1^k)$**:**
  - Uniformly sample a Blum integer N.
  - Uniformly sample $g \leftarrow \mathbb{QR}_N$, $\mathbf{g} \leftarrow \mathbb{QR}_N^k$, and $s \leftarrow \lfloor (N-1)/2 \rceil$.
  - Let $h = g^s, \mathbf{h} = (\mathbf{g})^s$
  - Output $K = (N, g, h, \mathbf{g}, \mathbf{h})$.
- **Extractable Key Generation** $\mathsf{ExtGen}(1^\lambda, 1^k, i^*)$**:**
  - Uniformly sample a Blum integer $N = p \cdot q$.
  - Uniformly sample $g \leftarrow \mathbb{QR}_N$, $\mathbf{g} \leftarrow \mathbb{QR}_N^k$, and $s \leftarrow \lfloor (N-1)/2 \rceil$.
  - Let $h = g^s$ and $\mathbf{h} = (-1)^{\mathbb{1}_{i^*}} \cdot (\mathbf{g})^s$, where $\mathbb{1}_{i^*} \in \{0,1\}^k$ is the indicator vector with 1 on the i^*-th coordinate, and 0 elsewhere.
  - Output $K = (N, g, h, \mathbf{g}, \mathbf{h})$, and $\mathsf{td} = (p, q, s)$.
- **Commit** $\mathsf{Com}(K = (N, g, h, \mathbf{g}, \mathbf{h}), \mathbf{x} \in \mathbb{Z}^k; r)$**:**
  - Parse $\mathbf{g} = (g_1, g_2, \ldots, g_k), \mathbf{h} = (h_1, h_2, \ldots, h_k)$.
  - Let $c_g = g^r \cdot \prod_{i=1}^k g_i^{x_i}, c_h = h^r \cdot \prod_{i=1}^k h_i^{x_i}$.
  - Output $c = (c_g, c_h)$.
- **Extraction** $\mathsf{Ext}(c = (c_g, c_h), \mathsf{td} = (p, q, s))$**:**
  - If $(c_h/c_g^s)^{(p-1)(q-1)/4} = 1 \pmod N$, then output 0, otherwise output 1.
- **Randomness Sampling** $\mathsf{Samp}(K)$**:**
  - Output an uniformly sampled $r \leftarrow \lfloor (N-1)/2 \rceil$.

Figure 2: Construction of somewhere-extractable linearly homomorphic commitment.

Lemma 5 (Somewhere $\mathbb{F}_2$-Extraction)**.** *The construction in Figure 2 satisfies the somewhere $\mathbb{F}_2$-extraction property.*

Proof. For any $\mathbf{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{Z}^k$, and any $i^* \in [k]$, and any randomness $r \in \mathbb{Z}$, let $(K, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, 1^k, i^*)$, and $c = \mathsf{Com}(K, \mathbf{x}; r)$. Then $c = (g^r \prod_{i=1}^k g_i^{x_i}, h^r \prod_{i=1}^k h_i^{x_i})$, where $\mathbf{g} = (g_1, g_2, \ldots, g_n)$ and $\mathbf{h} = (h_1, h_2, \ldots, h_n)$ are specified by the commitment key K. Hence, $\mathbf{h} = (-1)^{\mathbb{1}_{i^*}}(\mathbf{g})^s$, where $\mathbb{1}_{i^*} \in \{0,1\}^k$ is the indicator vector with the i^*-th position set to be 1, and $h = g^s$. Now we have

$$c = \left(g^r \cdot \prod_{i=1}^k g_i^{x_i}, g^{sr} \cdot \left(\prod_{i=1}^k g_i^{sx_i} \right) \cdot (-1)^{x_{i^*}} \right)$$

If we denote c as (c_g, c_h), then $c_h/c_g^s = (-1)^{x_{i^*}}$. In the Ext algorithm, since N is a Blum integer, $(p-1)(q-1)/4$ is an odd. Hence, we have $(c_h/c_g^s)^{(p-1)(q-1)/4} = 1$ if and only if $x_{i^*} \pmod 2 = 0$. □

Lemma 6 (Linear Homomorphic Extraction)**.** *The construction in Figure 2 satisfies linear homomorphic extraction property.*

Proof. Let the extraction space $\mathcal{E}$ be $\mathbb{J}_N^2$. Since Jacobi symbol can be computed efficiently, there exists an efficient algorithm $\mathcal{E}\mathsf{Ver}$ such that $\Pr[c \in \mathcal{E} \iff \mathcal{E}\mathsf{Ver}(1^\lambda, c) = 1] = 1$.

To show that the linear homomorphic operation is closed over $\mathcal{E} = \mathbb{J}_N^2$, we consider any integers $a, b \in \mathbb{Z}$, and any commitments $c = (c_g, c_h), d = (d_g, d_h) \in \mathbb{J}_N^2$. Since we define $a \cdot c + b \cdot d = (c_g^a \cdot d_g^b, c_h^a \cdot d_h^b)$, and the multiplication is closed in the group $\mathbb{J}_N$, we have $a \cdot c + b \cdot d \in \mathbb{J}_N^2$.

For linear homomorphic extraction property, we first claim a general property about the extraction algorithm Ext.

Claim 1. *For any $c = (c_g, c_h) \in \mathbb{J}_N^2$, and any $\mathsf{td} = (p, q, s)$, where $N = p \cdot q$ is a Blum integer, since $\mathbb{J}_N = \mathbb{H} \times \mathbb{QR}_N$, there exists a unique $m \in \mathbb{Z}_2$ and $g \in \mathbb{QR}_N$ such that $c_h/c_g^s = (-1)^m \cdot g$, then $\mathsf{Ext}(c, \mathsf{td}) = m \pmod 2$.*

We defer the proof of the claim to the end of the proof. Now for any $c = (c_g, c_h), d = (d_g, d_h) \in \mathcal{E}$, since $\mathbb{J}_N = \mathbb{H} \times \mathbb{QR}_N$, there exists a unique $m_c \in \mathbb{Z}_2$ and $g_c \in \mathbb{QR}_N$ such that $c_h/c_g^s = (-1)^{m_c} g_c$, and also an unique $m_d \in \mathbb{Z}_2$ and $g_d \in \mathbb{QR}_N$ such that $d_h/d_g^s = (-1)^{m_d} g_d$. Then, for $a \cdot c + b \cdot d = (c_g^a \cdot d_g^b, c_h^a \cdot d_h^b)$, we have

$$c_h^a \cdot d_h^b / (c_g^a \cdot d_g^b)^s = (c_h/c_g^s)^a \cdot (d_h/d_g^s)^b = (-1)^{a \cdot m_c + b \cdot m_d} \cdot (g_c^a \cdot g_d^b).$$

Since $(-1)^{a \cdot m_c + b \cdot m_d} \in \mathbb{H}$ and $g_c^a \cdot g_d^b \in \mathbb{QR}_N$, by the Claim 1, we have

$$\mathsf{Ext}(a \cdot c + b \cdot d, \mathsf{td}) = a \cdot m_c + b \cdot m_d \pmod 2 = a \cdot \mathsf{Ext}(c, \mathsf{td}) + b \cdot \mathsf{Ext}(d, \mathsf{td}) \pmod 2.$$

□

Proof of the Claim 1. Let $n = (p-1)(q-1)/4$ be the order of the group $\mathbb{QR}_N$. Hence, if $c_h/c_g^s = (-1)^m \cdot g$, where $g \in \mathbb{QR}_N$, then $g^n = 1$, and thus $(c_h/c_g^s)^n = (-1)^{m \cdot n} g^n = (-1)^m$. This completes the proof. □

Lemma 7 (Low-Depth Online Extraction)**.** *The construction in Figure 3 satisfies the low-depth online extraction property.*

Proof. The work [RT92, HAM02] proved that the iterative multiplication, and division operations are in TC^0. Hence, the computation of e_g and e_h can be done in TC^0. The exponentiation $c_{g,i}^{e_{g,i}}, {c'_{h,i}}^{e_{h,i}}$ can also be computed in TC^0, since $e_{g,i}, e_{h,i} \in \{0, 1\}$ are binary values. Finally, m is the iterative multiplication of $c_{g,i}^{e_{g,i}}, {c'_{h,i}}^{e_{h,i}}$, and hence can also be computed in TC^0. □

Lemma 8. *The* Ext *algorithm in Figure 2 and the* $\mathsf{PreComp}$, $\mathsf{OnlineExt}$ *algorithms in Figure 3 satisfy that for any $c \in \mathcal{E}$, any $(K, \mathsf{td}) \leftarrow \mathsf{ExtGen}(1^\lambda, 1^k, i^*)$, we have* $\mathsf{Ext}(c, \mathsf{td}) = \mathsf{OnlineExt}(\mathsf{PreComp}(1^\lambda, c), \mathsf{td})$

Proof. Following the algorithm OnlineExt, m is computed as $m = \prod_{i=0}^{\lambda} {c'_{g,i}}^{e_{g,i}} \cdot \prod_{i=0}^{\lambda} {c'_{h,i}}^{e_{h,i}} \pmod N$ Since $e_{g,i}$ and $e_{h,i}$ are the binary representation of e_g and e_h, the right hand side equals to $c_g^{e_g} \cdot c_h^{e_h} = {c_g}^{-s \cdot n} c_h^n = (c_h/c_g^s)^n$. Hence, $\mathsf{Ext}(c, \mathsf{td}) = \mathsf{OnlineExt}(\mathsf{PreComp}(1^\lambda, c), \mathsf{td})$. □

5 Dual Mode Interactive Batch Arguments for NP

In this section, we define and construct dual mode interactive batch arguments for NP. At a high-level, such an argument system allows for proving multiple instances of an NP language while incurring roughly the communication (and verification) cost of proving a single instance. We consider such protocols in the CRS model that may be executed in one of two modes – *normal* mode or *trapdoor* mode. The former

Pre-Computation and Online Extraction

- **Pre-Computation** $\mathsf{PreComp}(1^\lambda, c = (c_g, c_h))$**:**
  - Let $\mathbf{c}'_g = (c_g^{2^0}, c_g^{2^1}, \ldots, c_g^{2^\lambda})$, $\mathbf{c}'_h = (c_h^{2^0}, c_h^{2^1}, \ldots, c_h^{2^\lambda})$.
  - Output $c' = (\mathbf{c}'_g, \mathbf{c}'_h)$.
- **Online Extraction** $\mathsf{OnlineExt}(c' = (\mathbf{c}'_g, \mathbf{c}'_h), \mathsf{td} = (p, q, s))$**:**
  - Parse $\mathbf{c}'_g = (c'_{g,0}, c'_{g,1}, \ldots, c'_{g,\lambda})$ and $\mathbf{c}'_h = (c'_{h,0}, c'_{h,1}, \ldots, c'_{h,\lambda})$.
  - Let $n = (p-1)(q-1)/4$ be the order of $\mathbb{QR}_N$.
  - Let $e_g = -s \cdot n \pmod{2n}$, and $e_h = n$.
  - Let the binary representation of e_g be $(e_{g,0}, e_{g,1}, \ldots, e_{g,\lambda}) \in \{0,1\}^{\lambda+1}$, i.e. $e_g = e_{g,0} \cdot 2^0 + e_{g,1} \cdot 2^1 + e_{g,2} \cdot 2^2 + \ldots e_{g,\lambda} \cdot 2^\lambda$. Similarly, let the binary representation of e_h be $(e_{h,0}, e_{h,1}, \ldots, e_{h,\lambda}) \in \{0,1\}^{\lambda+1}$.
  - Let $m = \prod_{i=0}^{\lambda} {c'_{g,i}}^{e_{g,i}} \prod_{i=0}^{\lambda} {c'_{h,i}}^{e_{h,i}} \pmod{N}$
  - If $m = 1$, output 0, otherwise, output 1.

Figure 3: The extraction of the somewhere-extractable linearly homomorphic commitment.

corresponds to normal protocol execution while the latter mode is used in the security proof. Crucially, in the trapdoor mode, we require the protocol to satisfy non-adaptive *statistical* soundness.

Dual Mode Interactive Batch Arguments. We start by providing a formal definition. We shall denote by $\mathsf{Out}_A\langle A(a), B(b)\rangle$ the random variable that corresponds to the output of party A on execution of the protocol between A with input a, and B with input b. Here the probability is taken over the random coins of both A and B.

Definition 8 (Dual-Mode Interactive Batch Arguments). *A* dual-mode interactive batch argument,*denoted by a tuple of* PPT *algorithms* $(\mathsf{P}, \mathsf{V}, \mathsf{Gen}, \mathsf{TGen})$*, is an interactive protocol in the common reference string (CRS) model for an* NP *language L defined by relation $\mathcal{R}_L$ if it satisfies the following properties:*

Completeness. *For all* $\mathbf{x} = (x_1, \ldots, x_k)$ *and* $\mathbf{w} = (w_1, \ldots, w_k)$ *such that for each $i \in [k]$, $\mathcal{R}_L(x_i, w_i) = 1$, it holds that:*

$$\Pr\Big[\mathsf{Out}_\mathsf{V}\langle \mathsf{P}(\mathsf{crs}, \mathbf{x}, \mathbf{w}), \mathsf{V}(\mathsf{crs}, \mathbf{x})\rangle = 1 \;\Big|\; \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^k)\Big] = 1.$$

Dual Mode Indistinguishability. *The two setup modes are computationally indistinguishable, i.e. $\forall k \in \mathbb{N}, \forall i^* \in [k]$,*

$$\Big\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^k)\Big\}_{\lambda \in \mathbb{N}} \approx_c \Big\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{TGen}(1^\lambda, 1^k, i^*)\Big\}_{\lambda \in \mathbb{N}}$$

Non-Adaptive Statistical Soundness in Trapdoor Mode. *For every (possible unbounded) cheating prover* P^* *and all* $\mathbf{x} = (x_1, \ldots, x_k)$ *where $\exists i$ s.t. $x_i \notin L$, it holds that $\forall i^* \in [k]$ s.t. $x_{i^*} \notin L$:*

$$\Pr\Big[\mathsf{Out}_\mathsf{V}\langle \mathsf{P}^*(\mathsf{crs}, \mathbf{x}, \mathbf{w}), \mathsf{V}(\mathsf{crs}, \mathbf{x})\rangle = 1 \;\Big|\; (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TGen}(1^\lambda, 1^k, i^*)\Big] \le \mathsf{negl}(\lambda).$$

Remark 2. *The above definition implies (non-adaptive) soundness against* PPT *cheating provers in the* normal mode. *This is easily observed via a sequence of hybrids: (i) switch the* crs *being generated in the* normal mode *to the* trapdoor mode *for a randomly chosen index i^* while relying on the computational indistinguishability. With probability at least $1/k$ the chosen index i^* will be such that $x_{i^*} \notin L$; (ii) rely on the non-adaptive statistical soundness in the trapdoor mode.*

Our Construction. We construct a dual mode interactive batch argument for R1CS, where the instances are generated from instances of Boolean circuit satisfiability that all *share the circuit C* but have different statements x. (We refer the reader to Section A.2 for the corresponding reduction from Boolean satisfiability to R1CS.) This results in k instances of R1CS,

$$\{\mathbb{x}^{(j)}\}_{j\in[k]} = (\mathbb{F}, A, B, C, \{\mathsf{io}^{(j)}\}_{j\in[k]}, m, n),$$

where all the io^j are of the same length, and the instances *share the same* $\mathbb{F}$*, A, B, C, m and n*. Furthermore, as a consequence of the reduction, the witnesses to these instances $\{w^{(j)}\}_{j\in[k]}$ are all binary values, i.e. $\forall j \in [k]$, $w^{(j)} \in \{0,1\}^{m-|\mathsf{io}|-1}$. We work with the field $\mathbb{F}$, which is an extension field of $\mathbb{F}_2$ of size 2^λ. Specifically, $\mathbb{F} := \mathbb{F}_2[\alpha]/(v(\alpha))$ for some irreducible polynomial $v(\alpha)$ in $\mathbb{F}_2$ of degree λ.[11]

Our protocol relies on a single cryptographic component, namely, a somewhere-extractable linearly homomorphic commitment scheme LHC = (Gen, ExtGen, Com, Ext, Samp) (Section 4.1) which can be built (see Section 4.3) from the quadratic residuosity assumption (Definition 6). The dual mode property of our protocol comes exclusively from the use of this commitment scheme.

