Honest Majority Multi-Prover Interactive Arguments

Alexander R. Block* Christina Garman†

October 7, 2022

Abstract

Interactive arguments, and their (succinct) non-interactive and zero-knowledge counterparts, continue to enjoy advancements in both theory and practice, culminating with growing deployment for real world systems in recent years. Unfortunately concrete proof generation costs can still be quite expensive for large and complex statements. While recent work has sought to solve this problem by outsourcing proof computation to a group of workers in a privacy preserving manner, current solutions still require each worker to do work on roughly the same order as a single-prover solution.

We introduce the *Honest Majority Multi-Prover* (HMMP) model for interactive arguments. In these arguments, we distribute prover computation among M collaborating, but distrusting, provers. All provers receive the same inputs and have no private inputs, and we allow any $t < M/2$ provers to be statically corrupted before generation of public parameters, and all communication is done via an authenticated broadcast channel. In contrast with the recent works of Ozdemir and Boneh (USENIX '22) and Dayama et al. (PETS '22), we target prover *efficiency* over privacy.

We show that: (1) any interactive argument where the prover computation is suitably divisible into M sub-computations can be transformed into an interactive argument in the HMMP model; and (2) arguments that are obtained via compiling polynomial interactive oracle proofs with polynomial commitment schemes admit HMMP model constructions that experience a (roughly) $1/M$ speedup over a single-prover solution. The transformation of (1) preserves computational (knowledge) soundness, zero-knowledge, and can be made non-interactive via the Fiat-Shamir transformation. The constructions of (2) showcase that there are efficiency gains in proof distribution when privacy is not a concern.

1 Introduction

Interactive arguments [BCC88, GMR89] (or *arguments* in short) are interactive protocols that allow a computationally bounded (i.e., polynomial time) prover to certify an NP statement x with witness w to a computationally weak (i.e., sub-linear time) verifier. The *(perfect) correctness* condition states that an honest prover can always convince a verifier of a true statement, while the *soundness* condition states that any polynomial-time malicious prover cannot convince a verifier of a false statement (except with small probability). Arguments and their (succinct) non-interactive counterparts [Mic00, Val08, GW11, SG11, DFH12, BCCT12, GGPR13, BCCT13, BCI+13] remain relevant to both researchers and practitioners for their continued adoption in a variety of technologies, including privacy preserving smart contracts [KMS+16], e-cash [BSCG+14], decentralized private computation [BCG+20], and accountable anonymity [GGM16]. This continued study has advanced the state-of-the-art towards (nearly) optimal prover and verifier efficiency: for a witness of size N and security parameter λ, there are not arguments with $O_\lambda(N)$[1] prover time [GLS+21, XZS22], arguments with verifier time $O_\lambda(\log(N))$ [BCC+16, BBB+18, Lee21] or even $O_\lambda(1)$ [KZG10, BCHO22] under a variety of cryptographic assumptions. Additionally, these arguments achieve other desirable properties such as zero knowledge [GMR89] and various notions of knowledge-soundness [GI08, Lin03, GW11, WTS+18].

*University of Maryland, College Park and Georgetown University. alexander.r.block@gmail.com

†Purdue University. clg@purdue.edu

[1] Subscript λ omits poly(λ) factors.

Though the asymptotic costs of proof generation are nearly optimal, in practice the concrete costs become prohibitive for all but the most powerful machines when statement and witness sizes become large (e.g., $N \geqslant 2^{20}$). One simple solution for this problem is to outsource proof generation to more powerful machines: give the worker machine the statement x and witness w and have it generate the proof. However, many applications wish to protect the privacy of the delegator; i.e., sharing w in the clear is undesirable. While it is impossible to avoid sending the true witness w with a single worker machine, recent work [OB22, DPP+22] overcomes this problem by constructing privacy preserving delegation schemes using multiple worker machines to collaboratively build a proof. Both these schemes can handle the scenario where there is an honest majority of workers, or even the case of a single honest worker by leveraging certain secure multi-party computation (MPC) protocols [Yao82, GMW87, Kil91, IKOS07].

The main drawback of this approach is that each worker runs in time proportional to a single-prover solution, with additional overheads to run specialized MPC protocols tailored for their applications.[2] In essence, these delegation schemes share the witness w via a secret sharing scheme, and each worker produces a proof for their secret share. The key property of these schemes is that the proof produced via operating on a secret share of w is itself a secret share of a valid proof using witness w. Thus, the delegator can recover a complete proof by recombining the worker proofs, and the cost of computing a secret shared proof is identical to the cost of a single prover producing a proof for w.

While preserving privacy is generally desirable, in some applications it is not necessary. For example, zero knowledge rollups (zk-rollups) are an increasingly popular method for increasing the scalability of blockchains. In a zk-rollup, a node bundles together a number of (publicly known) transactions off-chain and then generates a zero-knowledge proof of validity. One significant challenge, though, is that these proofs are expensive for a single node to generate, which reduces the throughput gains that they are meant to achieve. Note that in this scenario, all proof inputs are publicly known to all nodes participating in the system [Eth, LNS20].

Of course, multiple provers are unnecessary if no privacy is required (i.e., just send w to one worker machine). Moreover, under the assumption that the majority of provers are honest, another strategy is to simply have each prover submit one bit certifying the validity of the statement x, with the delegator taking the majority and submitting this as a proof to the verifier. However, all provers in these solutions again run in roughly the same time as a single-prover solution. This seems wasteful:

if privacy is not a concern, is there a way for multiple distrusting provers to jointly compute a proof for a statement such that each participant performs <u>less</u> work than a single-prover solution?

One can also view this problem in the lens of fault-tolerant computing: given multiple faulty cores on a machine, can we distribute proof computation to these core in such a way that correctness is preserved and each core performs less work than a single-core solution?

1.1 Our Contribution

With the goal of leveraging collaboration to gain efficiency, we introduce the *Honest Majority Multi-Prover (HMMP) model* for interactive arguments. A HMMP model argument is an interactive argument where M mutually distrusting provers collaborate to convince a verifier of the validity of an NP statement x such that each prover is given the same witness w and has no other private inputs. Furthermore, up to $t < M/2$ provers can be statically corrupted before the generation of public parameters (or the first message of the verifier), and all provers communicate with each other and the verifier via an authenticated sender broadcast channel [PSL80, LSP82]. We require correctness to hold with high probability in the presence of $t < M/2$ static corruptions, and require soundness to hold when all M provers are corrupt.

We also define zero knowledge in the HMMP model analogously to the single-prover definition of zero knowledge. Informally, we say a HMMP model interactive argument is zero knowledge if a malicious verifier learns nothing more than the validity of the statement x when interacting with M honest prover algorithms. Note this definition differs from the various zero knowledge definitions of [OB22, DPP+22], but we see it as

[2] It is worth noting that the specialized MPC protocols presented in [OB22, DPP+22] are much more efficient than general purpose MPC tools, but still introduce overheads that we would like to avoid.

natural in our setting since all provers share the same witness and have no private inputs. See Section 2.1 for more discussion.

Our main result is showing that *any* argument can be transformed into an argument in the HMMP model. Our transformation perfectly preserves soundness and zero knowledge; in particular, it simply reduces to the soundness and zero knowledge of the original argument. Moreover, if the single-prover argument is publicly verifiable or public coin,[3] then our transformed argument preserves these properties. However, we experience a (small) multiplicative increase in the correctness error with respect to the original argument. For example, if the single-prover argument has perfect correctness, then our HMMP model argument has correctness with a small probability of error.

Our transformation leverages a simple "shuffle" and "audit" strategy, where all provers are shuffled during the generation of public parameters via a random permutation, and a committee of τ provers jointly check consistency of the next message. We stress that our transformation is *completely insecure* (i.e., not sound) if provers can be adatively corrupt, or statically corrupt after the generation of public parameters (or the first message of the verifier). See Section 2.1 for details. We capture the key properties of our HMMP transformation in the following theorem.

Theorem 1.1. *Let $M, \tau, t \in \mathbb{N}$ be parameters such that $t < M/2$ and $\tau \leqslant t$, and let Π_{arg} be an interactive argument. There exists a HMMP model argument Π_{hmp} with M provers and the following parameters:*

- *if Π_{arg} is publicly verifiable (resp., public coin), then Π_{hmp} is publicly verifiable (resp., public coin);*
- *if Π_{arg} has correctness error δ_{arg}, then Π_{hmp} has correctness error $\delta_{\mathrm{hmp}} = 1-(1-\delta_{\mathrm{arg}})\cdot(1-M\cdot(t/M)^{\tau})$ when at most t provers are corrupt;*
- *if Π_{arg} has $\varepsilon_{\mathrm{arg}}$ (computational) soundness, then Π_{hmp} has $\varepsilon_{\mathrm{hmp}} = \varepsilon_{\mathrm{arg}}$ (computational) soundness; and*
- *if Π_{arg} has γ_{arg} zero-knowledge error, then Π_{hmp} has zero-knowledge error $\gamma_{\mathrm{hmp}} = \gamma_{\mathrm{arg}}$.*

Furthermore, if Π_{arg} can be securely made non-interactive via the Fiat-Shamir transformation, then Π_{hmp} can be made non-interactive via the Fiat-Shamir transformation [FS87].

Theorem 1.1 applies to *any* interactive argument Π_{arg}, or even when Π_{arg} is an interactive *proof* (i.e., the prover is computationally unbounded). We achieve this with the following observation. Let P_{arg} be the interactive prover algorithm of Π_{arg}. Then we can naturally represent P_{arg} as a sequence of circuits $(C_0, C_1, \ldots, C_r)$, where each C_i represents the computation P_{arg} computes during round i of Π_{arg} (i.e., C_i is the circuit representing the next message function of P_{arg} during round i). For simplicity, assume that $|C_i|$ is divisible by M. Then our transformation can be viewed as the following two step process: (1) Each circuit C_i is partitioned into chunks $C_{i,j}$ each of size $|C_i|/M$ for $j \in \{1, \ldots, M\}$ such that the input gates of $C_{i,j}$ only depend on the output gates of $C_{i,k}$ for $k < j$ and the input wires of C_i. (2) During each round i of the argument, the j-th committee of τ provers (shuffled by permutation π) compute the circuit $C_{i,j}$ and broadcast the output wire values to the remaining provers. We again direct the reader to Section 2.1 for more details.