The formal description of the protocol is presented in Figure 4.

Below, we provide an overview of our protocol, focusing on how we implement batching.

1. Given k R1CS instances $\{\mathbb{x}^{(j)}\}_{j\in[k]}$, the prover needs to commit to k witnesses $\{w^{(j)}\}_{j\in[k]}$, where the witnesses are all of the same size $|w|$. This is done by first representing the k witnesses as a matrix W, where each witness occupies a single row. The prover uses LHC to commit to $|w|$ k-tuples corresponding to each column of the matrix W. From the compactness property of LHC, the total size of the commitments sent by the prover is proportional to the size of a *single witness* $|w|$.

   An observant reader may note that the message space for a k-tuple commitment in LHC is $\mathbb{Z}^k$, and not $\mathbb{F}^k$ as we would like. Here we utilize the fact that the prover is committing to the witness whose values are only binary (as consequence of the reduction from C-SAT to R1CS, see Section 3.1)[13], and therefore the message space for the commitment of a k-tuple is $\{0,1\}^k \subset \mathbb{Z}^k$. This also explains why we commit to the witness w rather than its multilinear extension, since they are equivalent when the witness is a binary string.

   Let $\{c_y\}_{y\in[|w|]}$ denote the commitments.

2. The prover and verifier run sumcheck protocols for polynomials $\{\mathcal{G}^{(j)}_{\tau,\mathsf{io}}\}_{j\in[k]}$ - there are k distinct polynomials since the witness (which determines the polynomial) for each R1CS instance is different. Specifically, the sumcheck protocol is to prove that the following sum

$$\sum_{x\in\{0,1\}^{\log m}} \mathcal{G}^{(j)}_{\tau,\mathsf{io}} := \sum_{x\in\{0,1\}^{\log m}} \widetilde{F}_{\mathsf{io}}(x)\cdot \widetilde{\mathsf{eq}}(x,\tau)$$

   is 0, where $\widetilde{F}_{\mathsf{io}}$ as defined in Section A.3 is a polynomial that depends on $A, B, C,$ io and w. The prover and verifier run all k sumchecks in parallel, where the verifier uses the *same randomness* across all the sumcheck protocols.

[11]One can think of the representation to be a λ length vector in $\mathbb{F}_2$ corresponding to the coefficients of the polynomial $f \in \mathbb{F}_2[\alpha]/(v(\alpha))$.

[13]Our protocol does not handle arbitrary R1CS instances where the witness may have values in $\mathbb{F}$ outside of $\{0,1\}$.

Protocol: Interactive Batch Argument $(\mathsf{P}, \mathsf{V}, \mathsf{Gen}, \mathsf{TGen})$

Common Reference String: $K \leftarrow \mathsf{LHC.Setup}_1(1^\lambda, 1^k)$
Common input: input $\{\mathbb{x}^{(j)}\}_{j\in[k]} := (\mathbb{F}, A, B, C, \{\mathrm{io}^{(j)}\}_{j\in[k]}, m, n)$, security parameter 1^λ
P's auxiliary input: witnesses $\{w^{(j)}\}_{j\in[k]}$ such that $\forall j \in [k]$, $\mathcal{R}_{\mathrm{R1CS}}(\mathbb{x}^{(j)}, w^{(j)}) = 1$

1. Prover commits to the k-tuple of witnesses in a *column-wise* fashion. Specifically $\forall y \in \{0,1\}^{s_2}$, $c_y := \mathsf{LHC.Com}(K, (w_y^{(1)}, \cdots, w_y^{(k)}); r_y)$ where $r_y \leftarrow\$ \mathsf{LHC.Samp}(K)$. Send $\{c_y\}_{y\in\{0,1\}^{s_2}}$ to the verifier V.
2. Verifier V samples a random vector $\tau \leftarrow\$ \mathbb{F}^s$ and sends τ to the prover P.
3. Prover P and verifier V run k sumcheck protocols $(\mathsf{P}_{\mathsf{SC}}(\mathcal{G}_{\mathrm{io}^{(j)},\tau}), \mathsf{V}_{\mathsf{SC}}(0))$[12] in parallel where the verifier uses the same randomness in each sumcheck. The output of the sumchecks are $r^* \in \mathbb{F}^s$ and $\{v^{(j)}\}_{j\in[k]}$.
4. The verifier V asks P for values $\{v_{A,2}^{(j)}, v_{B,2}^{(j)}, v_{C,2}^{(j)}\}_{j\in[k]}$.
5. Prover P computes $\forall j \in [k], X \in \{A, B, C\}$,
   - $v_{X,2}^{(j)} := \sum_{y\in\{0,1\}^{s_2}} \widetilde{X}''(r^*, y) \cdot w_y^{(j)}$,
   - $r_X := \sum_{y\in\{0,1\}^{s_2}} \widetilde{X}''(r^*, y) \cdot r_y$,

   sends $\{v_{A,2}^{(j)}, v_{B,2}^{(j)}, v_{C,2}^{(j)}\}_{j\in[k]}$ along with the *commitment openings* r_A, r_B, r_C to the verifier V.
6. The verifier V does the following
   (a) computes $\forall X \in \{A, B, C\}$, $c_X := \sum_{y\in\{0,1\}^{s_2}} \widetilde{X}''(r^*, y) \cdot c_y$
   (b) checks commitment opening, $\forall X \in \{A, B, C\}$, $c_X \stackrel{?}{=} \mathsf{LHC.Com}(K, (v_{X,2}^{(1)}, \cdots, v_{X,2}^{(k)}); r_X)$
   (c) computes $\forall j \in [k], X \in \{A, B, C\}$, $v_{X,1}^{(j)} := \sum_{y\in\{0,1\}^{s_1}} \widetilde{X}'(r^*, y) \cdot (\mathrm{io}||1)_y^{(j)}$, and $v_X^{(j)} = v_{X,1}^{(j)} + v_{X,2}^{(j)}$.
   (d) checks $\forall j \in [k]$, $v^{(j)} = (v_A^{(j)} \cdot v_B^{(j)} - v_C^{(j)}) \cdot \widetilde{\mathrm{eq}}(r^*, \tau)$, and reject if any of the checks fail.

   Accept if none of the checks have failed.

Figure 4: Interactive Batch Argument for R1CS.

3. At the end of the sumcheck protocol, the verifier needs to evaluate each polynomial $\{\mathcal{G}_{\tau,\mathrm{io}}^{(j)}\}_{j\in[k]}$ at a point $r^* \in \mathbb{F}^{\log m}$ determined by the sumcheck polynomial. Note that since the verifier used the same randomness across all the sumcheck protocols, it is the same value r^* for *all* the polynomials $\{\mathcal{G}^{(j)}\}_{j\in[k]}$. Since $\widetilde{\mathrm{eq}}(r^*, \tau)$ can be computed locally by the verifier, from Section A.3, the check at the end of the sumcheck reduces to computing, for each $j \in [k]$,

$$\widetilde{F}_{\mathrm{io}}^{(j)}(r^*) := \left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{A}'(r^*, y) \cdot (\mathrm{io}^{(j)}, 1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{A}''(r^*, y) \cdot w_y^{(j)} \right) \cdot \left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{B}'(r^*, y) \cdot (\mathrm{io}^{(j)}, 1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{B}''(r^*, y) \cdot w_y^{(j)} \right) - \left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{C}'(r^*, y) \cdot (\mathrm{io}^{(j)}, 1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{C}''(r^*, y) \cdot w_y^{(j)} \right)$$

The terms that depend solely on the instance and common input can be computed by the verifier locally. The remaining terms that depend on the witness, highlighted in red above, are terms that the prover needs to send to the verifier. We denote these terms by $\{v_{A,2}^{(j)}, v_{B,2}^{(j)}, v_{C,2}^{(j)}\}_{j\in[k]}$, where for each $j \in [k]$,

$$v_{A,2}^{(j)} := \sum_{y\in\{0,1\}^{s_2}} \widetilde{A}''(r^*, y) \cdot w_y^{(j)}, \qquad v_{B,2}^{(j)} := \sum_{y\in\{0,1\}^{s_2}} \widetilde{B}''(r^*, y) \cdot w_y^{(j)}$$

$$v_{C,2}^{(j)} := \sum_{y\in\{0,1\}^{s_2}} \widetilde{C}''(r^*, y) \cdot w_y^{(j)}$$

There are **two crucial observations** to be made here: (i) for each fixed r^*, the above values are simply a linear combination of each individual witness $w^{(j)}$ (with appropriate coefficients); and (ii) the linear coefficients are the *same for each witness*, and in fact the linear coefficients depend only on the index of the witness.

4. The above observations allow the prover to open the commitments to $\{v_{A,2}^{(j)}, v_{B,2}^{(j)}, v_{C,2}^{(j)}\}_{j\in[k]}$ by the linear homomorphism property of LHC since the property allows for the same linear coefficient to be applied *all* values in the k-tuple within the commitment.

   Specifically, the prover sends the values (and randomness) in the clear to the verifier, who then performs the same linear homomorphism over the committed values $\left\{c_y\right\}_{y\in[|w|]}$ to check if the openings sent by the prover are correct before computing the checks necessitated by the sumcheck.

   While we remarked that the values that are inside the commitment are binary, this is not true of the linear coefficients that are in $\mathbb{F}$. But since $\mathbb{F} = \mathbb{F}_2[\alpha]/(v(\alpha))$, by the *homomorphism with respect to polynomial* property of LHC and that $\mathbb{F}_2[\alpha]/(v(\alpha)) \subset \mathbb{Z}[\alpha]/(v(\alpha))$, it is fine that coefficents are in $\mathbb{F}$.

   Lastly it should be noted that the homomorphism properties are defined over $\mathbb{Z}[\alpha]/(v(\alpha))$ and not $\mathbb{F}_2[\alpha]/(v(\alpha))$, meaning there is no modular reduction in the coefficients of the resultant polynomial after the homomorphism operation, i.e. the λ length vector has elements in $\mathbb{Z}$ instead of $\mathbb{F}_2$. Therefore, to verify correctness of the commitment, the prover *computes the values* $\{v_{A,2}^{(j)}, v_{B,2}^{(j)}, v_{C,2}^{(j)}\}_{j\in[k]}$ *over* $\mathbb{Z}[\alpha]/(v(\alpha))$, i.e. no modular reduction. The verifier does the commitment check over $\mathbb{Z}[\alpha]/(v(\alpha))$, but once the check passes successfully reduces to $\mathbb{F}_2[\alpha]/(v(\alpha))$[14]. While this increases the number of bits sent by the prover, from Lemma 1, we see that the increase is quite small and is quantified in our main Theorem below.

Theorem 8. *Assuming the existence of somewhere-extractable linearly homomorphic commitments, the protocol in Figure 4 is a* dual-mode interactive batch argument *for* R1CS *where*

- *Total communication cost is* $O(m\lambda + (\lambda + k)\log m)$
- *The verifier's total run time is* $O(k|io| + n + m) \cdot \mathrm{poly}(\lambda)$

where all instances have the same length $|io|$.

Proof. We prove the desired properties below.

Completeness The completeness follows from (i) linear homomorphism property of the commitment scheme LHC; (ii) correctness of the sumcheck protocol for the polynomial $\mathcal{G}$ from Theorem 2; and (iii) polynomials in Section A.3.

The completeness is discussed in detail when we discussed the construction in Section 5.

[14]This is just reducing each element in the λ length vector to $\mathbb{F}_2$.

Dual Mode Indistinguishability This follows directly from the key indistinguishability of the somewhere-extractable linearly homomorphic commitment scheme LHC.

Non-adaptive statistical soundness We prove stronger a soundness guarantee than necessitated by the definition in Section 6, which will imply non-adaptive statistical soundness of the protocol. The stronger property will be crucial to the application of the Fiat-Shamir paradigm.

Costs. We denote below the costs for the batch verification protocol:

V**'s cost:** The verifier's cost includes the following, where X below indicates values A, B or C:

1. The cost to run k sumcheck protocols over a degree 2 polynomial with $\log m$ variables, which is $O(\log m)$ operations over the ring $\mathbb{Z}[\alpha]/(v(\alpha))$.
2. The cost to compute c_X is the cost to compute a *single* instance of $v_{X,2}$, which is $O(\lambda(n+m+|w|)) = O(\lambda(n+m))$ group operations.
3. The cost to check if the commitment is computed correctly corresponds to the cost of committing to a $\lambda \cdot k$ tuple, i.e. $O(\lambda k)$ group operations.
   The additional λ comes from the fact that the verifier has to commit to values in the ring in order to perform the check.
4. Lastly, the cost of computing k values $v_{X,1}^{(j)}$ is $O(k|\mathsf{io}^{(j)}|+n+m)$ (see Section 5 for details).

communication cost The commitment in the first message has a total cost of $O(2^s)\cdot|\mathsf{LHC.Com}| = O(m\cdot\lambda)$. The verifier then sends $O(\lambda \log m)$ bits. During the sumcheck, the total communication cost is $O(\log m)$ fields elements or $(\lambda \log m)$ bits. Finally when the commitments are opened, the prover sends $O(k \log m)$ bits (see Lemma 1). This yields a total communication cost of $O(m\lambda + (\lambda+k)\log m)$.
Note that there is a slight increase in communication cost due to the fact the prover sends the opening to the commitment without performing a modular reduction. But as observed from Lemma 1, this does not affect the cost by more than a logarithmic factor in m.

$\square$

Since we start with Boolean circuit satisfiability to generate R1CS instances, we get the following corollary with costs corresponding to the size the Boolean circuit.

Corollary 1. *If we start with* C-SAT *instances defined by a boolean circuit* $C : \{0,1\}^{|x|} \times \{0,1\}^{|y|} \mapsto \{0,1\}$, *then the protocol in Figure 4 is a* dual-mode interactive batch argument *for* C-SAT *where*

- *Total communication cost is* $O(|C| + k \log |C|)\mathsf{poly}(\lambda)$
- *The verifier's total run time is* $O(k|x| + |C|) \cdot \mathsf{poly}(\lambda)$

6 Strong Fiat-Shamir Compatibility

To make our (public-coin) protocol in Section 5, we want to apply the Fiat-Shamir transformation to the protocol, relative to some hash family. We describe the transformation in Section 7.1. But before we can get to the transformation, we need to show that our interactive protocol satisfies additional properties that make it suitable to prove soundness when we do apply the transformation.