1.1.1 Efficiency of Theorem 1.1

As just discussed, our HMMP model transformation applies to any interactive argument Π_{arg} by viewing the prover P_{arg} of that argument as a sequence of circuits that represent the next-message function of P_{arg} and having committees of τ provers jointly compute partitions of these circuits. Of course, assuming appropriate bounds on the sizes of these circuits we can derive efficiency guarantees of the transformation given by Theorem 1.1. While we go over this derivation in Section 2.1, we remark that in practice this may be inefficient. Typically, one would represent the prover algorithm P_{arg} as a random access memory (RAM) machine running in some time T, and so representing P_{arg} as a sequence of circuits incurs a $\mathrm{polylog}(T)$

[3] *Publicly verifiable* means any party with the transcript of the interaction (i.e., the proof) can verify its validity, and *public coin* means all messages sent by a verifier are uniformly and independently chosen at random.

overhead; i.e., $\sum_i |C_i| = O(T \cdot \mathrm{polylog}(T))$ [BCGT13]. This is a major hurdle towards our efficiency goals: we would like our transformation to be as efficient as possible.

We overcome this limitation by directly working with a specific class of interactive arguments that admit efficient sub-division of prover computations. In particular, we consider the class of arguments that are obtained from compiling *polynomial interactive oracle proofs* (PIOPs) [RRR16, BSCS16, BFS20] with *polynomial commitment schemes* [KZG10], which has become one common paradigm for designing efficient arguments [MBKM19, GWC19, CHM+20, BFS20, BHR+20, BHR+21, BCHO22, GLS+21, XZS22]. Briefly, a polynomial interactive oracle proof is an interactive proof where large prover messages are encoded as polynomials and are sent as an oracle for the verifier to query, and all other communication between the prover and verifier is efficient. A polynomial commitment scheme is a special commitment scheme that allows a committer to commit to some polynomial (of bounded degree) to a receiver, and allows the committer to provably open (via a (non-)interactive argument) to evaluations of the polynomial at a specific point specified by the receiver. The essential feature of a polynomial commitment scheme is that this "proof of evaluation" allows the committer to open to an evaluation point without opening the entire polynomial.

Many arguments obtained via this paradigm use the following methodology: (1) a prover uses the polynomial commitment scheme to commit to the polynomial oracles for the verifier to query; (2) the efficient communication between the prover and the verifier is a sum-check protocol [LFKN92] over a polynomial that depends on the committed polynomial; and (3) any query made by the verifier to its oracle(s) is completed using the evaluation proof of the polynomial commitment scheme.

We give an overview of this paradigm in Section 2.2 and discuss how to obtain more efficient HMMP model provers by sub-division of computation, and capture these efficiencies in the following theorem.

Theorem 1.2. *Let $M, \tau, t, \lambda \in \mathbb{N}$ be parameters such that $t < M/2$ and $\tau \leqslant t$. For every* NP *relation with statements of size* $\mathrm{poly}(\lambda)$ *and witnesses of size N, assuming the hardness of the discrete log problem for prime order groups there exists a HMMP model interactive argument Π_{hmp} with M provers and the following efficiency properties:*

- *the individual prover computation is $\widetilde{O}_\lambda(\tau \cdot (N/M))$;*
- *the communication complexity between provers is $\widetilde{O}_\lambda(\tau \cdot (N/M))$ bits;*
- *the verifier computation is $O_\lambda(M + N)$; and*
- *the verifier receives $O_\lambda(M \cdot \log(N))$ bits from all provers and sends $O_\lambda(\log(N))$ bits to all provers.*

We obtain Theorem 1.2 by first giving a HMMP model interactive oracle proof for arithmetic circuit satisfiability (which includes the sum-check protocol), followed by adapting the well-known Bulletproofs polynomial commitment scheme [BCC+16, BBB+18] to the HMMP model. Then by combining these two HMMP model protocols together, we obtain Theorem 1.2.

1.1.2 Defining the Adversarial Model

Our efficiency guarantees in Theorem 1.2 only hold with respect to t corrupt parties under the assumption that their goal is not to make the honest parties perform more computation. In particular, we assume that the goal of the corrupt parties is to try and force the verifier to reject even if the NP statement is true.

Corrupt parties could have the alternative goal of forcing honest parties to perform more computations. In this case, there is a simple attack: in each round of computation, one dishonest party intentionally submits an incorrect value to force an audit. This can happen $t < M/2$ times, which in the worst case (i.e., $t = \Theta(M)$) forces each honest prover to perform $\widetilde{O}_\lambda(\tau \cdot N)$ computation, which is proportional to a single prover solution.

This attack is unavoidable in our HMMP model transformation. While we don't directly address this attack in our work, we offer one possible solution. Whenever a party misbehaves, that party is permanently kicked out of the protocol and the provers continue working. Eventually, all malicious parties are removed from the protocol and from this point forward, the remaining provers enjoy the efficiency guarantees given by Theorem 1.2 with $t = 0$, at the one time cost of honest provers performing $\widetilde{O}_\lambda(\tau \cdot N)$ more work. Another possible solution is to implement a reputation system to punish misbehaving provers; this could amortize the damage done by malicious provers. In any case, the adversarial model we define in this work only considers adversaries try to force the verifier to reject a true statement.

1.2 Additional Related Work

Interactive arguments have a rich history of research over the past four decades [BCC88, GMR89, Mic00, Val08, GW11, SG11, DFH12, BCCT12, GGPR13, BCCT13, BCI+13]. The recent works of Ozdemir [OB22] and Dayama et al. [DPP+22] construct privacy preserving delegation schemes, and define their model in a similar way to ours. However, their goals are to preserve privacy, which is orthogonal to our goal of increasing efficiency while assuming no privacy. Another result loosely related to our work is due to Wu et al. [WZC+18]. [WZC+18] constructs DIZK, which works to distribute the creation of a zero-knowledge proof across multiple machines controlled by the prover, such as a compute cluster. This achieves something much closer to "centralized decentralization", where a single prover distributes their computation to many (honest) trusted nodes. In contrast, our model supports any number of distinct provers that wish to work together to convince a verifier of a statement, with only an honest-majority assumption. Also somewhat related is the recent work of Naor, Parter, and Yogev [NPY20]. Here, the authors construct interactive proofs with *distributed verifiers*. In this setting, the verifiers are represented by an n node graph G defining their communication pattern, and a prover communicates with all nodes via short messages.

Polynomial commitment schemes are a relatively new cryptographic primitive introduced by Kate, Zaverucha, and Goldberg [KZG10]. Several variants of polynomial commitment schemes have been explored since their introduction, including privately verifiable schemes [KZG10, PST13], publicly-verifiable schemes with trusted setup [BFS20], and zero-knowledge schemes [WBT+17]. Recent results have focused on obtaining publicly-verifiable schemes without a trusted setup [BCC+16, WBT+17, BBB+18, WTS+18, BSBHR18, KPV19, BSGKS20, BFS20, Set20, Lee21, SL20, ZXZS20]. More recently, Block et al. [BHR+20, BHR+21] construct the first *space-efficient* polynomial commitment schemes, where both the prover and verifier space is poly-logarithmic in the size of the polynomial, and recent work [BCHO22] expands these results to rank-1 constraint satisfaction problems. Finally, we mention the work of [GLS+21] and its follow-up [XZS22], which utilize error-correcting codes and Merkle hashing to obtain linear time SNARKs.

2 Technical Overview

We give an overview of Theorems 1.1 and 1.2. At a high level, for Theorem 1.1 we utilize a shuffle and audit strategy to ensure consistency among all provers. For Theorem 1.2, we identify three core components of the polynomial IOP + polynomial commitment scheme paradigm and efficiently divide the these components among M provers to get efficient arguments.

2.1 Overview of Theorem 1.1

Fix an argument scheme $\Pi_{\text{arg}} = (\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$, where $\mathsf{Setup}_{\text{arg}}$ is the setup algorithm and P_{arg} and V_{arg} are interactive algorithms representing the prover and verifier, respectively. Let $(C_0, C_1, \ldots, C_r)$ be (arithmetic) circuits that represent the next message function of P_{arg} during round i of the interactive argument for $i \in \{0, 1, \ldots, r\}$. Then as discussed before, we can perform the following process. Assume for simplicity that M divides $|C_i|$ for each i. Then we can partition each circuit C_i into sub-circuits $C_{i,j}$ each of size $|C_i|/M$ for $j \in \{1, \ldots, M\}$ such that the input gates of $C_{i,j}$ only depend on the output gates of $C_{i,k}$ for $k < j$ and the input wires of C_i. Note that such a partition can be obtained by utilizing a globally agreed upon topological sorting of each circuit.

2.1.1 Straw Man Transformation

Given these circuits, one can envision the following straw man transformation from Π_{arg} to $\Pi_{\text{hmp}} = (\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$, where $\boldsymbol{P}_{\text{hmp}} = (P_1, \ldots, P_M)$ are M collaborating provers.

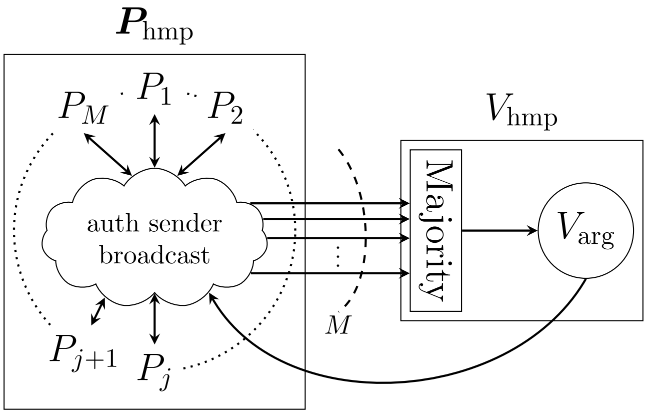

Figure 1: Diagram of the interaction between $\boldsymbol{P}_{\text{hmp}}$ and V_{hmp}.

1. Define $\mathsf{Setup}_{\text{hmp}} := \mathsf{Setup}_{\text{arg}}$.
2. Define $\boldsymbol{P}_{\text{hmp}}$ as follows: for any input z and interaction round i,
   (a) In order for $j \in \{1, \dots, M\}$, prover P_j computes $y_j = C_{i,j}(z, y_1, \dots, y_{j-1})$ and broadcasts y_j.
   (b) All provers use $(y_1, \dots, y_M)$ that correspond to the output wires of C_i to compute $y = C_i(z)$ (e.g., the first bit of y_1 may be the first output wire of C_i) and broadcast y to V_{hmp}.
3. Define V_{hmp} as follows: during any round of interaction, upon receiving M messages from $\boldsymbol{P}_{\text{hmp}}$, take the majority and feed this value into V_{arg}. If V_{arg} returns a message, broadcast that message to $\boldsymbol{P}_{\text{hmp}}$. Otherwise accept or reject as V_{arg}. Fig. 1 gives a pictorial representation of this interaction.