In this section, we shall we shall find it convenient to follow the methodology presented in [JKKZ21]. The authors in [JKKZ21] define the notion of a *Fiat-Shamir(*FS*)-Compatible Interactive Proof*, which are public-coin interactive proof systems that satisfy some additional properties. In this work, we strengthen the requirements from FS compatible protocols denoting such protocols *Strongly* FS*-Compatible Interactive Proof.*

We start by defining the necessary properties for a protocol to be strongly FS-compatibility in Section 6.1. Then in Section 6.2 we show that our protocol in Section 5 in the *trapdoor mode* is strongly FS-compatible, which also implies the non-adaptive statistical soundness, completing Theorem 4.

6.1 Definition

In this section, we present the definition of Strongly FS-Compatible Interactive Proofs. As a starting step, we first recall the definition of *round-by-round* soundness for interactive proofs from [CCH+19].

Definition 9 (Round-by-Round Soundness [CCH+19])**.** *Let* $\Pi = (\mathsf{P}, \mathsf{V})$ *be a public-coin interactive protocol for a promise language* $L = (L_{\mathsf{YES}}, L_{\mathsf{NO}})$*, and* $d(\lambda)$ *be a function of* λ*. Let* Q *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* $x \in L_{\mathsf{YES}}$, $\mathsf{State}(x, \phi) = \mathsf{Accept}$. *For any* $x \in L_{\mathsf{NO}}$, $\mathsf{State}(x, \phi) = \mathsf{Reject}$.
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* $\mathcal{A}$*,*
$$\Pr\left[\alpha \leftarrow \mathcal{A}(1^\lambda, x, \mathsf{trans}), \beta \leftarrow Q : \mathsf{State}(x, \mathsf{trans}|\alpha|\beta) = \mathsf{Accept}\right] \le d(\lambda)/|Q|.$$
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 $d\ell/|Q|$. This follows from a simple application of the union bound over each round.

We now present the definition of Strongly FS-Compatible Interactive Proofs.

Definition 10 (Strong FS-Compatible Interactive Proofs)**.** *Let* $d(\lambda), T_{\mathsf{PreComp}}$ *and* $\rho(\lambda)$ *be functions of* λ*. We say an* $\ell(\lambda)$*-round interactive protocol* $\Pi = (\mathsf{P}, \mathsf{V})$ *for a promise language* $\bar{L} = (L_{\mathsf{YES}}, L_{\mathsf{NO}})$ *satisfies* $(d(\lambda), T_{\mathsf{PreComp}}, \rho(\lambda))$*-strong-*FS*-compatibility, if the following conditions hold.*

Round-by-Round Soundness. Π *satisfies* $d(\lambda)$*-round-by-round soundness.*

Bad Challenge Function. *There exists an algorithm* BAD *such that, for any instance* $x \in L_{\mathsf{NO}}$*, any* $i \in [\ell]$*, any transcript prefix* $\mathsf{trans} = (\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_{i-1}, \beta_{i-1})$ *such that* $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$*, and any prover's i-th round message* α*,* $\mathsf{BAD}(x, \mathsf{trans}|\alpha; r)$ *outputs an uniformly random element in* $\mathcal{B}$*, where* $\mathcal{B}$ *is defined as*

$$\mathcal{B} = \{\beta \in Q \mid \mathsf{State}(x, \mathsf{trans}|\alpha|\beta) = \mathsf{Accept}\}.$$

If $\mathcal{B}$ *is empty,* $\mathsf{BAD}(x, \mathsf{trans}|\alpha; r)$ *outputs* $\perp$*. We require* BAD *to satisfy the following two properties:*

Efficient: *The* BAD *function can be computed in polynomial time.*

Low Depth Computation: *The* BAD *algorithm can be decomposed to a (deterministic) polynomial time algorithm* PreComp *and a family of* TC^0 *circuits* $\{\mathsf{OnlineBAD}_\lambda\}_\lambda$ *such that, for any instance* x*, any* $i \in [\ell]$*, transcript prefix and the prover's* i*-th round message* α*,*

$$\mathsf{BAD}(x, \mathsf{trans}|\alpha; r) = \mathsf{OnlineBAD}_{\lambda,[r]}(\mathsf{PreComp}(1^\lambda, x, \mathsf{trans}|\alpha)),$$

where the circuit $\mathsf{OnlineBAD}_\lambda$ *has the randomness* r *hardwired in it, and the size of the circuit* $\mathsf{OnlineBAD}_\lambda$ *is bounded by* $\rho(\lambda)$*, and the running time of* $\mathsf{PreComp}(1^\lambda, x, \mathsf{trans}|\alpha)$ *is bounded by* $T_{\mathsf{PreComp}}(\lambda)$*.*

6.2 Proof of Strong Fiat-Shamir Compatibility

In this section, we prove that our protocol in Section 5 in the *trapdoor mode* is $(d, T_{\mathsf{PreComp}}, \rho)$-strongly FS-compatible, where $d = O(1), T_{\mathsf{PreComp}} = \mathsf{poly}(\lambda) \cdot |C|, \rho = \mathsf{poly}(\lambda, k, |C|)$.

We start by proving the round-by-round soundness in Section 6.2.1, the existence of efficient BAD challenge functions in Section 6.2.2, and its low-depth computation property in Section 6.2.3. An immediate consequence of proving the round-by-round soundness of our protocol in the *trapdoor mode* is that it implies the

To prove the properties need for strong FS-compatibility, we define the following promise language $\bar{L}_{i^*} = (L_{\mathsf{YES}}, L_{\mathsf{NO}})$ for k R1CS instances:

- **Yes Instance:** All R1CS instances $\{\mathbb{x}^{(i)}\}_{i\in[k]}$ are satisfiable.
- **No Instance:** The i^*-th R1CS instance $\mathbb{x}^{i^*}$ is unsatisfiable.

6.2.1 Proof of Round-by-Round Soundness

We start by describing the function State. For any $x \in L_{\mathsf{YES}}$, we define $\mathsf{State}(x, \phi) = \mathsf{Accept}$, and for any $x \in L_{\mathsf{NO}}$, define $\mathsf{State}(x, \phi) = \mathsf{Reject}$. Let $k = \mathsf{poly}(\lambda)$ be a polynomial, and let $i^* \in [k]$ be an index. Let (K, td) be an extraction mode key generated by $\mathsf{ExtGen}(1^\lambda, 1^k, i^*)$ during the the CRS generation in the *trapdoor mode* of the protocol. We will start by extracting the witness from the commitments sent by the prover, and then split the descrption of the State function depending on whether the input transcript to State is in the "zero-testing" phase, or the sumcheck phase.

Witness Extraction. The prover sends the commitments $c = \{c_y\}_{y\in\{0,1\}^{s_2}}$.

We use the trapdoor td to extract the witness $w_y \leftarrow \mathsf{Ext}(c_y, \mathsf{td})$, for all $y \in \{0,1\}^{s_2}$, where $w_y \in \mathbb{F}_2$. This corresponds to extracting the i^*-th puported witness.

Next, let $F^*(x)$ be the following polynomial, defined with respect to the extracted witnesses

$$\begin{aligned} F^*(x) = &\left(\sum_{y\in\{0,1\}^{s_1}} A'(x,y)\cdot(\mathrm{io},1)_y^{(i^*)} + \sum_{y\in\{0,1\}^{s_2}} A''(x,y)\cdot w_y\right)\cdot \\ &\left(\sum_{y\in\{0,1\}^{s_1}} B'(x,y)\cdot(\mathrm{io},1)_y^{(i^*)} + \sum_{y\in\{0,1\}^{s_2}} B''(x,y)\cdot w_y\right) \\ &-\left(\sum_{y\in\{0,1\}^{s_1}} C'(x,y)\cdot(\mathrm{io},1)_y^{(i^*)} + \sum_{y\in\{0,1\}^{s_2}} C''(x,y)\cdot w_y\right). \end{aligned} \quad (3)$$

Correspondingly, let $Q^*(t_1, t_2, \dots, t_s) \in \mathbb{F}_2[t_1, t_2, \dots, t_s]$ be the multivariable polynomial

$$Q^*(t) = \sum_{x \in \{0,1\}^s} F^*(x) \cdot \widetilde{\mathrm{eq}}(x, t) \tag{4}$$

where $t = (t_1, t_2, \dots, t_s)$.

Zero-Testing. In this stage, the verifier sends an uniform random $\tau \leftarrow \mathbb{F}^s$. Let $\tau = (\tau_1, \tau_2, \dots, \tau_s)$, where $\tau_i \in \mathbb{F}$. We view this step as $2s$-rounds, where the verifier firstly sends τ_1 in the first round, the sender sends nothing in the second round, and the verifier sends τ_2 in the third round etc. Now for each $i \in [s]$, we define $\mathsf{State}(x, c, \tau_1, \tau_2, \dots, \tau_i)$ as

$$\mathsf{State}(x, c, \tau_1, \tau_2, \dots, \tau_i) = \begin{cases} \text{Accept}, & \text{If } Q^*(t)|_{t_1=\tau_1, t_2=\tau_2, \dots, t_i=\tau_i} = 0 \in \mathbb{F}[t_{i+1}, t_{i+1}, \dots, t_s]. \\ \text{Reject}, & \text{Otherwise.} \end{cases}$$

For the sake of simplicity, in subsequent proof, we denote $\mathsf{prefix} = (x, c, \tau_1, \tau_2, \dots, \tau_s)$ as the transcript when the zero-testing stage completes.

k-Parallel Sumchecks. In this stage, the prover and the verifier run the sumcheck protocol for k parallel R1CS instances, with same verifier's messages (i.e. randomness).

Let multivariate polynomial $g^*(x_1, x_2, \dots, x_s) \in \mathbb{F}[x_1, x_2, \dots, x_s]$ be the following polynomial:

$$g^*(x) = F^*(x) \cdot \widetilde{\mathrm{eq}}(x, \tau). \tag{5}$$

For any $i \in [s]$, let $\alpha_1 = \{g_1^{(j)}(x)\}_{j \in [k]}, \alpha_2 = \{g_2^{(j)}(x)\}_{j \in [k]}, \dots, \alpha_i = \{g_i^{(j)}(x)\}_{j \in [k]}$ be the univariate polynomials the prover sent from the 1st round to the i-th round, and let $x'_1, x'_2, \dots, x'_i \in \mathbb{F}$ be the verifier's messages. For $i \in [s]$, let $g_i^*(x) \in \mathbb{F}[x]$ be the following polynomial.

$$g_i^*(x) = \sum_{x_{i+1}, x_{i+2}, \dots, x_s \in \{0,1\}} g^*(x'_1, x'_2, \dots, x'_{i-1}, x, x_{i+1}, x_{i+2}, \dots, x_s) \tag{6}$$

Define the State function for sumcheck as follows.

$$\mathsf{State}(\mathsf{prefix}, \alpha_1, x'_1, \alpha_2, x'_2, \dots, \alpha_{i-1}, x'_{i-1}, \alpha_i, x'_i) = \begin{cases} \text{Accept}, & \begin{array}{c}\text{If } g_i^{(i^*)}(x'_i) = g_i^*(x'_i), \text{ and} \\ \forall i' \in [i], g_{i'}^{(i^*)}(0) + g_{i'}^{(i^*)}(1) = v_{i'}^{(i^*)}. \end{array} \\ \text{Reject}, & \text{Otherwise.} \end{cases}$$

where $v_0^{(i^*)} = 0$, and $v_{i'}^{(i^*)} = g_{i'-1}^{(i^*)}(x_{i'-1})$, for any $i' \geq 1$.

Property 1. The first property of round-by-round soundness is satisfied by the definition of State.

Property 2. We prove the second property by examining the State function for the zero-testing and the k-parallel sumcheck.

- **Zero-Testing:** For any integer $i \in [s]$, and any transcript prefix $(c, \tau_1, \tau_2, \dots, \tau_{i-1})$, if $\mathsf{State}(x, c, \tau_1, \tau_2, \dots, \tau_{i-1}) = \text{Reject}$, then $Q^*(t_1, t_2, \dots, t_s)|_{t_1=\tau_1, t_2=\tau_2, \dots, t_{i-1}=\tau_{i-1}} \neq 0 \in \mathbb{F}[t_i, t_{i+2}, \dots, t_s]$. Since the polynomial Q^* has individual degree 1, after some restrictions, Q^* still has individual degree 1. Hence, we can represent $Q^*|_{t_1=\tau_1, t_2=\tau_2, \dots, t_{i-1}=\tau_{i-1}}$ as

$$Q^*(t_1, t_2, \dots, t_s)|_{t_1=\tau_1, t_2=\tau_2, \dots, t_{i-1}=\tau_{i-1}} = \sum_{e_i, e_{i+1}, \dots, e_s \in \{0,1\}} q^*_{e_i, e_{i+1}, \dots, e_s} t_i^{e_i} t_{i+1}^{e_{i+1}} \dots t_s^{e_s} \tag{7}$$

$$= \sum_{e_{i+1}, e_{i+2}, \dots, e_s \in \{0,1\}} (q^*_{0, e_{i+1}, e_{i+2}, \dots, e_s} + q^*_{1, e_{i+1}, e_{i+2}, \dots, e_s} t_i) \cdot t_{i+1}^{e_{i+1}} \dots t_s^{e_s}, \tag{8}$$

where at least one of the coefficient $q^*_{e_i,e_{i+1},e_s}$ is non-zero. Hence,

$$\begin{aligned}&\Pr[\beta_i \leftarrow \mathbb{F} : \mathsf{State}(x, \beta_1, \beta_2, \ldots, \beta_{i^*}) = \mathsf{Accept}]\\ \leq &\Pr[\beta_i \leftarrow \mathbb{F} : \forall e_{i+1}, e_{i+2}, \ldots, e_s \in \{0,1\}, q^*_{0,e_{i+1},e_{i+2},\ldots,e_s} + q^*_{1,e_{i+1},e_{i+2},\ldots,e_s}\beta_i = 0 \in \mathbb{F}] \leq 1/|\mathbb{F}|\end{aligned}$$

– **Sumcheck:** For any $i \in [s]$, let $(\mathsf{prefix}, \alpha_1, x'_1, \ldots, \alpha_{i-1}, x'_{i-1})$ be a transcript prefix such that

$$\mathsf{State}(\mathsf{prefix}, \alpha_1, x'_1, \alpha_2, x_2, \ldots, \alpha_{i-1}, x'_{i-1}) = \mathsf{Reject}.$$

We have two cases:

- **For $i = 1$:** In this case, $\mathsf{State}(\mathsf{prefix}) = \mathsf{Reject}$. By the definition of State, we have $Q^*(\tau_1, \tau_2, \ldots, \tau_s) \neq 0$. Since $Q^*(\tau_1, \tau_2, \ldots, \tau_s) = g^*_1(0) + g^*_1(1)$, we know that $g^*_1(0) + g^*_1(1) \neq 0$. Hence, any polynomial $g_1^{(i^*)}(x)$ sent by the prover with $g_1^{(i^*)}(0) + g_1^{(i^*)}(1) = v_0 = 0$ can not be $g^*_1(x)$. Hence,

$$\begin{aligned}\Pr_{\alpha_1 \leftarrow \mathcal{A}(\mathsf{prefix})}\left[x'_i \leftarrow \mathbb{F} : \mathsf{State}(\mathsf{prefix}, \alpha_1, x'_1) = \mathsf{Accept}\right] &\leq \Pr\left[x'_i \leftarrow \mathbb{F} : g_1^{(i^*)}(x'_i) = g^*_1(x'_i)\right]\\ &\leq O(1)/|\mathbb{F}|.\end{aligned}$$

The last inequality follows from the Schwartz-Zipple Lemma, and $\deg g_1^{(i^*)} = O(1)$.

- **For $i \geq 2$:** In this case, $\mathsf{State}(\mathsf{prefix}, \alpha_1, x'_1, \ldots, \alpha_{i-1}, x'_{i-1}) = \mathsf{Reject}$. We only need to prove the case when $\forall i' \in [i-1], g_{i'}^{(i^*)}(0) + g_{i'}^{(i^*)}(1) = v_{i'}^{(i^*)}$ but $g_{i-1}^{(i^*)}(x'_{i-1}) \neq g^*_{i-1}(x'_{i-1})$. We have

$$\begin{aligned}&\Pr_{\alpha_1 \leftarrow \mathcal{A}(\mathsf{prefix}, \alpha_1, x'_1, \ldots, a_{i-1}, x'_{i-1})}\left[x'_i \leftarrow \mathbb{F} : \mathsf{State}(\mathsf{prefix}, \alpha_1, x'_1, \alpha_2, x'_2, \ldots, \alpha_i, x'_i) = \mathsf{Accept}\right]\\ =&\Pr_{\alpha_1 \leftarrow \mathcal{A}(\mathsf{prefix}, \alpha_1, x'_1, \ldots, a_{i-1}, x'_{i-1})}[x'_i \leftarrow \mathbb{F} : g_i^{(i^*)}(x'_i) = g^*_i(x'_i) \wedge g_i^{(i^*)}(0) + g_i^{(i^*)}(1) = v_i^{(i^*)}]\\ =&\Pr_{\alpha_1 \leftarrow \mathcal{A}(\mathsf{prefix}, \alpha_1, x'_1, \ldots, a_{i-1}, x'_{i-1})}[x'_i \leftarrow \mathbb{F} : g_i^{(i^*)}(x'_i) = g^*_i(x'_i) \wedge g_i^{(i^*)} \neq g^*_i] \leq O(1)/|\mathbb{F}|.\end{aligned}$$

The first equality follows from the definition of State. The second equality follows from the fact that $v_i^{(i^*)} = g_{i-1}^{(i^*)}(x_{i-1}) \neq g^*_{i-1}(x'_{i-1})$, and $g^*_{i-1}(x'_{i-1}) = g^*_i(0) + g^*_i(1)$. Hence, $g_i^{(i^*)}(0) + g_i^{(i^*)}(1) = v_i^{(i^*)}$ implies $g_i^{(i^*)} \neq g^*_i$. The last inequality follows from the Schwartz-Zippel lemma, and $\deg g^*_i = O(1)$.

Property 3. For any complete transcript $\mathsf{trans} = (c, \tau, \alpha_1, x'_1, \alpha_2, x'_2, \ldots, \alpha_s, x'_s)$, if $\mathsf{State}(x, \mathsf{trans}) = \mathsf{Reject}$, and there exists an $i \in [s]$ such that the checking $g_i^{(i^*)}(0) + g_i^{(i^*)}(1) = v_i(i^*)$ fails, then verification done by the sumcheck verifier also fails. Hence, $\mathsf{V}(x, \mathsf{trans}) = 0$. Now, if the sumcheck verification passes, and $g_s^{(i^*)}(x'_s) \neq g^*_s(x'_s)$, then $g^*_s(x'_s) \neq v^{(i^*)}$, i.e.

$$g^*(x'_1, x'_2, \ldots, x'_s) = F^*(x'_1, x'_2, \ldots, x'_s) \cdot \widetilde{\mathsf{eq}}((x'_1, x'_2, \ldots, x'_s), \tau) \neq v^{(i^*)}.$$

By the verification process, for any $X \in \{A, B, C\}$, if the opening $(v_{X,0}^{(i^*)}, r_X)$ to $c_X = \sum_{y \in \{0,1\}^{s_2}} \widetilde{X}(\tau, y) \cdot c_y$ is correct, then by the somewhere $\mathbb{F}_2$-extraction property of the commitment, we have

$$v_{X,0}^{(i^*)} \pmod 2 = \mathsf{LHC.Ext}\left(\sum_{y \in \{0,1\}^{s_2}} \widetilde{X}(\tau, y) \cdot c_y, \mathsf{td}\right).$$

By the linear homomorphic extraction with respect to polynomials property of the commitment, we have

$$\begin{aligned}\mathsf{LHC.Ext}\left(\sum_{y\in\{0,1\}^{s_2}} \widetilde{X}(\tau,y)\cdot c_y, \mathsf{td}\right) &= \sum_{y\in\{0,1\}^{s_2}} \widetilde{X}(\tau,y)\cdot \mathsf{LHC.Ext}(c_y,\mathsf{td}) \pmod 2\\ &= \sum_{y\in\{0,1\}^{s_2}} \widetilde{X}(\tau,y)\cdot w_y \in \mathbb{F}\end{aligned}$$

Hence, in the verification,

$$\begin{aligned}&(v_A^{(i^*)}\cdot v_B^{(i^*)} - v_C^{(i^*)})\cdot \widetilde{\mathrm{eq}}((x_1',x_2',\ldots,x_s'),\tau) \pmod 2\\ =&F^*(x_1',x_2',\ldots,x_s')\cdot \widetilde{\mathrm{eq}}((x_1',x_2',\ldots,x_s'),\tau) \pmod 2 \neq v^{(i^*)}.\end{aligned}$$

Then the checking for $v^{(i^*)}$ must fail, and thus $\mathsf{V}(x,\mathsf{trans}) = 0$.

6.2.2 Proof of Efficient BAD Challenge Function Property

Now, we proceed to define the BAD functions for each round and prove the required properties.

For any instance $x \in L_{\mathrm{NO}}$, let trans be a transcript prefix such that $\mathsf{State}(x,\mathsf{trans}) = \mathsf{Reject}$. We consider the zero-testing stage and sumcheck stage separately.

Zero-Testing. Let $\mathsf{trans} = (c,\tau_1,\tau_2,\ldots,\tau_{i-1})$ be a transcript prefix, where $i\in[s]$. Let

$$\mathcal{B} = \{\tau \in Q \mid \mathsf{State}(x,\mathsf{trans}|\tau) = \mathsf{Accept}\}$$

From $\mathsf{State}(x,\mathsf{trans}) = \mathsf{Reject}$, we know that $Q^*(t_1,t_2,\ldots,t_s)|_{t_1=\tau_1,t_2=\tau_2,\ldots,t_{i-1}=\tau_{i-1}} \neq 0 \in \mathbb{F}$.

In Equation 7, the coefficients $q^*_{0,e_{i+1},e_{i+2},\ldots,e_s}$ and $q^*_{1,e_{i+1},e_{i+2},\ldots,e_s}$ can be computed from w_y in TC^0 circuit.

To satisfy $\mathsf{State}(x,\mathsf{trans}|\tau) = \mathsf{Accept}$, τ needs to satisfy the following equations.

$$q^*_{0,e_{i+1},e_{i+2},\ldots,e_s} + \tau\cdot q^*_{1,e_{i+1},e_{i+2},\ldots,e_s} = 0, \forall e_{i+1},e_{i+2},\ldots,e_s \in \{0,1\} \tag{9}$$

To solve this equation, it suffices to find a non-zero $q^*_{1,e_{i+1},e_{i+2},\ldots,e_s}$, and find a candidate solution x_*, then check whether x_* satisfies all the equations. Hence, we define the BAD function in Figure 5.

Sumcheck. Let $\mathsf{trans} = (c,\tau,\alpha_1,\beta_1,\alpha_2,\beta_2,\ldots,\alpha_{i-1},\beta_{i-1})$ be a transcript prefix, where $i\in[s]$. Similarly, we firstly extract the witness w_y, and then drive $g^*(x), g_i^*(x)$ from the extracted witness. we define the bad challenge function for sumcheck as in Figure 6.

6.2.3 Proof of Low-Depth BAD Function Property

We need to show that the BAD challenge for both zero-testing and sumcheck can be computed in TC^0. As we will show below, this is already true for the BAD function in the zero-testing phase defined in Figure 5. But unfortunately we do not know how to prove this property for the sumcheck protocol, as it is currently defined. Instead, we show that due to the specific polynomial(s) that the sumcheck protocol is defined over, one can modify slightly the sumcheck protocol to allow for the BAD function to be computed in TC^0. So the low depth property of the BAD function is proven for our Figure 4 protocol in the *trapdoor mode* with a **modified sumcheck protocol**.[15]

[15]We chose not to present our protocol with the modified sumcheck since the low-depth property is not necessary to prove soundness of the Fiat-Shamir transformation. Instead, the underlying assumptions for the soundness will differ depending on whether the low-depth property is achieved. See Section 7.1 for details.

$\underline{\mathsf{BAD}(x, \mathsf{trans} = (c, \tau_1, \tau_2, \ldots, \tau_{i-1}))}$

1. Parse $c = \{c_y\}_{y \in \{0,1\}^{s_2}}$, and extract the witness $w_y \leftarrow \mathsf{LHC.Ext}(c_y, \mathsf{td})$, for any $y \in \{0,1\}^{s_2}$.
2. Computing $q^*_{0,e_{i+1},e_{i+2},\ldots,e_s}$ and $q^*_{1,e_{i+1},e_{i+2},\ldots,e_s}$ from $\{w_y\}_{y \in \{0,1\}^{s_2}}$, for any $e_{i+1}, e_{i+2}, \ldots, e_s \in \{0,1\}$, as follows.
   - Compute $F^*_x = F^*(x)$ from $\{w_y\}_{y \in \{0,1\}^{s_2}}$ for any $x \in \{0,1\}^s$ using Equation 3.
   - Using Equation 4, expand $Q^*(t_1, t_2, \ldots, t_s)$ to the linear combination of the monomials $\{t_1^{e_1} t_2^{e_1} \ldots t_s^{e_s}\}_{e_1,e_2,\ldots,e_s \in \{0,1\}}$, i.e. compute $q_{e_1,\ldots,e_s}$ s.t.
   $$Q^*(t_1, \ldots, t_s) = \sum_{e_1,\ldots e_s \in \{0,1\}} q_{e_1,\ldots,e_s} t_1^{e_1} \ldots t_s^{e_s}.$$
   - Compute $q^*_{e_i,\ldots,e_s}$ using Equation 7, i.e.
   $$q^*_{e_i,\ldots,e_s} = \sum_{e_1,\ldots,e_{i-1} \in \{0,1\}} q_{e_1,\ldots,e_s} \tau_1^{e_1} \tau_2^{e_2} \ldots \tau_{i-1}^{e_{i-1}}.$$
3. Find the smallest $(e'_{i+1}, e'_{i+2}, \ldots, e'_s) \in \{0,1\}^{s-i}$ such that $q^*_{1,e'_{i+1},\ldots,e'_s} \neq 0$.
   If such $(e'_{i+1}, e'_{i+2}, \ldots, e'_s)$ does not exist, then output $\perp$.
4. Let $x_* = -q^*_{0,e'_{i+1},e'_{i+2},\ldots,e'_s} / q^*_{1,e'_{i+1},e'_{i+2},\ldots,e'_s} \in \mathbb{F}$.
5. If $q^*_{0,e_{i+1},e_{i+2},\ldots,e_s} + x_* \cdot q^*_{1,e_{i+1},e_{i+2},\ldots,e_s} = 0$, for any $e_{i+1}, e_{i+2}, \ldots, e_s \in \{0,1\}$, then output x_*.
   Otherwise output $\perp$.

Figure 5: BAD function for zero-testing stage.

$\underline{\mathsf{BAD}(x, \mathsf{trans} = (c, \tau, \alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_{i-1}, \beta_{i-1}) | \alpha)}$

1. Parse $c = \{c_y\}_{y \in \{0,1\}^{s_2}}$, and extract the witness $w_y \leftarrow \mathsf{LHC.Ext}(c_y, \mathsf{td})$, for any $y \in \{0,1\}^{s_2}$.
2. Compute $g^*(x)$ from $\{w_g\}_{g \in \{0,1\}^{s_2}}$, using Equation 3 and Equation 5.
3. Compute $g_i^*(x)$ from $g^*(x)$, using Equation 6.
4. Parse $\alpha = \{g^{(j)}(x)\}_{j \in [k]}$. Let Root be the roots of the cubic equation $g_i^*(x) - g^{(i^*)}(x) = 0$.
5. Uniformly sample an element in Root, and output it.

Figure 6: BAD function for sumcheck stage.

Special sumcheck. Let $g : \mathbb{F}^n \mapsto \mathbb{F}$ be an ℓ-variate polynomial of degree d in each variable. In the sumcheck protocol (Figure 1), the prover P_{SC} in each round sends a degree d univariate g_i to the verifier V_{SC}. We show that if the polynomial has a specific structure, then it suffices for P_{SC} to send a degree $d-1$ univariate g'_i to the verifier V_{SC} in each round. Specifically, we will require the g to be such that $\forall i \in [\ell]$

$$\sum_{x_1,\cdots,x_\ell \in \{0,1\}} g(x_1,\cdots,x_\ell) = \sum_{x_i \in \{0,1\}} f_i(x_i) \cdot \left(\sum_{x_1,\cdots,x_{i-1},x_{i-1},\cdots x_\ell \in \{0,1\}} h_i(x_1,\cdots,x_\ell) \right)$$

where f_i is a univariate polynomial of x_i with degree 1, meaning that h_i has degree $d-1$ in x_i. Further, in the context of sumcheck protocol, we will require that the (two) coefficients of f_i are "efficiently" computatble by V_{SC}. We have intentionally kept the efficiency requirements vague as we shall demonstrate it for the specific case of sumcheck protocol within our dual-mode interactive batch argument in Figure 4.

Note that we have focused on the sum over the Boolean hypercube since that is most relevant to our work, but the properties can be further generalized to any $H \subset \mathbb{F}$.

Although we want this property to hold in the batch setting, for simplicity we shall consider a single instance, and the extension follows trivially. In Figure 4, the verifier V samples a random string $\tau \in \mathbb{F}^s$, which defines the s variate polynomial $\mathcal{G}_{\mathrm{io},\tau}$ over which the sumcheck protocol is run. Let's revisit how $\mathcal{G}_{\mathrm{io},\tau}$ is defined,

$$\sum_{x \in \{0,1\}^s} \mathcal{G}_{\mathrm{io},\tau}(x) = \sum_{x \in \{0,1\}^s} \widetilde{F}_{\mathrm{io}}(x) \cdot \widetilde{\mathrm{eq}}(\tau, x) = \sum_{x \in \{0,1\}^s} \widetilde{F}_{\mathrm{io}}(x) \cdot \prod_{j=1}^{s} (\tau_j \cdot x_j + (1-\tau_j)(1-x_j))$$

where $\mathcal{G}_{\mathrm{io},\tau}(x)$ is a polynomial with individual degree 3 and $\widetilde{F}_{\mathrm{io}}$ is a polynomial with individual degree 2. From the above description, it immediately follows that $\forall i \in [s]$,

$$\begin{aligned}\sum_{x \in \{0,1\}^s} \mathcal{G}_{\mathrm{io},\tau}(x) = &\sum_{x_i \in \{0,1\}} (\tau_i \cdot x_i + (1-\tau_i)(1-x_i)) \cdot \\ &\left(\sum_{x_{\bar{i}} \in \{0,1\}^{s-1}} \widetilde{F}_{\mathrm{io}}(x) \cdot \prod_{\substack{j=1\\ j \neq i}}^{s} (\tau_j \cdot x_j + (1-\tau_j)(1-x_j)) \right) \\ = &\sum_{x_i \in \{0,1\}} ((2 \cdot \tau_i - 1) \cdot x_i + (1-\tau_i)) \left(\sum_{x_{\bar{i}} \in \{0,1\}^{s-1}} \widetilde{F}_{\mathrm{io}}(x) \cdot \prod_{\substack{j=1\\ j \neq i}}^{s} (\tau_j \cdot x_j + (1-\tau_j)(1-x_j)) \right)\end{aligned}$$

where $x_{\bar{i}}$ indicate all variables in x except x_i. Once τ is fixed, the coefficients for $f_i(x_i) \coloneqq 2 \cdot \tau_i - 1) \cdot x_i + (1-\tau_i)$ are immediately determined, and computable in $O(1)$ operations in $\mathbb{F}$. Let $h_i(x) \coloneqq \widetilde{F}_{\mathrm{io}}(x) \cdot \prod_{\substack{j=1\\ j \neq i}}^{s} (\tau_j \cdot x_j + (1-\tau_j)(1-x_j))$.