Clearly the above transformation is secure when *all* provers are honest; however, it is clear that even a *single* malicious prover $\widetilde{P}_j$ could inject arbitrary values into the computation. The key observation is that we do not take advantage of our honest majority assumption. Of course, one could use traditional MPC techniques here; however, these are heavy-handed and introduce undesired inefficiencies since these techniques aim to protect the privacy of the individual provers. As each prover does not have a private input, this is wasteful. Instead, we leverage the honest majority assumption by utilizing a *permute and audit* mechanism to allow for cross-examination among provers.

2.1.2 Secure Transformation.

We modify the above straw man scheme in the following two ways. First, we introduce a τ *audit window* to the protocol: during round i of the interaction, every $C_{i,j}$ is computed by τ provers $P_j, P_{j+1}, \dots, P_{j+\tau-1}$, and their results are broadcast to all other provers. Then, all provers check to see if the τ submitted values are all the same. If they are not, then a *global audit* is initiated. In this global audit, *all* provers $P_1, \dots, P_M$ compute the value $C_{i,j}$, broadcast the result of this computation, and then take the majority of the answers. The computed majority value is then compared with the τ submitted values, and any prover that submitted an incorrect value is kicked from the protocol. By our honest majority assumption, if a global audit is initiated then it is guaranteed to *only* remove dishonest parties from the protocol; this is an invariant property of our audit procedure (see Lemma 4.7).

However, even introducing the above audit window is not sufficient for security unless $\tau > t$. In particular, if $\tau \leqslant t$, then an adversary can simply corrupt τ provers in quick succession and is still able to attack the protocol. Further, setting $\tau > t$ is undesirable; if $t = c \cdot M$ for some constant $c < 1/2$, then every prover has a $\tau = \Omega(M)$ blow-up in computational complexity, and at that point it would be simpler for all provers to compute C_i, defeating our efficiency goals for the HMMP model.

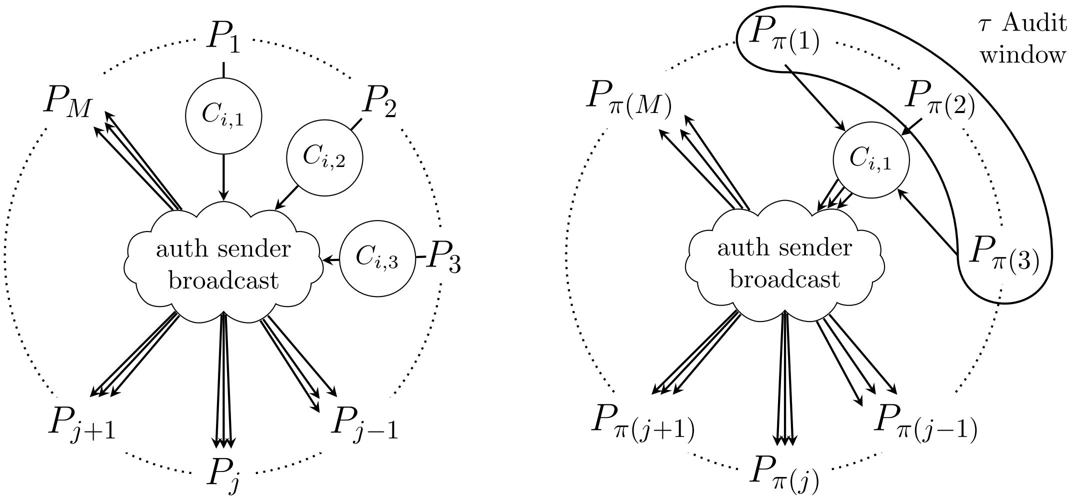

Figure 2: Demonstration of the insecure straw man implementation of $\boldsymbol{P}_{\text{hmp}}$ (pictured left) and the secure implementation of $\boldsymbol{P}_{\text{hmp}}$ (pictured right) with the τ audit window for computing $C_{i,1}$, for $\tau = 3$. Here π is a uniformly random permutation on the set $\{1, \ldots, M\}$ and i is the current round of interaction.

To remedy this, we introduce a uniformly random permutation π on the set $[M] := \{1, \ldots, M\}$ and modify the above audit procedure as follows. In round i, the computation of circuit $C_{i,j}$ is performed by parties $P_{\pi(j)}, \ldots, P_{\pi(j+\tau-1)}$, the result is broadcast, and we follow the remainder of the above audit procedure as prescribed. So long as the permutation is sampled *after* the adversary performs t corruptions, then with good probability (which we can tune via our choice of τ) every audit window will contain at least one honest prover. So long as at least one honest prover is in every audit window, then any incorrect behavior from corrupt provers will be caught by a global audit.

This leads to the following secure transformation of Π_{arg} to Π_{hmp}.

1. Define $\mathsf{Setup}_{\text{hmp}}$ as follows: sample $\mathsf{pp}' \leftarrow \mathsf{Setup}_{\text{arg}}$ and sample uniformly random permutation $\pi \colon [M] \to [M]$. Output $\mathsf{pp} = (\mathsf{pp}', \pi)$.

2. Define $\boldsymbol{P}_{\text{hmp}}$ as follows: for any input z and interaction round i,
   (a) For $j \in [M]$, provers $P_{\pi(j)}, \ldots, P_{\pi(j+\tau-1)}$ compute $y_{i,j} = C_{i,j}(z, y_1, \ldots, y_{j-1})$ and broadcasts $y_{i,j}$.
   (b) All provers audit the value $y_{i,j}$ as necessary.
   (c) After all values $y_1, \ldots, y_M$ are computed, all provers compute $y = C_i(z)$ appropriately and broadcast y to V_{hmp}.

   Fig. 2 presents this transformation pictorially (on the right), along with the insecure transformation (on the left).

3. V_{hmp} is defined analogously as before.

By definition, our HMMP model requires an adversary to perform t static corruptions *before* the generation of public parameters by $\mathsf{Setup}_{\text{hmp}}$. Thus $\mathsf{Setup}_{\text{hmp}}$ samples the permutation π and we obtain a secure transformation.[4]

[4] Alternatively, the verifier can sample π and send it as its first message, after t static corruptions have occurred.

Correctness. Suppose that Π_{arg} has correctness error δ_{arg}; that is, the probability that V_{arg} rejects a correct proof from an honest prover P_{arg} is at most δ_{arg}. Then our transformation guarantees correctness error δ_{arg} so long as every τ audit window contains at least one honest prover.

Proposition 2.1. *Let Π_{arg} be an argument with δ_{arg} correctness error and let Π_{hmp} be the argument obtained by applying Theorem 1.1. Let $\mathcal{H} \subseteq [M]$ be the indices of the honest provers and let π be a permutation over $[M]$. For every $i \in [M]$, if $\mathcal{H} \cap \pi(\mathcal{J}) \neq \emptyset$ for $\mathcal{J} = \{i, i+1, \ldots, i+\tau-1\}/(M\mathbb{Z})$ then the argument Π_{hmp} has correctness error δ_{arg}.*

Clearly, if the permutation π is fixed ahead of time before the corruption of t provers, then there is always a way for the adversary to construct a set $\mathcal{J}$ such that $\mathcal{H} \cap \pi(\mathcal{J}) = \emptyset$. Hence, we enforce the corruption of t provers before a uniformly random permutation π is sampled. Then we can guarantee with suitably high probability that Proposition 2.1 holds. The following lemma characterizes the probability that Proposition 2.1 holds over a uniformly random permutation.

Lemma 2.2. *Let $\tau, t, M \in \mathbb{N}$ be integers such that $\tau \leqslant t$ and $t < M$, and let $T \subset \{1, \ldots, M\}$ be a subset of size t. For permutation π over $[M]$, let $S_{(i,j)} = \{S_{\ell,\pi} \colon \ell = (i + k \mod M)\ \forall k \in \{0, \ldots, j-1\}\}$, where $S_{\ell,\pi} = \pi(\ell)$ for all $\ell \in \{1, \ldots, M\}$. Then $\Pr\left[\exists i \in [M] \colon S_{(i,\tau)} \subseteq T\right] \leqslant M \cdot (t/M)^{\tau}$, where the probability is taken over uniformly random permutation π.*

Lemma 2.2 states that the probability there exists an audit window with no honest provers is at most $M \cdot (t/M)^{\tau}$; i.e., with probability at least $1 - M \cdot (t/M)^{\tau}$, $\boldsymbol{P}_{\text{hmp}}$ behaves identically to the honest prover algorithm P_{arg}. Note τ is a parameter for the argument Π_{hmp} and can be chosen such that $M \cdot (t/M)^{\tau} = \mathsf{negl}(M)$, e.g., $\tau = \log^{1+\epsilon}(M)$ for small constant $\epsilon > 0$. Proposition 2.1 and Lemma 2.2 together guarantee that correctness holds with probability at least $(1 - \delta_{\text{arg}}) \cdot (1 - M \cdot (t/M)^{\tau})$, thus our correctness error is $\delta_{\text{hmp}} = 1 - (1 - \delta_{\text{arg}}) \cdot (1 - M \cdot (t/M)^{\tau})$ as desired.

Soundness. As stated in Section 1.1, we require soundness to hold with respect to all M provers being corrupt. Informally, we say that a HMMP model argument Π_{hmp} is ε-computationally sound if the probability M malicious collaborating provers convince a verifier of a false statement with probability at most ε. We similarly extend our soundness definition to *knowledge soundness* [BG92]. Informally, we say that a HMMP model argument Π_{hmp} has knowledge soundness if whenever M malicious collaborating provers cause a verifier to accept, then a valid (NP) witness can be efficiently extracted via an extractor by interacting with and rewinding the M malicious provers. Our definitions mirror both [OB22, DPP+22].

Under these definitions, it is not difficult to see that both soundness and knowledge soundness directly reduce to the single-prover argument case (this observation is similarly made in both [OB22, DPP+22]). In particular, any M malicious prover strategy that is successful in breaking soundness is itself a single-prover strategy for breaking soundness: the single prover simply simulates the interaction "in its head". For knowledge soundness, from an extractor perspective, any property that holds for a malicious prover also holds for a collection of malicious provers. See Section 4 and Proposition 4.10 for more details.

Defining and Obtaining Zero-Knowledge. We define zero-knowledge analogously to [DPP+22]. In particular, we say that a HMMP model argument is zero-knowledge with error γ if any malicious verifier interacting with M honest collaborating prover algorithms learns nothing other than the validity of the statement, except with probability at most γ. In our setting this definition of zero-knowledge is the most natural since all M provers are given the same witness and have no other private inputs to protect.

If collaborating provers each have some private input to protect, then one can define zero-knowledge with respect to an honest verifier and t malicious provers by stipulating the malicious provers learn nothing by interacting with the honest provers; [OB22] defines t-zero knowledge in this way, and [DPP+22] calls this notion of zero-knowledge *witness confidentiality*. Moreover, [DPP+22] defines another zero-knowledge variant with respect to t malicious provers *and* a malicious verifier, which they call *witness confidentiality with collusion*. While both of these notions of zero-knowledge are interesting, they do not apply (nor make sense) in our case since all provers have the same inputs.

As stated in Theorem 1.1, our transformation preserves zero-knowledge, which can be seen via the following abstraction. Given a zero-knowledge argument Π_{arg}, the prover next message function during round i can be viewed as two circuits: the "non-zero knowledge" circuit C_i and the "zero-knowledge" circuit D_i. Given sufficiently many random coins $r \in \{0,1\}^*$, C_i, and input z, if $C_i(z)$ is sent by the prover in the non-zero knowledge setting, then $D_i(C_i(z); r)$ is sent in the zero-knowledge setting. This abstraction allows us to preserve zero-knowledge by giving $\boldsymbol{P}_{\text{hmp}}$ a shared random string r as input in addition to the witness. Each prover then uses the same random string r and sends $y'_i = g(y; r)$, where y is computed as in Item 2c. In place of receiving r as input, one can assume a randomness beacon [Rab83, SJK$^+$17] to broadcast r, or leverage honest-majority MPC to agree upon r [BOGW88, RBO89]. Moreover, in the same way we divided the computation of C_i among our M provers, we can divide the computation of D_i among our M provers as well (e.g., if D_i is a large circuit).

Obtaining Non-Interactivity. Briefly, a non-interactive argument is an argument Π_{arg} where only a single message is sent from the prover to the verifier, and there is no other interaction. Similarly, we say a HMMP model argument Π_{hmp} is non-interactive if M collaborating and communicating provers generate a single message (i.e., the proof string), send this message to the verifier, and there is no other interaction.

First note that if Π_{arg} was already a non-interactive argument, the Π_{hmp} obtained via Theorem 1.1 is trivially non-interactive. Moreover, if Π_{arg} is interactive and can be securely made non-interactive via the Fiat-Shamir transformation, then the argument Π_{hmp} obtained by applying Theorem 1.1 can be made non-interactive via the Fiat-Shamir transformation in the following way. All provers are assumed to have access to a shared random oracle H, and the provers each use H to generate the next message of the verifier. In particular, instead of sending M messages to the verifier, each prover takes the majority of the messages it would have sent to the verifier, appends this new message to the transcript so far, then applies the random oracle H to the updated transcript and uses the output as the verifier message. Security now holds information theoretically in the random oracle model: if any dishonest prover does not follow this procedure, then with high probability over the random oracle, the verifier challenges generated in a dishonest way will be incorrect, which guarantees that with high probability the dishonest parties are caught in subsequent rounds.

2.2 Overview of Theorem 1.2

Recall that Theorem 1.1 does not guarantee efficiency gains for collaborating provers, which is the goal of our model, as the circuits C_i may be large especially when obtained via RAM to circuit transformations. We overcome this limitation of Theorem 1.1 by directly applying our HMMP model to a class of arguments for NP that follow the paradigm of compiling polynomial interactive oracle proofs with polynomial commitment schemes. Arguments obtained via this paradigm have algebraic structure that we can take advantage of during the collaborative proof generation. We first give an overview of the paradigm, then discuss how to efficiently distribute the computation of each sub-component, then describe how to compose these components to obtain Theorem 1.2.

2.2.1 Paradigm Overview

As mentioned before, this paradigm compiles any polynomial interactive oracle proof (polynomial IOP, PIOP) for NP with a suitable polynomial commitment scheme to obtain interactive arguments for NP. We begin by giving an overview of polynomial commitment schemes, or polynomial commitments in short. Briefly, polynomial commitments consist of a tuple of algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$. This tuple allows a sender to (non-interactively) produce a commitment $C = \mathsf{Com}(f)$ for polynomial f, open a commitment C to a polynomial f using Open, and additionally supports "proofs of evaluation" using Eval. The Eval algorithm is an interactive protocol between the sender and receiver which does the following: on input commitment C, evaluation point x from the receiver, and value y from the sender, Eval certifies that the sender knows a polynomial f such that $f = \mathsf{Open}(C)$ and $y = f(x)$. Intuitively, Eval allows the sender to open the commitment C at specific evaluation points without revealing the entire polynomial. Crucially, this proof of evaluation requires far less computation than just opening the entire polynomial f.

Given the outline of a polynomial commitment scheme, we turn to polynomial IOPs. We focus on PIOPs where the prover sends a single oracle, and the rest of the communication between the prover and verifier is succinct (i.e., poly-logarithmic in the size of the witness). These PIOPs (roughly) operate as follows:

1. The first message sent by the prover is an oracle proof string consisting of all evaluations of a (low-degree) polynomial, say $f \in \mathbb{F}[X_1, \ldots, X_n]$ for finite field $\mathbb{F}$. The verifier has query access to these evaluations.
2. The prover and verifier engage in a sum-check protocol [LFKN92] over some auxiliary (low-degree) polynomial,say $p \in \mathbb{F}[X_1, \ldots, X_n]$, that depends on f.
3. At the end of the sum-check, the verifier has some claim of the form "$p(r) = y$" for $r \in \mathbb{F}^n$ sampled uniformly at random. The verifier then makes a constant number of queries to the oracle f to verify this claim, and accepts or rejects.

To obtain arguments, we compile the above PIOP with a polynomial commitment. This is done by (1) having the prover commit to the polynomial f in Item 1 using Com; and (2) for every query to f made by the verifier in Item 3, the prover and verifier instead run the Eval protocol to obtain these queries. The remainder of the PIOP remains the same; i.e., the prover and verifier still run a sum-check protocol. Note this extends to PIOPs that send multiple oracles: simply commit to each oracle and run Eval whenever a query is made.

This paradigm yields interactive arguments for all of NP [WTS+18, BFS20, BHR+20, Set20, BHR+21, Lee21, BCHO22, GLS+21, XZS22]. Therefore, if we extend the above paradigm to the HMMP model with the efficiency gains proportional to $1/M$ for M provers, then we obtain Theorem 1.2 when utilizing the Bulletproofs polynomial commitment scheme [BCC+16, BBB+18]. There are three tasks to implement in the HMMP model: (1) generating the polynomial f in Item 1; (2) computing the sum-check in Item 2; and (3) computing the commitment for Item 1 and the proof of evaluation for Item 3; i.e., computing the polynomial commitment. We concretely consider the following problems in the HMMP model to address these tasks. First, we give a HMMP model protocol for circuit satisfiability. Second, we give a HMMP model protocol for the sum-check protocol. Third, we instantiate the well-known Bulletproofs polynomial commitment scheme in the HMMP model. In what follows, we give an overview of these protocols along with formal propositions outlining their efficiency guarantees. All together, this yields Theorem 1.2.

We first fix some notation. We focus on constructing an argument for the NP complete language of (arithmetic) circuit satisfiability. Let $\mathbb{F}$ be a finite field and let $C \colon \{0,1\}^* \to \mathbb{F}$ be an arithmetic circuit of size N; i.e., C has N wires (including input and output wires), and gates in C perform addition and multiplication over $\mathbb{F}$. Let $M \leqslant N$ be the number of provers, let $t < M/2$ be the number of corrupt parties, and let $\tau \leqslant t$ be the audit window parameter.

2.2.2 Circuit SAT

Generally, the polynomial f sent in Item 1 of a PIOP is the encoding of a wire transcript (i.e., values of wires) of an arithmetic circuit C of size N on input x, which corresponds to proving circuit satisfiability. For arithmetic circuits, this is proving that $C(x) = y$ for some $y \in \mathbb{F}$. Our first step is to give an efficient HMMP model protocol for producing this transcript.

For simplicity, assume that N/M is an integer. Partition the circuit C into M partitions $C_1, \ldots, C_M$ each of size N/M such that for any $j \in \{1, \ldots, M\}$, partition C_j only depends on wires from partitions C_i for $i \leqslant j$. Then the M provers $(P_1, \ldots, P_M)$ produce a wire transcript of C as follows. Let x be the input to the circuit C. For $j = 1, 2, \ldots, M$, audit window $P_{\pi(j)}, P_{\pi(j+1)}, \ldots, P_{\pi(j+\tau-1)}$ compute the wire transcript of partition C_j, using input x and previously obtained transcripts for circuits C_i for all $i < j$. The audit window then broadcasts their computed transcript, invoking a global audit and re-balance as necessary.

At the end of the protocol, each party combines all partial transcripts in order to obtain the final wire transcript for C. Since each C_j consists of N/M gates, each audit window performs $O(N/M)$ computations to generate the wire transcript for C_j, yielding $O(\tau \cdot (N/M))$ arithmetic gate evaluations for each prover.[5]

[5] We do not count the work required to write down the entire transcript, as this cost is minor in comparison to generating the actual transcript.

For simplicity, we assumed that N/M was an integer. However, this is not necessary: we can instead let $k = \lfloor N/M \rfloor$ and partition C into at most $2M$ partitions $C_1, \ldots, C_{2M}$ each of size at most k with the same dependency properties as before. This increases individual prover complexity by a factor of at most 2, yielding the same asymptotic results. The following proposition formally captures our HMMP model protocol for circuit satisfiability.

Proposition 2.3. *There exists an interactive proof in the HMMP model for arithmetic circuit satisfiability with the following properties: for circuits of size N over finite field $\mathbb{F}$ and parameters τ, M such that $M \leqslant N$,*

- *each prover computes $O(\tau \cdot (N/M))$ arithmetic gate evaluations; and*
- *each prover broadcasts $O(\tau \cdot (N/M))$ field elements.*

2.2.3 Sum-Check Protocol

Let $f \in \mathbb{F}[X_1, \ldots, X_n]$ be a polynomial of constant individual degree (i.e., $\deg_i(f) = \Theta(1)$ for all i). The sum-check protocol [LFKN92] is an interactive proof certifying that $\sum_{x \in \{0,1\}^n} f(x) = y$ for some $y \in \mathbb{F}$. The protocol proceeds for n rounds, where in the first round the prover sends y and for every round $j = 1, \ldots, n$, the prover computes a polynomial $g_j(X_j) := \sum_{x \in \{0,1\}^{n-j}} g(r, X_j, x)$, where $r = (r_1, \ldots, r_{j-1})$ are random field elements sent by the verifier in each prior round.