Then, the polynomial g_i sent in the i-th round of the sumcheck protocol in Figure 1, with $t_1, \cdots, t_{i-1}$ receiver messages in the $i-1$ rounds, is

$$g_i(x) = (2 \cdot \tau_i - 1) \cdot x + (1-\tau_i)) \sum_{b_{i+1},\cdots,b_s \in \{0,1\}} h_i(t_1,\cdots,t_{i-1},x,b_{i+1},\cdots,b_\ell)$$

The correctness follows directly from the correctness of the original sumcheck protocol and our definition of h_i.

[16] If the coefficients of the degree 2 polynomial $g'_i(x)$ are $(g'_{i,2}, g'_{i,1}, g'_{i,0})$, then the coefficients of g_i are $(g'_{i,2}(2\tau_i - 1), g'_{i,2}(1-\tau_i) + g'_{i,1}(2\tau_i - 1), g'_{i,0}(2\tau_i - 1) + g'_{i,1}(1-\tau_i), g'_{i,0}(1-\tau_i))$.

Special Sumcheck Protocol: $(\mathsf{P}'_{\mathsf{SC}}, \mathsf{V}'^{\mathcal{G}}_{\mathsf{SC}})$ **for** $\sum_{x\in\{0,1\}^s} \mathcal{G}_{\mathbf{io},\tau}(x) = 0$

Common input: $\tau \in \mathbb{F}^s, v \in \mathbb{F}$.

P′**'s auxiliary input:** polynomial $\mathcal{G}_{\mathrm{io},\tau}$ of individual degree 3

1. Set $i := 1, v_0 := v = 0$.
2. Prover $\mathsf{P}'_{\mathsf{SC}}$ computes the univariate polynomial $g'_i : \mathbb{F} \mapsto \mathbb{F}$ of degree 2

$$g'_i(x) := \sum_{b_{i+1},\cdots,b_\ell\in\{0,1\}} h_i(t_1,\cdots,t_{i-1},x,b_{i+1},\cdots,b_\ell)$$

   send g'_i to the verifier $\mathsf{V}^{g}_{\mathsf{SC}}$.
3. Verifier $\mathsf{V}'^{\mathcal{G}}_{\mathsf{SC}}$ does the following:
   (a) Check if g'_i is a univariate polynomial of degree at most 2.
   (b) Computes $g_i(x) := ((2\cdot\tau_i - 1)\cdot x + (1-\tau_i))\cdot g'_i(x)$.[16]
   (c) Check that $g_i(0) + g_i(1) = v_{i-1}$. If not, $\mathsf{V}'^{\mathcal{G}}_{\mathsf{SC}}$ rejects.
   (d) Choose a random element $t_i \leftarrow\$\, \mathbb{F}$
   (e) Set $v_i := g_i(t_i)$.
   (f) if $i < s$, set $i := i + 1$, send t_i to prover P_{SC} and go back to Step 2.
   (g) if $i = s$, check if $v_\ell = \mathcal{G}_{\mathrm{io},\tau}(t_1,\cdots,t_\ell)$ by querying the oracle $\mathcal{G}$ at the point $(t_1,\cdots,t_\ell)$.

Figure 7: Special sumcheck protocol $(\mathsf{P}'_{\mathsf{SC}}(g), \mathsf{V}'^{\mathcal{G}}_{\mathsf{SC}}(v))$

The soundness of the *special sumcheck protocol* in Figure 7 can be reduced to the soundness of sumcheck protocol in Figure 1. If P'^* is the cheating prover for the special sumcheck protocol, then it can be converted to a cheating prover P^* for the original sumcheck protocol - when P'^* sends g'_i in the i-th round, P^* simply multiplies to it $(2\cdot\tau_i - 1)\cdot x + (1-\tau_i))$ to get g_i which it then passes over to the verifier.

We now prove that after the modification, the BAD functions can be decomposed to PreComp and an OnlineBAD , where OnlineBAD can be computed in TC^0.

Lemma 9 (Low-Depth BAD Function). *If we take $v(\alpha) = \alpha^{2\cdot 3^l} + \alpha^{3^l} + 1$, where $l \geq 1$ is an integer, then the* BAD *challenge function satisfies low-depth computation property.*

Proof. We prove the lemma for the zero-testing stage and the sumcheck stage respectively.

Zero-Testing. The BAD function is described in Figure 5. We exam each steps.

1. We decompose the witness extraction as LHC.PreComp and LHC.OnlineExt, and put LHC.PreComp into PreComp, and LHC.OnlineExt into OnlineBAD. Since LHC.OnlineExt can be computed in TC^0, this step is in TC^0.

2. For $\{F_x^*\}_{x\in\{0,1\}^s}$, we compute them in parallel. Since the Equation 3 can be computed in TC^0, F_x^* for all x can be computed in TC^0.

   For $q^*_{e_i,e_{i+1},\dots,e_s}$, they can be computed by expanding $Q^*(t_1,t_2,\dots,t_s)$. Since the coefficient of the monomials can be computed as follows

$$q_{e_1,\dots e_s} = \sum_{x\in\{0,1\}^s} F_x^* \cdot \prod_{i=1}^{s}(x_i+1)\cdot(1-e_i).$$

   and the iterative multiplication in $\mathbb{F}_2[\alpha]/(v(\alpha))$ can be computed by TC^0 circuits [HV06], this step can also be computed by TC^0 circuits.

3. Finding the smallest index such that $q^*_{1,e_{i+1},\dots,e_s} \neq 0$ can also be done in TC^0.

4. The inversion $x_* = -q^*_{0,e'_{i+1},\dots,e'_s}/q^*_{1,e'_{i+1},\dots,e'_s}$ can be computed as $x_* = q^*_{0,e'_{i+1},\dots,e'_s}\cdot(q^*_{1,e'_{i+1},\dots,e'_s})^{|\mathbb{F}|-2}$. Since multiplication and exponentiation in $\mathbb{F}_2[\alpha]/(v(\alpha))$ can be computed in TC^0, this step is also in TC^0.

5. Finally, the checking $q^*_{0,e_{i+1},e_{i+2},\dots,e_s} + x_*\cdot q^*_{1,e_{i+1},e_{i+2},\dots,e_s} = 0$ is a polynomial of x_*, $q^*_{0,e_{i+1},e_{i+2},\dots,e_s}$ and $q^*_{1,e_{i+1},e_{i+2},\dots,e_s}$, and thus can be computed in TC^0.

Sumcheck. The BAD function is described in Figure 6. We exam each steps.

1. The extraction of w_y can be decomposed to LHC.PreComp and LHC.OnlineExt, similar to the extraction in zero-testing stage. Hence, this step can be computed in TC^0.

2. The computation of $g_i^*(x)$ by Equation 6 is a constant layer of iterative addition and multiplication, and hence can be computed in TC^0.

3. For the root finding of $g_i^*(x) - g^{(i^*)}(x) = 0$, we modify this step as follows.

   – Compute $g'^*_i(x) = g_i^*(x)/((2\tau_i - 1)\cdot x + (1-\tau_i))$.

   – Solving the quadratic equation $g'^*_i(x) - g'_i(x) = 0$. Then the roots of $g_i^*(x) - g^{(i^*)}(x) = 0$ are the roots of $g'^*_i(x) - g'_i(x) = 0$ and $\tau_i + 1$.

   The computation of the first step is in TC^0. By Lemma 10, the second step can also be computed in TC^0. Hence, $\mathsf{OnlineBAD}_\lambda$ is in TC^0.

□

In the following lemma, we use the techniques in [BRS67] to solve the quadratic equations in $\mathbb{F}$, and use the result of [HV06] to prove that the root finding algorithm can be computed in TC^0.

Lemma 10 (Quadratic Equation Solving in TC^0). *Let l be an integer, and let $\lambda = n = 2\cdot 3^l$. Let $\mathbb{F} = \mathbb{F}_2[\alpha]/(\alpha^{2\cdot 3^l} + \alpha^{3^l} + 1)$ be an extension field of $\mathbb{F}_2$, and let*

$$f(x) = ax^2 + bx + c$$

be a non-zero polynomial in $\mathbb{F}[x]$. There exists a TC^0 circuit Solve of size $\mathrm{poly}(\lambda)$ that takes as input a, b, c, and outputs a set Root, where

$$\mathrm{Root} = \{x \in \mathbb{F} \mid f(x) = 0\}.$$

Proof. The circuit Solve is constructed in Figure 8. We consider several cases.

Solve(a, b, c)

- **If** $a = b = 0$**, but** $c \neq 0$, let Root $= \phi$.
- **If** $a = 0$**, but** $b \neq 0$, let Root $= \{c/b\}$.
- **If** $a \neq 0$**,** then the equation is a quadratic equation.
  - **If** $b = 0$, let Root $= \{(c/a)^{|\mathbb{F}|/2}\}$.
  - **If** $b \neq 0$, we rewrite the equation as $\left(\frac{a}{b}x\right)^2 + \left(\frac{a}{b}x\right) = \frac{ca}{b^2}$. Let $T : \mathbb{F} \longrightarrow \mathbb{F}$, $T(y) = y + y^2$. If we view $\mathbb{F}$ as a linear vector space over field $\mathbb{F}_2$, then T is a linear transformation. Let Root $= \{\frac{b}{a} \cdot y \mid y \in T^{-1}(ca/b^2)\}$.

Output Root.

Figure 8: Equation solving algorithm Solve.

- **Degree** $f = 0$**.** Since $f(x) \neq 0$, then Root is empty.
- **Degree** $f = 1$**.** Then the equation degenerates to a linear equation, and has unique root c/b. By [HV06], the exponention of in $\mathbb{F}$ can be computed in TC^0, hence, $c/b = c \cdot b^{|\mathbb{F}|-2}$ can also be computed in TC^0.
- **Degree** $f = 2$**.** Then we have two cases.
  - **If** $b = 0$, then the unique root $(c/a)^{|\mathbb{F}|/2}$ can be computed in TC^0.
  - **If** $b \neq 0$, then the equation can be written as $\left(\frac{a}{b}x\right)^2 + \left(\frac{a}{b}x\right) = \frac{ca}{b^2}$. Let $T : \mathbb{F} \longrightarrow \mathbb{F}, T(y) = y + y^2$, then T is $\mathbb{F}_2$-linear over $\mathbb{F}$, and we only need to find a y such that $T(y) = ca/b^2$. To find such y, we firstly compute $T(\alpha^i) = \sum_{j=0}^{n-1} T_{i,j}\alpha^j$, where $T_{i,j} \in \mathbb{F}_2$. Let $\mathbf{T} = (T_{i,j})_{i,j\in[n]}$ be the matrix of $T_{i,j}$. Then we only need to solve a linear equation about $\mathbf{T}$. Let the rank of $\mathbf{T}$ be r.

    By LU decomposition, we can find a permutation matrix $\mathbf{P}$, a lower triangle non-singular matrix $\mathbf{L}$, and a upper triangle matrix $[\mathbf{U}_1|\mathbf{U}_2]$ such that

$$\mathbf{P} \cdot \mathbf{T} = \mathbf{L} \cdot \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix},$$

    where $\mathbf{U}_1 \in \mathbb{F}_2^{r\times r}$ is a $r \times r$ non-singular matrix. Now, we can express ca/b^2 as a polynomial in $\mathbb{F}_2[\alpha]$, i.e. $ca/b^2 = \sum_{i=0}^{n-1} z_i\alpha^i$. If we denote $y = \sum_{i=0}^{n-1} y_i\alpha^i$, and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})^T$, $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})^T$, then

$$T(y) = z \iff \mathbf{T} \cdot \mathbf{y} = \mathbf{z} \iff \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{y} = \mathbf{L}^{-1} \cdot \mathbf{P} \cdot \mathbf{z}$$

    If the last $n - r$ coordinates of $\mathbf{L}^{-1} \cdot \mathbf{P} \cdot \mathbf{z}$ is non-zero, then Root should be empty. Otherwise, let $\mathbf{z}'$ be the first r coordinates of $\mathbf{L}^{-1} \cdot \mathbf{P} \cdot \mathbf{z}$, and let

$$T^{-1}(ca/b^2) = \left\{\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2) \mid \mathbf{z}_2 \in \{0,1\}^{n-r}, \mathbf{z}_1 = \mathbf{U}_1^{-1} \cdot (\mathbf{z}_2 - \mathbf{U}_2 \cdot \mathbf{z}')\right\}.$$

Since $\mathbf{T}$ is a matrix determined only by the basis $\{\alpha^0, \alpha^1, \ldots, \alpha^{n-1}\}$ of the $\mathbb{F}_2$-linear space $\mathbb{F}$, the matrix $\mathbf{P}, \mathbf{T}, \mathbf{L}^{-1}, \mathbf{U}_1^{-1}, \mathbf{U}_2$ are fixed matrix, and thus can be hardwired in the OnlineBAD circuit. Hence, the computation of T^{-1} is in TC^0. In fact, since the quadratic equation has at most 2 roots, we have $r \geq n-1$. Hence, the size of OnlineBAD is bounded by a polynomial of λ.

□

7 Non-Interactive Batch Arguments for NP

We now construct a non-interactive batch argument system for NP. We start by formally defining this notion below.

Definition 11 (Non-Interactive Batch Arguments). *A non-interactive batch argument for an* NP *language L defined by relation $\mathcal{R}_L$ is a tuple of algorithms* (Gen, P, V) *satisfying the following properties:*

- **Completeness:** *For all $\mathbf{x} = (x_1, \ldots, x_k)$ and $\mathbf{w} = (w_1, \ldots, w_k)$ such that for each $i \in [k]$, $\mathcal{R}_L(x_i, w_i) = 1$, it holds that:*

$$\Pr[\mathsf{V}(\mathsf{crs}, \mathbf{x}, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda),\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathbf{x}, \mathbf{w})] = 1.$$

- **(Non-adaptive) Soundness:** *For every PPT adversary P^* and all $\mathbf{x} = (x_1, \ldots, x_k)$ where $\exists i$ s.t. $x_i \notin L$, it holds that:*

$$\Pr[\mathsf{V}(\mathsf{crs}, \mathbf{x}, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda),\ \pi \leftarrow \mathsf{P}^*(\mathsf{crs})] \leq \mathsf{negl}(\lambda).$$

In Section 7.1 we show that the Fiat-Shamir transform w.r.t. $\mathcal{H}$ when applied to any strongly FS compatible protocol is sound as long as $\mathcal{H}$ is correlation intractable for TC^0. Next, in Section 7.2, we construct a non-interactive batch argument system for NP by demonstrating that the above transformation remains sound when applied to our dual-mode interactive batch argument (Section 5), even though the aforementioned protocol does not satisfy strong FS compatibility. Finally, in Section 7.3, we show that we show that although our described protocol has a linear dependence on k, this can be made sub-linear.