During every round $j \in \{1, \ldots, n\}$ of the protocol, the prover evaluates g at 2^{n-j} points to compute the constant-degree polynomial $g_j(X_j)$. Because computing g_j is linear, we can sub-divide this computation into M different sums. For simplicity, assume that $2^{n-j}/M$ is an integer. Then we can partition $\{0,1\}^{n-j}$ into subsets $S_1, \ldots, S_M \subset \{0,1\}^{n-j}$ each of size $2^{n-j}/M$. For every $i \in \{1, \ldots, M\}$, the audit window $P_{\pi(i)}, \ldots, P_{\pi(i+\tau-1)}$ computes and broadcasts the polynomial $g_{j,i}(X_j) := \sum_{x \in S_i} g(r, X_j, x)$. Note that g has constant individual degree, so each audit window broadcasts $O(1)$ field elements. After broadcasting all $g_{j,i}$, each prover reconstructs the polynomial $g_j := \sum_i g_{j,i}$, sends it to the verifier, and proceeds to the next round.

Each audit window performs $2^{n-j}/M$ evaluations of g and $O(2^{n-j}/M)$ field additions to compute $g_{j,i}$. Additionally, each prover performs $O(M)$ field additions to reconstruct g_j since g_j has constant degree. Thus each prover performs $O(\tau \cdot (2^{n-j}/M))$ evaluations of g and $O(\tau \cdot (2^{n-j}/M) + M)$ field additions in round j. As with the circuit SAT algorithm, if $2^{n-j}/M \geqslant 1$ and is not an integer, we can set $k = \lfloor 2^{n-j}/M \rfloor$ and partition $\{0,1\}^{n-j}$ into $S_1, \ldots, S_{2M}$ sets of size at most k. This increases the prover complexity by a factor of at most 2, yielding the same asymptotic complexity. Thus in the worst case, the total prover computation over all rounds $j = 1, \ldots, n$ is $O(\tau \cdot (2^n/M))$ evaluations of g and $O(\tau \cdot (2^n/M) + n \cdot M)$ field additions, and each prover broadcasts $O(\tau \cdot n)$ field elements.

We conveniently assumed that $2^{n-j}/M \geqslant 1$, but it is possible that in some round j we have $2^{n-j} < M$. To handle this case, consider j^* as the unique index such that $2^{n-j^*} < M \leqslant 2^{n-j^*+1}$. Then we let the first 2^{n-j^*} audit windows evaluate g at a single point to construct g_{j^*}; i.e., they compute $g_{j^*,i} = g(r, X_{j^*}, i)$ where $i \in \{0, \ldots, 2^{n-j^*} - 1\}$ is interpreted in binary in the natural way. Then the next 2^{n-j^*-1} audit windows evaluate g at a single point and construct g_{j^*+1} as above. This continues until a single audit window constructs the polynomial g_n.

By dividing up the computation this way, we ensure that every prover computes $O(\tau)$ evaluations of g and $O(M)$ field additions and broadcasts $O(\tau)$ field elements over *all* rounds $j^*, \ldots, n$. To see this, notice that $2^{n-j^*} < M \leqslant 2^{n-j^*+1} < 2M$; moreover $\sum_{j=j^*}^{n} 2^{n-j} < 2M$. Thus, our above division of computation only requires each audit window to evaluate g at at most 2 points, giving the stated complexity $O(\tau)$. Additionally, each round $j = j^*, \ldots, n$ requires 2^{n-j} additions to reconstruct the polynomial g_j, giving $O(M)$ additions overall. We capture our sum-check protocol in the following corollary.

Corollary 2.4. *For any n-variate polynomial g over finite field $\mathbb{F}$ of constant individual degree, there exists a HMMP model protocol for computing the sum-check protocol of f on $\{0,1\}^n$ such that for parameters τ and $M \leqslant 2^n$,*

- *each prover computes $O(\tau \cdot (2^n/M))$ evaluations of g and $O(\tau \cdot (2^n/M) + n \cdot M)$ field additions; and*

- *each prover broadcasts $O(\tau \cdot n)$ field elements.*

We actually prove a more general result about the sum-check protocol for polynomials of individual degree d being summed over a set H^n, where $H \subset \mathbb{F}$. See Proposition 5.1 and Section 5.2 for details.

2.2.4 Polynomial Commitment Scheme

Recall that polynomial commitments allow a sender to commit to a polynomial and additionally allow the sender to open the polynomial at specific evaluations specified by the receiver without opening the entire polynomial. Such schemes are the continued subject of recent research and have many attractive features such as succinct proofs, small commitments, and their use in the PIOP + polynomial commitment paradigm to obtain succinct arguments. As such, there are many different schemes from a variety of cryptographic assumptions, including discrete-log [BCC+16, BBB+18, WTS+18, BHR+20], groups of unknown order [BFS20, BHR+21], bilinear pairings [KZG10, Lee21, OB22, BCHO22], and error-correcting codes with Merkle Hashing [GLS+21, XZS22].

In this work, we do not tackle implementing every polynomial commitment scheme in the HMMP model. We instead focus on the well-known and studied Bulletproofs scheme due to Bootle et al. [BCC+16] and Bünz et al. [BBB+18]. We give the complete details of how we implement Bulletproofs in the HMMP model in Section 5.3. Here, we give an overview of the core components of Bulletproofs and how to implement them efficiently in the HMMP model.

The Bulletproofs polynomial commitment scheme utilizes a vector-commitment and an inner product. Let $Y \in \mathbb{F}^N$ be a vector given to a prover and let $X \in \mathbb{F}^N$ be a public vector, and for simplicity assume $N = 2^n$. Then the prover commits to Y using a Pedersen commitment [Ped91] over a prime-order group $\mathbb{G}$. For generators $g_1, \ldots, g_N$, the commitment to Y is $C := \prod_i g_i^{Y_i}$. Computation of C is easily divisible among M provers: let N/M be an integer and partition the set $\{1, \ldots, N\}$ into subsets $S_1, \ldots, S_M$ each of size N/M. Then audit window $P_{\pi(i)}, \ldots, P_{\pi(i+\tau-1)}$ computes and broadcasts the partial commitment $C_i = \prod_{j \in S_i} g_j^{Y_j}$. One all C_i are obtained, each prover locally computes the commitment $C = \prod_i C_i$. Therefore each prover performs $O(\tau \cdot (N/M))$ group exponents and $O(\tau \cdot (N/M) + M)$ group multiplications, and each prover broadcasts $O(\tau)$ group elements. As before, if N/M is not an integer, we can partition $\{1, \ldots, N\}$ into subsets $S_1, \ldots, S_{2M}$ each of size at most $\lfloor N/M \rfloor$, and each prover does at most two times the computation as the case where N/M is an integer.

The inner-product argument uses a halving protocol to reduce the inner-product claim $\gamma = \langle X, Y \rangle$ into another inner-product claim $\gamma' = \langle X', Y' \rangle$, where $X', Y' \in \mathbb{F}^{N/2}$. This is done by splitting X and Y into (X_L, X_R) and (Y_L, Y_R), where X_L, Y_L are the first $N/2$ symbols of X, Y, respectively, and X_R, Y_R are the last $N/2$ symbols of X, Y, respectively. The prover computes "cross inner-products" $\gamma_0 = \langle X_R, Y_L \rangle$ and $\gamma_1 = \langle X_L, Y_R \rangle$. Then, given a random verifier challenge α, new vectors X', Y' are computed as $\alpha X_L + \alpha^{-1} X_R$ and $\alpha^{-1} Y_L + \alpha Y_R$; moreover, γ' is defined using $\alpha, \gamma_0, \gamma_1$, and γ such that $\gamma' = \langle X', Y' \rangle$. This halving protocol continues until the vector Y' is of length 1.

We divide the above computation among M provers, then generalize it to any round $j = 1, \ldots, n$ of the halving protocol. Suppose that $N/(2M)$ is an integer. Partition X_L, X_R and Y_L, Y_R into M vectors $X_{L,i}, X_{R,i}$ and $Y_{L,i}, Y_{R,i}$ each of length $N/(2M)$ for all $i \in \{1, \ldots, M\}$. Then each audit window $P_{\pi(i)}, \ldots, P_{\pi(i+\tau-1)}$ computes and broadcasts partial sums $\gamma_{0,i} = \langle X_{R,i}, Y_{L,i} \rangle$ and $\gamma_{1,i} = \langle X_{L,i}, Y_{R,i} \rangle$. Once all $\gamma_{0,i}, \gamma_{1,i}$ are obtained, each prover locally computes $\gamma_0 = \sum_i \gamma_{0,i}$ and $\gamma_1 = \sum_i \gamma_{1,i}$. Upon receiving α from the verifier, each audit window now computes partial parts of X', Y'. That is, audit window $P_{\pi(i)}, \ldots, P_{\pi(i+\tau-1)}$ computes and broadcasts vectors $X'_i = \alpha X_{L,i} + \alpha^{-1} X_{R,i}$ and $Y'_i = \alpha^{-1} Y_{L,i} + \alpha Y_{R,i}$. Once all X'_i, Y'_i are obtained, each prover locally computes $X' = (X'_1, \ldots, X'_M)$ and $Y' = (Y'_1, \ldots, Y'_M)$.

For general round $j = 1, \ldots, n$, if $2^{n-j-1}/M \geqslant 1$ is an integer, then each audit window performs $O(2^{n-j}/M)$ field multiplications and additions to compute values $\gamma_{0,i}, \gamma_{1,i}$ and vectors X'_i, Y'_i. Each audit window also broadcasts $O(2^{n-j}/M)$ field elements, and each prover performs $O(M)$ additions to reconstruct γ_0, γ_1 and $O(\tau \cdot (2^{n-j}/M))$ additions to reconstruct vectors X', Y'. Thus each prover performs $O(\tau \cdot (2^{n-j}/M))$ field multiplications and $O(\tau \cdot (2^{n-j}/M) + M)$ additions, and broadcasts $O(\tau \cdot (2^{n-j}/M))$ field elements. This gives that over all rounds $j = 1 \ldots, n$, each prover performs $O(\tau \cdot (N/M))$ field multiplications and

$O(\tau \cdot (N/M) + n \cdot M)$ additions, and broadcasts $O(\tau \cdot n \cdot (N/M))$ field elements. Again we note that if $2^{n-j-1}/M \geqslant 1$ is not an integer, we obtain the same asymptotic complexities. By the same ideas as the division of the sum-check, if j^* is the unique round where $2^{n-j^*-1} < M \leqslant 2^{n-j^*}$, we can divide up the remaining computation such that each prover only performs $O(\tau)$ field multiplications, $O(M)$ additions, and broadcasts $O(\tau)$ field elements over all rounds $j = j^*, \ldots, n$. Thus our total prover complexity is $O(\tau \cdot (N/M))$ field multiplications and $O(\tau \cdot (N/M) + n \cdot M)$ field additions, and each prover broadcasts $O(\tau \cdot n \cdot (N/M))$ field elements.