7.1 Fiat-Shamir Transformation

To obtain a non-interactive protocol in the common reference string (CRS) model, we want to start with a public-coin interactive protocol and apply the Fiat-Shamir transformation to it. In the aforementioned transformation, a hash function is picked to be a part of the CRS, and then the prover non-interactively generates verfier's messages as the hash of the transcript so far. We make a couple of minor (largely cosmetic) modifications to the transformation: (a) firstly, we apply the transformation to interactive proofs that are already in the CRS setting for a promise language $\bar{L}$; and (b) prior to hashing, we require the prover to pre-process the transcript using a *public deterministic* function PreComp that can be computed in polynomial time. The rest of the transformation remains unchanged and is described in Figure 9.

To show the soundness of the transformation in Figure 9 when instantiated with a concrete hash function family $\mathcal{H}$, we build on recent progress on constructing an appropriate correlation intractable hash function family given a public coin interactive proof (see the related work in Section 1.2 for more details). Specifically, we show that if we start with a public-coin interactive protocol for some promise language $\bar{L}$ that is *strongly Fiat-Shamir compatible* (Definition 10), then it suffices for $\mathcal{H}$ to be correlation intractable against functions that can be computed in TC^0 (a milder requirement). Intuitively it suffices for $\mathcal{H}$ to be CI for TC^0 since strong FS-compatibility ensures that the function BAD can be computed in TC^0 (with appropriate pre-processing of the inputs).

Protocol: Non-interactive protocol $(\mathsf{P}_{\mathsf{FS}}, \mathsf{V}_{\mathsf{FS}})$ for $\bar{L}$ in the CRS Model

Common Reference String: $\mathsf{CRS} := (\mathsf{crs}, \{k_i\}_{i\in[\ell]})$ consisting of crs, the common reference string for the interactive protocol, and ℓ keys $\left\{k_i \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)\right\}_{i\in[\ell]}$

Common input: input x, security parameter 1^λ where $\lambda = \lambda(|x|)$.

P**'s auxiliary input:** witness w such that $\mathcal{R}_{L_{\mathsf{YES}}}(x, w) = 1$

1. Prover P_{FS} on input (x, w, CRS) sets $i := 1$, $\mathsf{trans} := \emptyset$ and does the following:
   For $i \in [\ell]$
   (a) Compute
   $$\alpha_i \leftarrow \mathsf{P}(\mathsf{crs}, x, w, \mathsf{trans}) \text{ and } \beta_i := \mathcal{H}.\mathsf{Hash}(k_i, \mathsf{PreComp}(1^\lambda, x, \mathsf{trans}|\alpha_i)).$$
   (b) Set $\mathsf{trans} := \mathsf{trans}|\alpha_i|\beta_i$.

   Output trans.
2. Verifier V_{FS} on input $(x, \mathsf{CRS}, \mathsf{trans})$ does the following
   For $i \in [\ell]$
   (a) Check $\beta_i \stackrel{?}{=} \mathcal{H}.\mathsf{Hash}(k_i, \mathsf{PreComp}(1^\lambda, x, \mathsf{trans}|\alpha_i))$ where $\mathsf{trans}_{i-1} := \alpha_1|\beta_1|\cdots|\alpha_{i-1}|\beta_{i-1}$.

   Accept if and only if $\mathsf{V}(\mathsf{crs}, x, \mathsf{trans})$ accepts.

Figure 9: Fiat-Shamir Transformation

Theorem 9 (Soundness of FS Transform). *Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an $\ell(\lambda)$ round $(d, T_{\mathsf{PreComp}}, \rho)$-strongly* FS *compatible protocol for some promise language $\bar{L}$ in the common reference string model. Denote the verifier's messages $\beta_1, \cdots, \beta_\ell \in \{0,1\}^\lambda$, denote the overall prover and verifier's runtimes by $T_{\mathsf{P}}(\lambda)$ and $T_{\mathsf{V}}(\lambda)$ respectively.*

Let n be an upper bound on the output of PreComp. *Suppose that there exists a $\mathcal{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ correlation intractable* (CI) *hash function family for* TC^0 *(Definition 7) where inputs of $n(\lambda)$ bits are mapped to λ bits. Then the resulting non-interactive protocol by applying the Fiat-Shamir transform w.r.t. the hash family $\mathcal{H}$ as described in Figure 9 has the following properties:*

Completeness. *If Π has completeness 1, then Π_{FS} also has completeness 1.*

Computational Soundness. *For any* PPT *cheating prover* P^*, *there exists a negligible function* negl *such that for every $x^* \in L_{\mathsf{NO}}$ and $\lambda = \lambda(|x^*|)$,*
$$\Pr[\mathsf{V}(\mathsf{CRS}, x^*, \pi) = 1 \mid \mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda),\ \pi \leftarrow \mathsf{P}^*(\mathsf{CRS})] \leq \mathsf{negl}(\lambda)$$

Efficiency. *There exists a polynomial* p *that depends on the* CI *hash family $\mathcal{H}$ such that the total verifier runtime is $\ell(\lambda)\cdot(T_{\mathsf{PreComp}}(\lambda) + \mathsf{p}(\rho(\lambda))) + T_{\mathsf{V}}(\lambda)$, and the total prover runtime is $\ell(\lambda)\cdot(T_{\mathsf{PreComp}}(\lambda) + \mathsf{p}(\rho(\lambda))) + T_{\mathsf{P}}(\lambda)$.*

Note that by the definition of $(d, T_{\mathsf{PreComp}}, \rho)$-strongly FS compatibility, ρ is a polynomial function.

Proof. Fix an interactive protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for promise language $\bar{L}$ in the common reference string model and fix a CI hash function family $\mathcal{H}$ for TC^0. We then prove the desired properties below. The proof structure largely follows [JKKZ21] with several simplifications for our setting.

Completeness. Completeness follows directly from the completeness of the underlying interactive protocol.

Computational Soundness. We prove soundness by the way of contradiction. Suppose soundness does not hold, there exists a PPT prover P^*, a polynomial $\mathsf{p}(\cdot)$ and an infinite set of *false* statements X of $x^* \in L_{\mathsf{NO}}$ such that for each $x^* \in X$, and $\lambda = \lambda(|x|)$,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}}[\mathsf{V}(\mathsf{CRS}, x^*, \pi) = 1] \geq \frac{1}{\mathsf{p}(\lambda)}$$

By assuming without loss of generality that each distinct x^* maps to a different λ, we get the above to hold for an infinite set Λ consisting of the corresponding λ for the statements x^*.

Now, since Π is (d, ρ)-strongly FS compatible (Definition), it is also round-by-round sound. This guarantees that there is a function State such that for every $\lambda \in \Lambda$,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}}\left[(\mathsf{State}(x^*_\lambda, \emptyset) = \mathsf{Reject}) \wedge (\mathsf{State}(x^*_\lambda, \pi^*) = \mathsf{Accept})\right] \geq \frac{1}{\mathsf{p}(\lambda)}.$$

Since there are $\ell(\lambda)$ rounds in the protocol, by a hybrid argument $\forall \lambda \in \Lambda$, $\exists i \in [\ell(\lambda)]$ such that,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}}\left[(\mathsf{State}(x^*_\lambda, \mathsf{trans}^*_{i-1}) = \mathsf{Reject}) \wedge (\mathsf{State}(x^*_\lambda, \mathsf{trans}^*_{i-1}) = \mathsf{Accept})\right] \geq \frac{1}{\ell(\lambda) \cdot \mathsf{p}(\lambda)}.$$

where π^* is parsed as $(\alpha^*_1, \beta^*_1, \cdots, \alpha^*_\ell, \beta^*_\ell)$ and $\forall j \in [\ell(\lambda)]$, $\mathsf{trans}^*_k := (\alpha^*_1, \beta^*_1, \cdots, \alpha^*_j, \beta^*_j)$.

The above only happens if $\beta^*_i \in \mathcal{B}$, where

$$\mathcal{B} = \{\beta \mid \mathsf{State}(x^*, \mathsf{trans}^*_{i-1}|\alpha^*_i|\beta) = \mathsf{Accept}\}.$$

We can now rewrite the same probability above in terms of whether the cheating prover outputs β^*_i from the bad set $\mathcal{B}$ defined above,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}}\left[(\mathsf{State}(x^*_\lambda, \mathsf{trans}^*_{i-1}) = \mathsf{Reject}) \wedge (\beta^*_i \in \mathcal{B})\right] \geq \frac{1}{\ell(\lambda) \cdot \mathsf{p}(\lambda)}.$$

To break the correlation intractability of $\mathcal{H}$ for TC^0, we need to construct a PPT adversary $\mathcal{A}_{\mathsf{CI}}$ and a function $f = \{f_\lambda\}$ computable in TC^0 such that for infinitely many λ,

$$\Pr\left[\mathcal{H}.\mathsf{Hash}(k, x) = f_\lambda(x) \;\middle|\; k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda), x \leftarrow \mathcal{A}_{\mathsf{CI}}(1^\lambda, k)\right] \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

Note that BAD is a function that takes in as input $(x^*_\lambda, \mathsf{trans}_{i-1}|\alpha_i)$ and randomness $r \leftarrow\$ \{0,1\}^*$, and outputs an element β such that if $\mathsf{State}(x^*_\lambda, \mathsf{trans}_{i-1}) = \mathsf{Reject}$, then with probability $1 - \mathsf{negl}(\lambda)$, β is uniformly random element in the set $\mathcal{B}\{\beta \mid \mathsf{State}(x^*, \mathsf{trans}_{i-1}|\alpha_i|\beta) = \mathsf{Accept}\}$.

But since BAD is not guaranteed to be in TC^0, so we cannot use it directly to define a function f in TC^0. Instead, we leverage the fact that BAD can be decomposed into a *deterministic* polynomial time function PreComp, and a family of TC^0 circuits $\mathsf{OnlineBAD}_{\lambda,[r]}$ with randomness r is hardwired (see Definition 10 for details) such that $\mathsf{BAD}(x, \mathsf{trans}|\alpha_i; r) = \mathsf{OnlineBAD}_{\lambda,[r]}(\mathsf{PreComp}(1^\lambda, x^*, \mathsf{trans}|\alpha_i))$.[17] This then allows us to define the function f in TC^0 to be the following

[17]This is in fact the reason why in our Fiat-Shamir transformation, parties must pre-compute before applying the hash function.

$$f_{\lambda,r}(\cdot) := \mathsf{OnlineBAD}_{\lambda,[r]}(\cdot),$$

where the randomness r corresponds to the hardwired randomness of OnlineBAD.

By definition of $(d, T_{\mathsf{PreComp}}, \rho)$-strong FS compatibility (Definition 10), and from the fact that PreComp is a *deterministic* function, for every $\mathsf{trans}_{i-1}, \alpha_i$ and $\beta \in \mathcal{B}$,

$$\Pr_{r \leftarrow\$ \{0,1\}^*}\left[f_{\lambda,r}(y) = \beta \;\middle|\; \begin{array}{l} \mathsf{State}(x^*_\lambda, \mathsf{trans}_{i-1}|\alpha_i|\beta) = \mathsf{Accept} \\ y = \mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}_{i-1}|\alpha_i) \end{array} \right] = \frac{1 - \mathsf{negl}(\lambda)}{d(\lambda)}.$$

Combining the equations above, we get the following,

$$\Pr_{\substack{\mathsf{CRS}\leftarrow\mathsf{Gen}(1^\lambda) \\ \pi^*\leftarrow\mathsf{P}^*(\mathsf{CRS}) \\ r \leftarrow\$ \{0,1\}^*}}\left[\begin{array}{ll} & \mathsf{State}(x^*_\lambda, \mathsf{trans}^*_{i-1}) = \mathsf{Reject} \\ \wedge & \beta^*_i = f_{\lambda,r}(\mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}^*_{i-1}|\alpha^*_i)) \end{array} \right] \geq \frac{1 - \mathsf{negl}(\lambda)}{d(\lambda)} \cdot \frac{1}{\ell(\lambda) \cdot \mathsf{p}(\lambda)}.$$

For $d(\lambda)$ and $\ell(\lambda)$ polynomial functions in λ, the above equation implies that there exists an r, and by extension $f := \{f_{\lambda,r}\} \in \mathsf{TC}^0$, such that

$$\Pr_{\substack{\mathsf{CRS}\leftarrow\mathsf{Gen}(1^\lambda) \\ \pi^*\leftarrow\mathsf{P}^*(\mathsf{CRS})}}\left[\begin{array}{ll} & \mathsf{State}(x^*_\lambda, \mathsf{trans}^*_{i-1}) = \mathsf{Reject} \\ \wedge & \beta^*_i = f_{\lambda,r}(\mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}^*_{i-1}|\alpha^*_i)) \end{array} \right] \geq \frac{1}{\mathsf{q}(\lambda)}.$$

This gives us an adversary $\mathcal{A}_{\mathsf{CI}}$, using the cheating prover P^*, that breaks the correlation intractability of $\mathcal{H}$ for $f \in \mathsf{TC}^0$,

$\underline{\mathcal{A}_{\mathsf{CI}}(1^\lambda)}$:

1. Receive k from the CI challenger.
2. Sample $i \in [\ell(\lambda)]$, and set $k_i := k$
3. Sample $\forall j \in [\ell(\lambda)] \setminus \{i\}$, $k_j \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$.
4. Sample crs as specified by Π.
5. Set $\mathsf{CRS} := (\mathsf{crs}, \{k_i\}_{i\in[\ell]})$.
6. Run the cheating prover P^*, $\mathsf{trans}^* \leftarrow \mathsf{P}^*(\mathsf{CRS})$.
7. Output $\mathsf{PreComp}(1^\lambda, x^*, \mathsf{trans}^*_{i-1}|\alpha^*_i)$, where α^*_i is taken from trans^*.

By the above equations, with probability $1/\mathsf{poly}(\lambda)$, $\mathcal{A}_{\mathsf{CI}}$ outputs $\mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}^*_{i-1}|\alpha^*_i)$ such that

$$\beta^*_i := \mathcal{H}.\mathsf{Hash}(k_i, \mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}^*_{i-1}|\alpha^*_i)) = f_{\lambda,r}(\mathsf{PreComp}(1^\lambda, x^*_\lambda, \mathsf{trans}^*_{i-1}|\alpha^*_i)).$$

Thus giving us the desired contradiction to prove soundness.

Efficiency. This follows directly from [JKKZ21], where the extra overhead in the the prover and verifier running time if from evaluating the hash function ℓ times after performing the pre-computation PreComp on the transcript, where the size of the hash function is polynomially related to $\rho(\lambda)$.

□

Remark 3. *We note that if we relax the strong* FS*-compatible protocol requirements to not require the low-depth property from* BAD*, and to simply require* BAD *to be computable in time $\rho(\lambda)$, then it suffices for the* CI *hash family to be correlation intractable against functions computable in time $\rho(\lambda)$. This follows identically as in the proof of Theorem 9*

7.2 Non-interactive batch arguments

Let us denote the interactive protocol from Section 5 by $\Pi_{\mathsf{R1CS},k}$. We apply the Fiat-Shamir transformation with respect to an appropriate hash function family $\mathcal{H}$. The transformation is essentially identical, and is presented in Figure 10 for completeness. But there are a couple of important distinctions from Figure 9 that prevent us from directly invoking Theorem 9. Firstly, note that Theorem 9 refers to a promise language, while we want to apply it to the batching of k instances. Secondly, as the crs is generated by in the *normal mode* by calling Gen - the corresponding interactive protocol no longer satisfies strong FS-compatibility as proven in Section 6.