In the context of Bulletproofs, each prover also computes "folded" Pedersen commitments. In fact, we divide up computation of these commitments in the same way as we divide up the inner products above. This results in similar complexities for group exponents and multiplications.

Proposition 2.5. *Assuming the hardness of discrete-log in prime-order groups, there exists a HMMP model inner-product argument with the following properties: for vectors of length $N = 2^n$ over a finite field and parameters τ, M such that $M \leqslant N$,*

- *each prover computes $O(\tau \cdot (N/M))$ field multiplications and group exponents, $O(\tau \cdot (N/M + n \cdot M))$ field additions and group multiplications; and*
- *each prover broadcasts $O(\tau \cdot n \cdot (N/M))$ field and group elements.*

2.2.5 Efficient HMMP Model Arguments for NP.

Given the above building blocks, we now combine them all to obtain Theorem 1.2. Let $\mathbb{F}$ be a finite field and $\mathbb{G}$ be a prime-order group for some λ-bit prime. Let $\boldsymbol{P}_{\text{hmp}}$ denote our HMMP model prover algorithm

Using our Circuit SAT algorithm (Proposition 2.3), $\boldsymbol{P}_{\text{hmp}}$ obtains a wire transcript. We view this wire transcript as a function $f\colon \{0,1\}^n \to \mathbb{F}$, where $N = 2^n$ is the size of the arithmetic circuit. While f may not be a polynomial, it uniquely defines a multi-linear extension $\tilde{f}\colon \mathbb{F}^n \to \mathbb{F}$ (see Fact 5.4). A multi-linear extension is a polynomial with individual degree at most 1 such that $\tilde{f}(x) = f(x)$ for all $x \in \{0,1\}^n$.

Using our Bulletproofs algorithm, $\boldsymbol{P}_{\text{hmp}}$ commits to $\tilde{f}$ using a Pedersen commitment to commit to the vector $Y \in \mathbb{F}^N$ defined as $Y_i = f(i-1)$, where $i-1$ is viewed as an n-bit string in the natural way. The prover and verifier then engage in a sum-check over some auxiliary polynomial g of constant individual degree that depends on the polynomial $\tilde{f}$ and some other efficiently computable constant individual degree polynomials; in particular, these other polynomials are computable in $O(\log(N))$ time [CMT12, Tha13, XZZ+19].

After engaging in the sum-check protocol, the verifier invokes the polynomial commitment scheme to obtain some constant number of evaluations of the polynomial $\tilde{f}$. Note that the evaluation of $\tilde{f}$ is exactly given by an inner product $\langle Y, Z \rangle$, where $Z \in \mathbb{F}^N$ is a vector that depends on an evaluation point x specified by the verifier, and some interpolation polynomials (Fact 5.4); moreover, this vector is computable using $\widetilde{O}(N)$ field multiplications and can be divided up between $\boldsymbol{P}_{\text{hmp}}$ to obtain $\widetilde{O}(\tau \cdot (N/M))$ field multiplications for each prover (Fact 5.4). Thus for any evaluation of $\tilde{f}$ at point x, $\boldsymbol{P}_{\text{hmp}}$ computes the vector Z and engages in the Bulletproofs inner-product argument with the verifier. This happens a constant number of times. Noting that we assume $\mathbb{F}$ and $\mathbb{G}$ operations take $\text{poly}(\lambda)$ time, and that we do not count the cost of writing down field/group elements, Propositions 2.3 and 2.5 and Corollary 2.4 all together yield the desired efficiencies given in Theorem 1.2.

3 Preliminaries

We let $\lambda \in \mathbb{Z}^+$ denote the security parameter. For natural number $n \in \mathbb{N}$, we let $N = 2^n$. A function $\mu\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ is called *negligible* if $1/\mu(n) = o(p(n))$ for any positive polynomial p. For integers $a \leqslant b$, we let $[a,b]$ denote the set $\{a, a+1, \ldots, b\}$, and for positive integer k, we let $[k]$ denote the set $[1,k]$. For integer M and integers $a \leqslant b$, we let $[a,b]_M$ denote the set $\{a \pmod M, (a+1) \pmod M, \ldots, b \pmod M\} \subset \mathbb{Z}_M$. For a finite, non-empty set S, we let $x \xleftarrow{\$} S$ denote sampling x uniformly at random from S. We let $\mathbb{F}$ denote a finite field and $\mathbb{G}$ denote a finite group.

3.1 Interactive Arguments

We give formal definitions of standard interactive arguments. In Section 4, we complement these definitions with our HMMP model definitions.

Definition 3.1 (Witness Relation Ensemble)**.** *A* witness relation ensemble *(or relation ensemble) is a binary relation $\mathcal{R}_{\mathcal{L}}$ that is polynomially bounded, polynomial-time recognizable, and defines the language $\mathcal{L} = \{x \colon \exists w, (x, w) \in \mathcal{R}_{\mathcal{L}}\}$.*

We now define interactive arguments for relation ensembles.

Definition 3.2 (Interactive Arguments)**.** *Let $\mathcal{R}$ be a relation ensemble. Let (Setup, P, V) denote three algorithms such that P and V are a pair of PPT interactive algorithms and Setup is a non-interactive algorithm that outputs public parameters pp on input 1^λ and aux, for any auxiliary input aux. Let $\langle P(w), V(z)\rangle(\mathsf{pp}, x)$ denote the output of V after interacting with P on common inputs pp and x, where P additionally receives w such that $(x, w) \in \mathcal{R}$ and V receives auxiliary input $z \in \{0,1\}^*$. We say that (Setup, P, V) is an* interactive argument *for relation $\mathcal{R}$ if the following hold:*

1. ***δ-Correctness.*** *For all $(x, w) \in \mathcal{R}$ and every $z \in \{0,1\}^*$ we have*

$$\Pr[\langle P(w), V(z)\rangle(\mathsf{pp}, x) = 1] \geqslant 1 - \delta(\lambda),$$

   where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux})$ and the random coins of V.

2. ***ε-Computational Soundness.*** *For every x and w such that $(x, w) \notin \mathcal{R}$, every non-uniform interactive polynomial-sized adversary P^*, and every $z \in \{0,1\}^*$, we have*

$$\Pr[\langle P^*(w), V(z)\rangle(\mathsf{pp}, x) = 1] < \varepsilon(\lambda),$$

   where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux})$ and the random coins of V.

If the above soundness definition holds with respect to unbounded cheating provers, then we say that (Setup, P, V) is an interactive proof *for $\mathcal{R}$.*

Two desirable properties of interactive arguments include public verifiability and public coin, which we define here.

Definition 3.3 (Publicly Verifiable)**.** *An interactive argument (Setup, P, V) is* publicly verifiable *if any party viewing the transcript $T = \langle P(w), V(z)\rangle(x)$ can certify the validity of the interaction.*

Definition 3.4 (Public Coin)**.** *An interactive argument (Setup, P, V) is called* public coin *if all messages sent from V to P are chosen uniformly and independently at random, and independently of the prover's messages.*

Zero-knowledge is often desirable as well. We recall the standard definition of zero-knowledge with auxiliary input.

Definition 3.5 (Zero-knowledge Arguments)**.** *A public-coin interactive argument (Setup, P, V) for relation ensemble $\mathcal{R}$ is said to have* computational zero-knowledge *with respect to an auxiliary input if for every PPT interactive algorithm V^* there exists a PPT simulator $\mathcal{S}$ that runs in polynomial time with respect to its first input such that for every $(x, w) \in \mathcal{R}$ and any $z \in \{0,1\}^*$: $\mathrm{View}(\langle P(w), V^*(z)\rangle(x)) \approx_c \mathcal{S}(x, z)$. Here $\mathrm{View}(\langle P(w), V^*(z)\rangle(x))$ denotes the distribution of the transcript of the interaction between P and V^* and $\approx_c$ denotes computational indistinguishability. If the statistical distance between the two distributions is negligible then the interactive argument is said to have* statistical zero-knowledge*. If the simulator is allowed to abort with probability at most $1/2$, but the distribution of its output conditioned on no abort is identical to $\mathrm{View}(\langle P(w), V^*(z)\rangle(x))$, the the interactive argument is said to have* perfect zero-knowledge*.*

Arguments often need some form of knowledge soundness for security. We recall one such notion: witness-extended emulation.

Definition 3.6 (Witness-extended Emulation [Lin03])**.** *Given a public-coin interactive argument* (Setup, P, V) *and arbitrary prover algorithm* P^**, let* $\mathsf{Rec}(P^*, \mathsf{pp}, x, st)$ *denote the message transcript between* P^* *and* V *on shared input* x*, initial prover state* st*, and* pp *generated by* Setup*. Furthermore, let* $\mathsf{E}^{\mathsf{Rec}(P^*, \mathsf{pp}, x, st)}$ *denote a machine* E *with a transcript oracle for this interaction that can be rewound to any round and run again on fresh verifier randomness. The tuple* (Setup, P, V) *has* witness-extended emulation *if for every deterministic polynomial-time* P^* *there exists an expected polynomial-time emulator* E *such that for all non-uniform polynomial-time adversaries* A *the following holds:*

$$\Pr\left[A(tr) = 1 : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux}),\ (x, st) \leftarrow A(\mathsf{pp}), \\ tr \leftarrow \mathsf{Rec}(P^*, \mathsf{pp}, x, st) \end{array}\right] \approx \tag{1}$$
$$\Pr\left[\begin{array}{l} A(tr) = 1\ \wedge \\ tr \text{ accepting} \implies (x, w) \in \mathcal{R} \end{array} : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux}),\ (x, st) \leftarrow A(\mathsf{pp}), \\ (tr, w) \leftarrow \mathsf{E}^{\mathsf{Rec}(P^*, \mathsf{pp}, x, st)}(\mathsf{pp}, x) \end{array}\right].$$

4 The Honest Majority Multi-Prover Model

In this section, we give our formal definition of the Honest Majority Multi-Prover Model for interactive arguments and prove Theorem 1.1. We begin with the definition of the HMMP model.

Definition 4.1. *The* Honest Majority Multi-Prover (HMMP) Model *is an interactive protocol framework where* M *provers* $P_1, \dots, P_M$ *collectively convince a verifier* V *of the validity of a statement. During any round of communication, the provers* $P_1, \dots, P_M$ *are allowed to communicate and collectively send a message to* V*. Communication is done via an authenticated sender broadcast channel.*

Given Definition 4.1, we define interactive arguments in this model.