We now prove that the transformation in Figure 10 is sound, thereby giving us a non-interactive batch argument.

Theorem 10. *Let a $\mathcal{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ correlation intractable hash function family for TC^0 (Definition 7) where inputs of $n(\lambda)$ bits are mapped to λ bits, and $\mathsf{LHC} = (\mathsf{Gen}, \mathsf{ExtGen}, \mathsf{Com}, \mathsf{Ext}, \mathsf{Samp})$ be a somewhere-extractable linearly homomorphic commitment scheme (Section 6). Then the protocol in Figure 10 is a non-interactive batch argument.*

Protocol: Non-interactive batch argument (P,V) for $\mathsf{R1CS}$ instances

Common Reference String: $\mathsf{CRS} := (K, \{k_i\}_{i\in[\ell]})$ consisting of $\mathsf{crs} \leftarrow \Pi_{\mathsf{R1CS},k}.\mathsf{Gen}(1^\lambda, 1^k)$, and ℓ keys $\left\{k_i \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)\right\}_{i\in[\ell]}$

Common input: input $\left\{\mathbb{x}^{(j)}\right\}_{j\in[k]} := (\mathbb{F}, A, B, C, \left\{\mathrm{io}^{(j)}\right\}_{j\in[k]}, m, n)$, security parameter 1^λ.

P**'s auxiliary input:** witnesses $\left\{w^{(j)}\right\}_{j\in[k]}$ such that $\forall j \in [k]$, $\mathsf{Sat}_{\mathsf{R1CS}}\left(\mathbb{x}^{(j)}, w^{(j)}\right) = 1$

1. Prover P_{FS} on input $(\{\mathbb{x}^{(j)}, w^{(j)}\}_{j\in[k]}, \mathsf{CRS})$ sets $i := 1$, $\mathsf{trans} := \emptyset$ and does the following:

   For $i \in [\ell]$

   (a) Compute $\alpha_i \leftarrow \mathsf{P}(K, \{\mathbb{x}^{(j)}, w^{(j)}\}_{j\in[k]}, \mathsf{trans})$ and $\beta_i := \mathcal{H}.\mathsf{Hash}(k_i, \mathsf{PreComp}(1^\lambda, \{\mathbb{x}^{(j)}\}_{j\in[k]}, \mathsf{trans}|\alpha_i))$

   (b) Set $\mathsf{trans} := \mathsf{trans}|\alpha_i|\beta_i$.

   Output trans.

2. Verifier V_{FS} on input $(\left\{\mathbb{x}^{(j)}\right\}_{j\in[k]}, \mathsf{CRS}, \mathsf{trans})$ does the following

   For $i \in [\ell]$

   (a) Check $\beta_i \overset{?}{=} \mathcal{H}.\mathsf{Hash}(k_i, \mathsf{PreComp}(1^\lambda, \{\mathbb{x}^{(j)}\}_{j\in[k]}, \mathsf{trans}|\alpha_i)$ where $\mathsf{trans}_{i-1} := \alpha_1|\beta_1|\cdots|\alpha_{i-1}|\beta_{i-1}$.

   Accept if and only if $\mathsf{V}(\mathsf{crs}, \{\mathbb{x}^{(j)}\}_{j\in[k]}, \mathsf{trans})$ accepts.

Figure 10: Non-Interactive Batch Argument for k instances.

Proof. As noted in the introduction of Section 7.2, the differences in the interactive protocols in this section and the interactive protocol of Section 7.1 for which we apply the Fiat-Shamir transform mean we cannot directly invoke Theorem 9. Instead, we reduce the interactive protocol to one suitable to the application of the aforementioned theorem.

We prove soundness by the way of contradiction. Suppose soundness does not hold, there exists a PPT prover P^*, a polynomial $\mathsf{p}(\cdot)$ and an infinite set of *false* statements

$$X := \left\{ \vec{\mathbb{x}}^* := \left\{ \mathbb{x}^{(j)} \right\}_{j \in [k]} \middle| \exists i \in [k] s.t. \mathbb{x}^{(i)} \notin L_{\mathsf{R1CS}} \right\}$$

such that for each $\vec{\mathbb{x}}^* \in X$, and $\lambda = \lambda(|\vec{\mathbb{x}}^*|)$,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi) = 1] \geq \frac{1}{\mathsf{p}(\lambda)} \tag{10}$$

By assuming without loss of generality that each distinct $\vec{\mathbb{x}}^*$ maps to a different λ, we get the above to hold for an infinite set Λ consisting of the corresponding λ for the statements $\vec{\mathbb{x}}^*$.

Claim 2. *For every $i^* \in [k]$ and every large enough $\lambda \in \Lambda$,*

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}'(1^\lambda, i^*) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi) = 1] \geq \frac{1}{2\mathsf{p}(\lambda)},$$

Proof. Suppose for the sake of contradiction, that there $\exists i^* \in [k]$ and an infinite set $\Lambda_0 \subseteq \Lambda$ such that for all $\lambda \in \Lambda_0$,

$$\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}'(1^\lambda, i^*) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi) = 1] < \frac{1}{2\mathsf{p}(\lambda)}, \tag{11}$$

Then we shall use P^* to break the dual mode indistinguishability of $\Pi_{\mathsf{R1CS},k}$, by constructing an adversary $\mathcal{A}$ that receives crs from the challenger that was either generated as $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$ or $\mathsf{crs} \leftarrow \mathsf{TGen}(1^\lambda, 1^k, i^*)$.

<u>$\mathcal{A}(1^\lambda)$:</u>

1. Receive crs from dual mode challenger.
2. Sample $\forall i \in [\ell(\lambda)]$, $k_i \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$.
3. Sample crs as specified by Π.
4. Set $\mathsf{CRS} := (\mathsf{crs}, \{k_i\}_{i \in [\ell]})$.
5. Run the cheating prover P^*, $\pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})$.
6. Output $\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi)$.

From the description of $\mathcal{A}$, when crs was generated as $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^k)$ (resp. $\mathsf{crs} \leftarrow \mathsf{TGen}(1^\lambda, 1^k, i^*)$), CRS has the distribution corresponding to the output of Gen (resp. Gen′). Therefore,

$$\begin{aligned}
\Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathcal{A} = 1] - \Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}'(1^\lambda, i^*) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathcal{A} = 1]& \\
&= \Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}(1^\lambda) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi) = 1] - \Pr_{\substack{\mathsf{CRS} \leftarrow \mathsf{Gen}'(1^\lambda, i^*) \\ \pi^* \leftarrow \mathsf{P}^*(\mathsf{CRS})}} [\mathsf{V}(\mathsf{CRS}, \vec{\mathbb{x}}^*, \pi) = 1] \\
&\geq \frac{1}{2\mathsf{p}(\lambda)},
\end{aligned}$$

where the last inequality follows from Equations (10) and (11). This breaks the dual mode indistinguishability of $\Pi_{\mathsf{R1CS},k}$, thus arriving at our desired contradiction. □

Let $\Pi_{\mathsf{R1CS},k,i^*}$ to indicate the protocol in the trapdoor mode where the trapdoor is picked for the index $i^* \in [k]$.

Claim 3. *With probability $\geq 1/k$ over the random choice of $i^* \leftarrow_\$ [k]$, $\Pi_{\mathsf{R1CS},k,i^*}$ is a strong* FS*-compatible protocol for $\bar{L}$ in the trapdoor mode for i^*, where $\bar{L} = (L_{\mathsf{YES}}, L_{\mathsf{NO}})$ is defined as follows*

- **Yes Instance:** $\forall j \in [k], \mathbb{x}^{(j)} \in L_{\mathsf{R1CS}}$.
- **No Instance:** $\mathbb{x}^{(i^*)} \notin L_{\mathsf{R1CS}}$.

Proof. There is at least one statement $\mathbb{x}^{(i)} \notin L_{\mathsf{R1CS}}$, therefore with probability $\geq 1/k$ the i^* is picked for the trapdoor mode setup generation corresponds to an instance $\mathbb{x}^{(i^*)}$ such that $\mathbb{x}^{(i^*)} \notin L_{\mathsf{R1CS}}$. The strong FS-compatibility then follows directly from Section 6. □

We can now finally invoke Theorem 9. From the above two claims, P^* with probability $\geq 1/(2 \cdot k \cdot \mathsf{p}(\lambda))$ breaks the soundness of the Fiat-Shamir transform of $\Pi_{\mathsf{R1CS},k,i^*}$ for the promise language $\bar{L}$ (defined above), contradicting the aforementioned Theorem 9. Therefore no such P^* can exist, completing the proof.

□

Finally, as in the interactive case, we get the following corollary with costs corresponding to the size the Boolean circuit when we start with an instance of Boolean circuit satisfiability.

Corollary 2. *If we start with* C-SAT *instances defined by a boolean circuit $C : \{0,1\}^{|x|} \times \{0,1\}^{|y|} \mapsto \{0,1\}$, then the protocol in Figure 10 is a non-interactive batch argument for* C-SAT *where*

- *Total communication cost is $O(|C| + k \log |C|)\mathsf{poly}(\lambda)$.*
- *The verifier's total run time is $O(k|x| + |C|) \cdot \mathsf{poly}(\lambda) + \mathsf{p}(\rho(\lambda)) \log |C|$ where* p *is a polynomial that depends only on the* CIH.

We note that ρ must be at least as large as BAD, which in turn has size at least $|C|$. Since the additive term depends on the specific instantiation of the CIH, we choose to leave the verification time in the form above.

7.3 Communication with sub-linear dependence on k

We have so far constructed non-interactive batch arguments where the communication complexity grows linearly with k (see Theorem 10 and Corollary 2). We now describe how to get this dependence in communication complexity to be sub-linear in k without any changes to our protocol. Note that this consideration is meaningful only in the setting where $k \gg |C|$, as otherwise the additive term in k is dominated by the $|C|$.

The high level idea is simple: group instances into a single larger instance. Let us elaborate, we group k_1 (of the k) C-SAT instances to form a larger circuit $C' : \{0,1\}^{k_1|x|} \times \{0,1\}^{k_1|y|} \mapsto \{0,1\}$ such that

$$C'((x_1, \cdots, x_k), (y_1, \cdots, y_k)) = \begin{cases} 1 & \text{if } C(x_1, w_1) = 1 \wedge \cdots \wedge C(x_{k_1}, w_{k_1}) = 1 \\ 0 & \text{o.w.} \end{cases}$$

where C is the circuit for the underlying C-SAT instances. The size of this circuit C' from the above description is thus $|C'| = k_1|C| + O(k_1)$.

Starting with k instances of C-SAT defined over the circuit C, we now have k/k_1 instances of C-SAT defined over the circuit C'[18]. Importantly, note that the circuit across all of the k/k_1 instances are the same,

[18] Assume for simplicity that k_1 divides k exactly.

allowing us to apply our non-interactive argument protocol giving us, by Corollary 2, total communication cost of

$$O(|C'| + (k/k_1)\log|C'|)\cdot\mathsf{poly}(\lambda) = O(k_1|C| + (k/k_1)\log(k_1|C|))\cdot\mathsf{poly}(\lambda)$$

By setting $k_1 = \left\lceil \sqrt{\frac{k\log(k|C|)}{|C|}} \right\rceil$, we get total communication cost to be

$$O\left(\sqrt{k|C|\log(k|C|)} + \frac{\sqrt{k|C|\log(k|C|)}}{|C|}\log(\sqrt{k|C|\log(k|C|)})\right)\cdot\mathsf{poly}(\lambda)$$
$$= \widetilde{O}(\sqrt{k|C|})$$

which has only sub-linear dependence on both k and $|C|$, where $\widetilde{O}$ hides terms that are polynomial in λ and poly-logarithmic in k and $|C|$. If we simply wanted sub-linear dependence on k, we get a simpler formulation if we set $k_1 = \sqrt{k}$ where the resultant communication cost is $O(\sqrt{k}|C| + \sqrt{k}\log(\sqrt{k}|C|))\cdot\mathsf{poly}(\lambda)$.

7.4 Barriers to Achieving Adaptive Soundness

In this section we describe the main ideas that demonstrate that achieving adaptive security in our setting poses significant challenges. This was already established to be the case for *privately verifiable* non-interactive batch arguments in [BHK17]. There are a couple of points of differences that prevent us directly applying the barriers in their paper: (i) the BARG in their paper is also a batch argument of knowledge (BARK); and (ii) the BARK in their work has communication that is independent of k, while the communication in our work depends sub-linearly on k.

We start by describing why our BARG is also an argument of knowledge, i.e. BARK. And then we show that the sub-linear dependence on k is good enough to demonstrate a barrier. We have kept the discussion in this section to be largely informal, and we refer the reader to [BHK17] for more details on defining BARKs, and the adaptivity barrier.

Achieving (non-adaptive) proof of knowledge. Roughly, the proof of knowledge property states that if a prover can convince the verifier to accept, then it must be the case that the prover has knowledge of the witness. This is formalized by an extraction algorithm that extracts the witness from the prover given only the statements and the prover's code. Typically one requires that the running time of the extraction algorithm is related to the probability with which the prover causes the verifier to accept. In our proofs, we've already seen how to extract a single witness from a batch of k witnesses - by specifying the index of the witness we want to extract during the CRS generation. To extract all k witnesses, we follow the same strategy by picking a different index for the CRS each time. The dual-mode indistinguishability ensures that when the switch is made, the probability that the prover convinces the verifier does not drop by much, thus allowing for extraction of all k witnesses.

Adapting the [BHK17] barrier. At a high level, the authors in [BHK17] show that if one could construct adaptively secure BARKs, then in combination with a RAM delegation scheme one could construct SNARKs, thereby running into the lower bounds of Gentry-Wichs [GW11]. This would imply that non-adaptive BARKs are the best that one could hope for. As noted, unlike the [BHK17], whose communication complexity is independent of k, our protocol has a sub-linear dependence on k.

The transformation to SNARKs follows the strategy: (1) start with an NP language L with statement x, compute a Merkle digest of the witness w corresponding to x. Let $|w| = n$, given the digest d, for each $i \in [n]$, there is $O(\log n)$ "proof" that w_i corresponds to the i-th position that resulted in digested d. This can be viewed as n statements each with a witness of length $\widetilde{O}(\log n)$ and relation circuit C of size

$\widetilde{O}(\log n)$. By Section 7.3, we have a batch proof of size $\widetilde{O}(\sqrt{n \log n})$ to prove that the digest d was computed honestly. By the above observation, this is also an argument of knowledge. We will refer to such a proof as *slightly succinct* (In [BHK17] the batch proofs is of size $\widetilde{O}(\log n)$, and thus fully succinct). (2) Use the RAM delegation protocol to prove that the NP relation for L accepts (x, w) where d corresponds to the digest of w. The adaptivity of the batch arguments of knowledge is crucial since the prover get to chooses d, the statement for the batch argument. We refer the reader to their paper for the details. While their SNARK achieves *full* succinctness, the SNARK achieved above is only slightly succinct. But even slightly succinct is good enough to demonstrate a barrier as this contradicts the [GW11] lower bound for black-box security reduction to falsifiable assumptions.