Definition 4.2 (HMMP Model Interactive Arguments)**.** *Let* $\mathcal{R}$ *be a relation ensemble and let* $M \in \mathbb{Z}^+$*. Let* $\boldsymbol{P} = (P_1, \dots, P_M)$*, let* $(\boldsymbol{P}, V)$ *denote a pair of PPT interactive HMMP model algorithms, and let* Setup *denote a non-interactive setup algorithm that outputs public parameters* pp *given security parameter* 1^λ*. Let* $\langle \boldsymbol{P}(\mathsf{pp}, x, w), V(\mathsf{pp}, x)\rangle$ *denote the output of* V*'s interaction with* $\boldsymbol{P}$ *on common inputs pp and statement* x*, where* $\boldsymbol{P}$ *additionally has witness* w*. The triple* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *is an* M-prover interactive argument for $\mathcal{R}$ in the HMMP model *if*

1. δ***-Correctness.*** *For every* $(x, w) \in \mathcal{R}$*, for every non-uniform polynomial-sized adversary* A *that statically corrupts* $t < M/2$ *provers before the selection of public parameters, and for all* $z \in \{0,1\}^*$*, we have* $\Pr[\langle \widetilde{\boldsymbol{P}}(w), V(z)\rangle(\mathsf{pp}, x) = 1] \geqslant 1 - \delta(\lambda)$*, where* $\widetilde{\boldsymbol{P}}$ *denotes all honest and corrupt provers, and the probability is taken over* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux})$ *and the random coins of* V*. If* $\delta = 0$*, then we say the argument has* perfect correctness.
2. ε***-Computational Soundness.*** *For every* x *and* w *such that* $(x, w) \notin \mathcal{R}$*, all non-uniform polynomial-sized adversaries* $\widetilde{\boldsymbol{P}} = (\widetilde{P}_1, \dots, \widetilde{P}_M)$*, and every* $z \in \{0,1\}^*$*, we have* $\Pr[\langle \widetilde{\boldsymbol{P}}(w), V(z)\rangle(\mathsf{pp}, x) = 1] \leqslant \varepsilon(\lambda)$*, where the probability is taken over* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathrm{aux})$ *and the random coins of* V*.*

If $P_1, \dots, P_M$ *are computationally unbounded algorithms, then we replace all adversaries above with another computationally unbounded algorithm and obtain an* interactive proof *for* $\mathcal{R}$ *in the HMMP model.*

Our HMMP model definitions public verifiability, public coin, zero-knowledge, and non-interactivity are identical to the standard definitions. Note that all of our definitions naturally extend to *interactive proofs*: arguments where the prover is assumed to be computationally unbounded.

Definition 4.3 (HMMP Model Public Verifiability)**.** *A HMMP model interactive argument* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *is* publicly verifiable *if any party viewing the transcript* $T = \langle \boldsymbol{P}(w), V(z)\rangle(x)$ *can certify the validity of the interaction. Importantly, the interaction among provers* $\boldsymbol{P}$ *is not part of the transcript* T*.*

Definition 4.4 (HMMP Model Public Coin)**.** *A HMMP model interactive argument* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *is called* public coin *if all messages sent from* V *to* $\boldsymbol{P}$ *are chosen uniformly and independently at random, and independently of the prover's messages.*

Protocol 1: $\boldsymbol{P}_{\text{hmp}}$ Implementation

```
Parties: $M$ provers $\boldsymbol{P}_{\text{hmp}} = (P_1, \ldots, P_M)$.
Input  : $\mathsf{pp} \leftarrow \mathsf{Setup}_{\text{hmp}}(1^\lambda, \text{aux}, 1^M, 1^\tau)$, $(x, w) \in \mathcal{R}_{\mathcal{L}}$, partial transcript $\Pi$ with all messages sent
         between $\boldsymbol{P}_{\text{hmp}}$ and $V_{\text{hmp}}$ so far.
Internal State: Each $P_i$ has tuple $\mathcal{P} \in [M]^M$ of the current provers, initially set as
         $\mathcal{P} = (1, 2, \ldots, M)$.
Let $i \in \{0, 1, \ldots, r\}$ be the current round of interaction. foreach $j \in [M]$ do
    foreach $k \in [j, j + \tau - 1]$ do
        $P_{\pi(k)}$ computes and broadcasts $y_{j,k}^{(i)} = C_{i,j}(z)$ for $z = \mathsf{pp} \circ x \circ w \circ \Pi \circ y_1^{(i)} \circ \cdots \circ y_{j-1}^{(i)}$.
    if $y_{j,k}^{(i)}$ not all equal for $k \in [j, j + \tau - 1]$ then
        All provers set $y_j^{(i)} = \mathsf{Audit}(\mathsf{pp}, x, w, \Pi, j, \{y_{j,k}^{(i)}\}_k$.
    else All provers set $y_j^{(i)} = y_{j,j}^{(i)}$.
For all $j \in [M]$, $P_j$ uses $y_1^{(i)}, \ldots, y_M^{(i)}$ to define $y^{(i)} = C_i(\mathsf{pp} \circ x \circ w \circ \Pi)$ and broadcasts $y^{(i)}$ to $V_{\text{hmp}}$.
```

Definition 4.5 (HMMP Model Zero-Knowledge)**.** *A public-coin interactive HMMP model argument* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *for a relation ensemble* $\mathcal{R}$ *is said to have* γ-computational zero-knowledge *with respect to an auxiliary input if for every PPT interactive algorithm* V^* *there exists a PPT simultor* $\mathcal{S}$ *running in polynomial time with respect to its first input such that for every* $(x, w) \in \mathcal{R}$ *and any* $z \in \{0,1\}^*$*, we have* $\text{View}(\langle \boldsymbol{P}(w), V^*(z)\rangle(x)) \approx_\gamma S(x, z)$*, where all provers in* $\boldsymbol{P}$ *are honest. Here,* View *denotes the distribution of the transcript of the interaction between* $\boldsymbol{P}$ *and* V^* *and* $\approx_\gamma$ *denotes* γ*-computational indistinguishability. If the statistical distance between the above two distributions is negligible, then the argument has* statistical zero-knowledge*. If the simulator is allowed to abort with probability at most* $1/2$*, but the distribution of its output conditioned on no abort is identical to* $\text{View}(\langle \boldsymbol{P}(w), V^*(z)\rangle(x))$*, then the argument has* perfect zero-knowledge*.*

We formally define HMMP model witness-extended emulation as well.

Definition 4.6 (HMMP Model Witness-extended Emulation)**.** *Given a public-coin HMMP model interactive argument* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *and adversary* A_{hmp} *against the argument, let* $\mathsf{Rec}(\widetilde{\boldsymbol{P}}, \mathsf{pp}, x, st)$ *denote the message transcript between* $\widetilde{\boldsymbol{P}}$ *and* V *on shared input* x*, initial prover state* st*, and* pp *generated by* Setup*, where* $\widetilde{\boldsymbol{P}} = (\widetilde{P}_1, \ldots, \widetilde{P}_M)$ *are* M *corrupt HMMP model prover algorithms. Furthermore, let* $\mathsf{E}^{\mathsf{Rec}(\widetilde{\boldsymbol{P}}, \mathsf{pp}, x, st)}$ *denote a machine* E *with a transcript oracle for this interaction that can be rewound to any round and run again on fresh verifier randomness. The tuple* $(\mathsf{Setup}, \boldsymbol{P}, V)$ *has* witness-extended emulation *if for every deterministic polynomial-time* A_{hmp} *there exists an expected polynomial-time emulator* E *such that for all non-uniform polynomial-time adversaries* A *the following holds:*

$$\Pr\left[A(tr) = 1 \colon \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \text{aux}),\ (x, st) \leftarrow A(\mathsf{pp}), \\ tr \leftarrow \mathsf{Rec}(\widetilde{\boldsymbol{P}}, \mathsf{pp}, x, st) \end{array}\right] \approx \tag{2}$$

$$\Pr\left[\begin{array}{l} A(tr) = 1 \ \wedge \\ tr \text{ accepting} \implies (x, w) \in \mathcal{R} \end{array} : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \text{aux}),\ (x, st) \leftarrow A(\mathsf{pp}), \\ (tr, w) \leftarrow \mathsf{E}^{\mathsf{Rec}(\widetilde{\boldsymbol{P}}, \mathsf{pp}, x, st)}(\mathsf{pp}, x) \end{array}\right].$$

4.1 The HMMP Model Transformation

Given any interactive argument such that the prover has a suitably partitionable next-message function, we can transform this interactive argument to one that is secure in the HMMP model. This transformation preserves zero-knowledge and non-interactivity. Our construction makes explicit use of the partitioned function and employs a "permute and audit" strategy where provers are shuffled and must perform computations in audit windows to cross-check other provers. The properties of this transformation are given in Theorem 1.1.

4.1.1 Construction.

Let $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$ be an interactive argument where P_{arg} is suitably partitionable. Then we construct a HMMP model interactive argument $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$ with the desired properties.

The Setup Algorithm. The algorithm $\mathsf{Setup}_{\text{hmp}}$ is a modification of $\mathsf{Setup}_{\text{arg}}$ that additionally outputs a random permutation π on the set $[M]$. Suppose $\mathsf{Setup}_{\text{arg}}$ takes as input 1^λ for security parameter λ and additional input aux. Then we define $\mathsf{Setup}_{\text{hmp}}$ as follows: on input $1^\lambda, \text{aux}, 1^M, 1^\tau$ for security parameter λ, auxiliary input aux, number of provers M, and audit parameter τ, first sample $\mathsf{pp}_{\text{arg}} \leftarrow \mathsf{Setup}_{\text{arg}}(1^\lambda, \text{aux})$. Next sample uniformly random permutation $\pi \colon [M] \to [M]$. Finally output $\mathsf{pp} := (\mathsf{pp}_{\text{arg}}, \pi, M, \tau)$.

As stated previously, we include the random permutation π to shuffle the (potentially corrupt) provers to ensure soundness. We remark that π can also be the first message sent by the verifier V_{hmp} in the definition of the interaction, but we choose to include this in the setup phase.

The Multi-Prover Algorithm. We present $\boldsymbol{P}_{\text{hmp}}$ in Protocol 1 and describe it in detail. Suppose that Π_{arg} has $r+1$ rounds of interaction $\{0, 1, \ldots, r\}$. Suppose that C_i is the circuit representing the next message function of P_{arg} during round i of the interaction. Let $C_{i,1}, \ldots, C_{i,M}$ be a partition of C_i such that the inputs to $C_{i,j}$ for any $j \in [M]$ only depend on the input wires of C_i and the output wires of any $C_{i,k}$ for $k < j$. Given a permutation π and parameter τ, circuit $C_{i,j}$ is computed by all provers $P_{\pi(k)}$ for $k \in [j, j+\tau-1]_M$, where $[j, j+\tau-1]_M$ denotes the set $\{j, j+1, \ldots, j+\tau-1\}$ modulo M. Then each $P_{\pi(k)}$ broadcasts their computation of the circuit $C_{i,j}$, and if any of the τ messages differ then the sub-routine Audit is invoked.