Acknowledgments

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

References

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001. 7

[BBH+19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 522–551. Springer, Heidelberg, December 2019. 6

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. 3

[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. 3

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013. 54

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2010. 14, 19

[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. 3, 4, 48, 49

[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. 5, 6, 11

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Heidelberg, April / May 2018. 14

[BRS67] E.R. Berlekamp, H. Rumsey, and G. Solomon. On the solution of algebraic equations over finite fields. *Information and Control*, 10(6):553–564, 1967. 39

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 4, 5, 6, 31

[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. 5

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 4, 5, 19

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995. 3

[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. 5

[CHP12] Jan Camenisch, Susan Hohenberger, and Michael Østergaard Pedersen. Batch verification of short signatures. *Journal of Cryptology*, 25(4):723–747, October 2012. 4

[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. 5

[CPV20] Michele Ciampi, Roberto Parisella, and Daniele Venturi. On adaptive security of delayed-input sigma protocols and fiat-shamir NIZKs. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 670–690. Springer, Heidelberg, September 2020. 5

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 54–74. Springer, Heidelberg, March 2012. 3

[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. 14

[DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 3–31. Springer, Heidelberg, March 2018. 14

[DGI$^+$19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. 6, 14

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 5

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998. 4

[GH18] Sanjam Garg and Mohammad Hajiabadi. Trapdoor functions from the computational Diffie-Hellman assumption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 362–391. Springer, Heidelberg, August 2018. 14

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003. 7

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-Proofs. Cryptology ePrint Archive, Report 2011/456, 2011. http://eprint.iacr.org/2011/456. 3

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Comput. Complex.*, 11(1-2):1–53, 2002. 4

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 3, 4, 48, 49

[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. 25

[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. 5

[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. 39, 40

[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. 10, 14

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-Interactive Zero Knowledge from Sub-exponential DDH. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2021. 4, 5, 6, 11, 19

[JK20] Ruta Jawale and Dakshita Khurana. Lossy correlation intractability and PPAD hardness from sub-exponential LWE. Cryptology ePrint Archive, Report 2020/911, 2020. https://eprint.iacr.org/2020/911. 5

[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. 5, 10, 11, 12, 13, 30, 42, 44

[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. 4, 6

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015. 14

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997. 3

[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. 3, 4

[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. 5

[KRR$^+$20] Inbar Kaslasi, Guy N. Rothblum, Ron D. Rothblum, Adam Sealfon, and Prashant Nalini Vasudevan. Batch verification for statistical zero knowledge proofs. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 139–167. Springer, Heidelberg, November 2020. 4

[KRV21] Inbar Kaslasi, Ron D. Rothblum, and Prashant Nalini Vasudevan. Public-Coin Statistical Zero-Knowledge Batch Verification against Malicious Verifiers. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2021. 4

[KZ20] Yael Tauman Kalai and Rachel Zhang. SNARGs for bounded depth computations from sub-exponential LWE. Cryptology ePrint Archive, Report 2020/860, 2020. https://eprint.iacr.org/2020/860. 5

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. 8, 17

[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. 5

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 3

[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. 3

[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. 14

[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. 4, 5, 6, 11, 19

[RR20] Guy N. Rothblum and Ron D. Rothblum. Batch verification and proofs of proximity with polylog overhead. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 108–138. Springer, Heidelberg, November 2020. 3, 4, 7

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016. 3, 4, 7, 8

[RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In *Computational Complexity Conference*, volume 102 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 3, 4, 7

[RT92] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM Journal on Computing*, 21(5):896–908, 1992. 25

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020. 7, 9, 11, 12, 16, 55, 56

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. 8, 17

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 71–89. Springer, Heidelberg, August 2013. 54, 57

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237. IEEE Computer Society Press, May 2013. 54, 57

A Additional Preliminaries

Let $\mathbb{Z}$ be the set of all integers. For any $n \in \mathbb{Z}$, denote $[n] = \{1, 2, \ldots, n\}$. For any positive integer p, let $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ be the quotient ring of $\mathbb{Z}$ divided by the ideal $p\mathbb{Z}$.

Threshold Gate. Let $x_1, x_2, \ldots, x_n$ be n binary variables. A threshold gate is defined as the following function:

$$\mathsf{Th}_t(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \sum_{i\in[n]} x_i \geq t \\ 0 & \text{Otherwise} \end{cases}$$

Threshold Circuits and TC^0. A threshold circuit is a directed acyclic graph, where each node either computes a threshold gate of unbounded fan-in or a negation gate. We use TC^0 to denote the class of constant depth polynomial-size threshold circuits.

Theorem 11 (Schwartz-Zippel Lemma). *Let $\mathbb{F}$ be a field, and let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of total degree $d \geq 0$, let $S \subseteq \mathbb{F}$ be a finite subset of $\mathbb{F}$, then* $\Pr[x \leftarrow S^n : f(x) = 0] \leq d/|S|$.

A.1 Low-Degree Extension

Throughout this work, we find it convenient to work with polynomials rather than functions. Specifically, given any function $g : \{0,1\}^m \mapsto \mathbb{F}$, we consider a multivariate polynomial $\widetilde{g} : \mathbb{F}^m \mapsto \mathbb{F}$ such that $\widetilde{g}$ is an extension of g, i.e. $\forall x \in \{0,1\}^m$, $\widetilde{g}(x) = g(x)$ (agree on the Boolean hybercube). We say that $\widetilde{g}$ is a *low-degree extension* if the individual degree over each variable is exponentially smaller than $|\mathbb{F}|$.

We shall consider the special case of a low-degree extension, the *multlinear extension (MLE)*, where $\widetilde{g}$ is multivariate polynomial with degree at most 1 in each variable. In fact, given any function $Z : \{0,1\}^m \mapsto \mathbb{F}$, its MLE is unique and is given by,

$$\begin{aligned}\widetilde{Z}(x_1, \cdots, x_m) &= \sum_{e\in\{0,1\}^m} Z(e) \cdot \prod_{i=1}^{m} (x_i \cdot e_i + (1 - x_i)(1 - e_i)) \\ &= \sum_{e\in\{0,1\}^m} Z(e) \cdot \widetilde{\mathrm{eq}}(x, e)\end{aligned}$$

where $\widetilde{\mathrm{eq}}$ is the MLE of the equality function $\mathrm{eq}(x, y)$ that returns 1 if and only if $x = y$.

Representation for multilinear polynomials. Any multilinear polynomial $f(\cdot) : \mathbb{F}^m \mapsto \mathbb{F}$ can be uniquely represented by the list of evaluations of f over the Boolean hypercube $\{0,1\}^m$. This is referred to as the *dense* representation of the polynomial, and the size of this representation is $O(2^m)$.

Given an input $x \in \mathbb{F}^m$, and a dense representation of the polynomial f, from our above discussion it is easy to see that $f(x)$ is a linear combination of the dense representation where the linear coefficient for the e-th term for $e \in \{0,1\}^m$ is computed as $\widetilde{\mathrm{eq}}(x, e)$.

For any $r \in \mathbb{F}^m$, the evaluation of the multilinear polynomial f can be computed in $O(2^m)$ operations in $\mathbb{F}$ [Tha13, VSBW13].

A.2 C-SAT to R1CS

For completeness we describe the transformation from [BCG+13] below to be consistent with the notation used in the paper.

For the transformation we will find it convenient to view R1CS instances as a sequence of m constraints

$$\{\langle a, z\rangle \cdot \langle b, z\rangle = \langle c, z\rangle\}_{i\in[m]}$$

where z is of the form $(\text{io}, 1, w)$ for some witness w, and $a, b, c \in \mathbb{F}^m$ correspond to the rows of matrices A, B and C respectively.

We define a variable $X \in \{0, 1\}^{|C|}$ that will correspond to the value of each wire in the circuit, including the $|x|+|y|$ input wires corresponding to the statement and witness. Further, define a variable $Z := (1, X) \in \{0, 1\}^{|C|+1}$.

Gates For each gate $X_k = \text{NAND}(X_i, X_j)$, we rewrite it as $(1-X_i)(1-X_j) = 1-X_k$ and add the constraint

$$\langle(1, -e_i), z\rangle \cdot \langle(1, -e_j), z\rangle = \langle(1, -e_k), z\rangle$$

where $e_i \in \{0, 1\}^{|C|}$ is 1 in the i-th position and 0 elsewhere.

It is easy to see that each row has only $O(1)$ non-zero elements, and there are $O(|C| - |x| - |y|)$ gate constraints.

Boolean checks For each input wire X_i, we check if it is a boolean value $X_i^2 = X_i$, or alternatively $X_i(1-X_i) = 0$ and add constraint

$$\langle(0, e_i), z\rangle \cdot \langle(1, -e_i), z\rangle = \langle 0, z\rangle$$

Again, it is easy to see that each row has only $O(1)$ non-zero elements, and there are $O(|x| + |y|)$ gate constraints.

To set the above to be a square matrix, we can pad an extra row of 0s, unaffecting the constraints.

A.3 R1CS Polynomials

We start by giving a sketch of the polynomials required for the construction of $\mathcal{G}$ in Theorem 2. We refer the reader to [Set20] for a full proof and detailed discussion. Recall that an R1CS instance is specified by the tuple $\mathbb{x} = (\mathbb{F}, A, B, C, \text{io}, m, n)$.

New Notation. To start with, we consider a more fine grained notation of R1CS instances than considered in [Set20], where the matrices A, B and C are split into two depending on whether they correspond to the coefficients of the public component io or the private component w. Specifically, $A = (A', A'')$ where $A' \in \mathbb{F}^{m\times(|\text{io}|+1)}$ and $A'' \in \mathbb{F}^{m\times(m-|\text{io}|-1)}$. Similarly for B and C. For simplicity assume that $|\text{io}| + 1 = 2^{s_1}$ and $|w| = 2^{s_2}$, i.e. $m = 2^{s_1} + 2^{s_2}$.

Remark 4. *When we construct an* R1CS *instance from* C-SAT *we can further claim that A' contains only $O(|io|) = O(|x|)$ non-zero entries. We can assume without loss of generality that each input wire in C goes into a single gate. Therefore, the there are $O(|x|)$ non-zero entries corresponding to Boolean constraints and $O(|x|)$ non-zero entries corresponding to gate constraints.*

Given the notation discussed, one can rewrite the R1CS constraints as a sequence of 2^s constraints, where $s = \lceil \log m \rceil$. We implicitly assume that the matrix functions $A', A'', B', B'', C', C''$ return the value 0 when input string indexes to a position out of bounds (i.e. larger than m). It is easy to see that $s = \max\{s_1, s_2\} + 1$. For all $x \in \{0, 1\}^s$,

$$F_{\text{io}}(x) := \left(\sum_{y\in\{0,1\}^{s_1}} A'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} A''(x,y)\cdot w_y\right)\cdot$$
$$\left(\sum_{y\in\{0,1\}^{s_1}} B'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} B''(x,y)\cdot w_y\right)$$
$$-\left(\sum_{y\in\{0,1\}^{s_1}} C'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} C''(x,y)\cdot w_y\right)$$

Replacing each of the functions $A', A'', B', B'', C', C''$ by their multilinear extension, one arrives at the polynomial $\widetilde{F}_{\text{io}} : \mathbb{F}^s \mapsto \mathbb{F}$,

$$\widetilde{F}_{\text{io}}(x) := \left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{A}'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{A}''(x,y)\cdot w_y\right)\cdot$$
$$\left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{B}'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{B}''(x,y)\cdot w_y\right)$$
$$-\left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{C}'(x,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{C}''(x,y)\cdot w_y\right)$$

The following lemma pertaining to $\widetilde{F}$ is proved in [Set20].

Lemma 11 ([Set20]). *$\forall x \in \{0,1\}^s, \widetilde{F}_{io}(x) = 0$ if and only if $\mathcal{R}_{\text{R1CS}}(\mathbb{x}, w) = 1$.*

This lets us define the zero-test polynomial $Q_{\text{io}}(t)$ for $t \in \mathbb{F}^s$

$$Q_{\text{io}}(t) := \sum_{x\in\{0,1\}^s} \widetilde{F}_{\text{io}}(x)\cdot\widetilde{\text{eq}}(t,x)$$

The following lemma states that if $\widetilde{F}_{\text{io}}$ is not 0 at *all* points on the Boolean hypercube, then Q_{io} evaluated at random point $\tau \in \mathbb{F}^s$ is 0 with negligible probability over the randomness in sampling τ.

Lemma 12 ([Set20]).

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

By setting the parameters as described in the theorem, we derive that the above probability is negligible in λ. Then $\mathcal{G}_{\text{io},\tau} := \widetilde{F}_{\text{io}}(x)\cdot\widetilde{\text{eq}}(\tau,x)$ satisfies Theorem 2.

We will also find it useful to compute the evaluation of $\widetilde{F}$ at a point $r^* \in \mathbb{F}^s$,

$$\widetilde{F}_{\text{io}}(r^*) := \left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{A}'(r^*,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{A}''(r^*,y)\cdot w_y\right)\cdot$$
$$\left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{B}'(r^*,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{B}''(r^*,y)\cdot w_y\right)$$
$$-\left(\sum_{y\in\{0,1\}^{s_1}} \widetilde{C}'(r^*,y)\cdot(\text{io},1)_y + \sum_{y\in\{0,1\}^{s_2}} \widetilde{C}''(r^*,y)\cdot w_y\right)$$

This lets us define values $\overline{A}(r^*), \overline{B}(r^*), \overline{C}(r^*)$

$$\begin{aligned}\overline{A}(r^*) &= \sum_{y \in \{0,1\}^{s_1}} \widetilde{A}'(r^*, y) \cdot (\mathrm{io}, 1)_y + \sum_{y \in \{0,1\}^{s_2}} \widetilde{A}''(r^*, y) \cdot w_y \\ &= v_{A,1} + v_{A,2}\end{aligned}$$

with the corresponding terms for $\overline{B}(r^*)$ and $\overline{C}(r^*)$. Lastly, we calculate the time taken to compute $v_{A,1}$, $v_{A,2}$ and their corresponding counterparts for B and C.

Time to Compute $v_{A,1}$: To compute $\widetilde{A}'(r^*, y)$ for all $y \in \{0,1\}^{s_1}$ the *total time* taken is $O(n+m)$. This follows from the definition of $\widetilde{A}'(r^*, y)$

$$\begin{aligned}\widetilde{A}'(r^*, y) &= \sum_{e \in \{0,1\}^s} A'(e, y)\widetilde{\mathrm{eq}}(e, r^*) \\ &= \langle\ (A'(0, y), \cdots, A'(m, y))\ ,\ (\widetilde{\mathrm{eq}}(0, r^*), \cdots, \widetilde{\mathrm{eq}}(m, r^*)))\ \rangle\end{aligned}$$

From [Tha13, VSBW13], a table containing the values $(\widetilde{\mathrm{eq}}(0, r^*), \cdots, \widetilde{\mathrm{eq}}(m, r^*))$ can be computed using $O(m)$ space and $O(m)$ time (ignoring polynomial factors in $\log|\mathbb{F}|$). With the table computed, it takes only $O(m)$ time to compute $\widetilde{A}'(r^*, y)$. Since A' has at most $O(n)$ non-zero values, the time to compute $\widetilde{A}'(r^*, y)$ for all $y \in \{0,1\}^{s_1}$ is $O(n+m)$. Finally, the time to compute $v_{A,1}$ is $O(|\mathrm{io}| + n + m)$.

For k computations with the same $A'(r^*, \cdot)$ the total time to compute the $v_{A,1}$ values is $O(k \cdot |\mathrm{io}| + n + m)$.

Time to Compute $v_{A,2}$: Similar idea as above, with the total time $O(|w| + n + m)$ for a single instance and $O(k|w| + n + m)$ for k instances with the same $A''(r^*, \cdot)$.