The sub-routine Audit (presented in **??**) is a *global* audit of the current message being computed. That is, if function f_i is being computed and Audit is called, all M provers engage in a global audit of this computation. In this sub-routine, all provers compute the value $C_{i,j}$ and compare it to the values broadcast by the provers $P_{\pi(k)}$ for $k \in [j, j+\tau-1]_M$. Then for every k such that the value presented by $P_{\pi(k)}$ differs from the globally computed value, the index k is added to a set A, and then the sub-routine $\mathsf{Rebalance}$ (presented in **??**) is called.

The sub-routine $\mathsf{Rebalance}$ removes prover $P_{\pi(j)}$ from the protocol and replaces it with the most recent prover $P_{\pi(k)}$ for $k < j$ that was not removed from the protocol, for every $j \in A$. This ensure that over the course of audits if malicious parties submit incorrect values then they are removed from the protocol for future computations.

Our Audit and $\mathsf{Rebalance}$ sub-routines also allow us to handle aborting provers. Since all provers have the same inputs, if any prover $P_{\pi(i)}$ aborts during the execution of the protocol, the remaining provers simply treat the value that was to be computed by this prover as a null value (e.g., $\perp$) and proceed as normal. Then when this value is given, the provers engage in Audit and $\mathsf{Rebalance}$ to replace the prover that aborted.

The Verifier Algorithm. We present the algorithm V_{hmp} in Protocol 4. The algorithm is identical to V_{arg}, except that additionally V_{hmp} takes the majority of the prover messages it receives, and then uses this majority value as input to V_{arg}. It then forwards the message from V_{arg} to the broadcast channel, sending this message to all M provers, and accepts or rejects as V_{arg}. Additionally, in the case that Audit is called by $\boldsymbol{P}_{\text{hmp}}$, V_{hmp} receives updated prover tuples $\mathcal{P}_i$ for every $i \in [M]$, computes the majority of these tuples and updates its state $\mathcal{P}$ with this majority.

4.1.2 Correctness and Soundness

We show that the defined interactive argument $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$ is an argument in the HMMP model, as per Definition 4.1. To show correctness and soundness, we first show some key properties about the algorithm $\boldsymbol{P}_{\text{hmp}}$. Fix the number of provers $M \in \mathbb{N}$, the number of corrupted parties $t \in \mathbb{N}$ such that $t < M/2$, and the audit window size $\tau \in \mathbb{N}$ such that $\tau \leqslant t$. We first show that the algorithm Audit never evicts an honest prover.

Protocol 2: Audit sub-routine

```
   Input  : Public parameters pp, statement $x$, audit parameter $\tau$, partial transcript $\Pi$, current index $j$, claimed values $\{y^{(i)}_{j,k}\}_{k\in[j,j+\tau-1]_M}$.
   Output : Value $y^{(i)}_j$.
1  foreach $k \in [M]$ do
2      $P_k$ computes and broadcasts $\tilde{y}^{(i)}_k = C_{i,j}(\mathsf{pp} \circ x \circ w \circ \Pi \circ y^{(i)}_1 \circ \cdots \circ y^{(i)}_{j-1})$.
3  foreach $k \in [M]$ do
4      $P_k$ computes $y^{(i)}_j = \mathrm{majority}(\tilde{y}^{(i)}_1, \ldots, \tilde{y}^{(i)}_M)$.
5      $P_k$ computes and broadcasts $A_k = \{\ell \in [j, j+\tau-1]_M \colon y^{(i)}_\ell \neq y^{(i)}_j\}$.
6  foreach $k \in [M]$ do
7      $P_k$ computes $A = \mathrm{majority}(A_1, \ldots, A_M)$.
8  Run Rebalance($A$).
9  return $y^{(i)}_j$.
```

Protocol 3: Rebalance sub-routine

```
Input : Set $A \subset [M]$.
foreach $i \in A$ do
    foreach $j = i, i-1, \ldots, 1, M, \ldots, i+1$ do
        Find first $j$ such that $k = (i-j) \bmod M \notin A$.
    Remove party $P_{\pi(i)}$ from the protocol and set $P_{\pi(i)} := P_{\pi(k)}$.
    foreach $j \in [M]$ do
        $P_j$ updates internal state $\mathcal{P}_j$ by replacing $\pi(i)$ with $\pi(k)$.
        $P_j$ broadcasts $\mathcal{P}_j$ to $V_{\mathrm{hmp}}$.
```

Lemma 4.7. *Let* $\mathsf{pp}, x, \tau, \Pi, j, \{y^{(i)}_{j,k} \colon k \in [j, j+\tau-1]_M\}$ *be an input to* Audit*, and let* $\mathcal{H} \subseteq [M]$ *be the indices of the honest provers. Let* $A \subset [M]$ *be the set computed in Protocol 2 via* $\mathsf{Audit}(\mathsf{pp}, x, \tau, \Pi, j, \{y^{(i)}_{j,k}\}_k)$*. Then* $\mathcal{H} \cap \pi(A) = \emptyset$*, where* $\pi(A) := \{\pi(i) \colon i \in A\}$*.*

Proof. The proof follows directly from the honest majority assumption. That is, in Line 7 of Protocol 2, since there is an honest majority of provers, the set A contains all indices $k \in [j, j+\tau-1]_M$ such that $P_{\pi(k)}$ did not compute the correct value of $C_{i,j}$. Since any honest prover computes the correct value of $C_{i,j}$, this implies that the computed A only contains indices k such that $P_{\pi(k)}$ submitted an incorrect value, which implies that $\mathcal{H} \cap \pi(A)$. □

The next property we observe is that if every window of τ provers contains at least one honest party, then this property is invariant under Audit. That is, running Audit never introduces a window of τ dishonest provers.

Lemma 4.8. *Let* $\mathcal{H} \subseteq [M]$ *be the indices of the honest provers and fix* $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathrm{hmp}}(1^\lambda, \mathrm{aux}, 1^M, 1^\tau)$*. If for every* $j \in [M]$ *it holds that* $\mathcal{H} \cap \pi(\mathcal{J}) \neq \emptyset$ *for* $\mathcal{J} = [j, j+\tau-1]_M$*, then after executing* $\mathsf{Audit}(\mathsf{pp}, x, \tau, \Pi, j, \{y^{(i)}_{j,k}\}_k)$ *for any* $j \in [M]$*, it holds that* $\mathcal{H} \cap \pi(\mathcal{J}) \neq \emptyset$*.*

Proof. Note that Audit is only called during window $j \in [M]$ of the computation in round i if not all messages computed during this round by the window of τ provers is equal; thus without loss of generality, Audit is only called when there exists at least one dishonest prover. By assumption, there exists at least one honest prover in every window of τ provers. This implies that every honest prover is at most distance τ from another honest prover; that is, for every $j, k \in \mathcal{H}$, it holds that $|\pi(j) - \pi(k)| \leqslant \tau$. By Lemma 4.7, honest provers are never evicted from their windows during an audit, which implies that after any Audit it holds that $|\pi(j) - \pi(k)| \leqslant \tau$ for every $j, k \in \mathcal{H}$, which in turn applies that there exists at least one honest prover in every window of τ provers. □

Now given Lemmas 4.7 and 4.8, we recall and prove Proposition 2.1.

Proposition 2.1. *Let* Π_{arg} *be an argument with* δ_{arg} *correctness error and let* Π_{hmp} *be the argument obtained by applying Theorem 1.1. Let* $\mathcal{H} \subseteq [M]$ *be the indices of the honest provers and let* π *be a permutation over*

Protocol 4: V_{hmp} Implementation

```
Input    : $\mathsf{pp} \leftarrow \mathsf{Setup}_{\text{hmp}}(1^\lambda, \text{aux}, 1^M, 1^\tau)$, $x \in \{0,1\}^*$, partial transcript $\Pi$ with all messages sent
           between $\boldsymbol{P}_{\text{hmp}}$ and $V_{\text{hmp}}$ so far.
Output   : Accept or Reject.
Internal State: Tuple $\mathcal{P} \in [M]^M$ of the current provers, initially set as $\mathcal{P} = (1, 2, \dots, M)$.
if Audit and Rebalance have been invoked then
    Update $\mathcal{P} = \text{majority}(\mathcal{P}_1, \dots, \mathcal{P}_M)$, where $\mathcal{P}_i$ are broadcast from Rebalance.
Set $y = \text{majority}\{\{y^{(i)}\}_{i \in \mathcal{P}}\}$.
Compute $\alpha \leftarrow V_{\text{arg}}(\mathsf{pp}, x, (\Pi, y))$.
if $\alpha$ is accept or reject then return $\alpha$.
Broadcast $\alpha$ to $\boldsymbol{P}_{\text{hmp}}$.
```

$[M]$. For every $i \in [M]$, if $\mathcal{H} \cap \pi(\mathcal{J}) \neq \emptyset$ for $\mathcal{J} = \{i, i+1, \dots, i+\tau-1\}/(M\mathbb{Z})$ then the argument Π_{hmp} has correctness error δ_{arg}.

Proof. By Lemmas 4.7 and 4.8, we have that if every window of τ provers contains at least one honest party, then the output of $\boldsymbol{P}_{\text{hmp}}$ with $t < M/2$ corruptions is identical to the output of $\boldsymbol{P}_{\text{hmp}}$ with $t = 0$ corruptions. This shows the result. □

Finally, the following proposition follows directly from applying Lemma 2.2. This proposition states that the probability of sampling a bad permutation π (i.e., a permutation where no honest parties exist in some window of τ provers) is small (depending of the choice of τ).

Proposition 4.9. *Let $\mathcal{H} \subseteq [M]$ be the indices of the honest provers. Then $\Pr[\exists i \in [M] \colon \mathcal{H} \cap \pi(\mathcal{J}_i) = \emptyset] \leqslant M \cdot (t/M)^\tau$, where t is the number of corrupt provers and the probability is taken over uniformly random permutation $\pi \colon [M] \to [M]$.*

For completeness, we recall and prove Lemma 2.2.

Lemma 2.2. *Let $\tau, t, M \in \mathbb{N}$ be integers such that $\tau \leqslant t$ and $t < M$, and let $T \subset \{1, \dots, M\}$ be a subset of size t. For permutation π over $[M]$, let $S_{(i,j)} = \{S_{\ell,\pi} \colon \ell = (i + k \mod M) \ \forall k \in \{0, \dots, j-1\}\}$, where $S_{\ell,\pi} = \pi(\ell)$ for all $\ell \in \{1, \dots, M\}$. Then $\Pr\left[\exists i \in [M] \colon S_{(i,\tau)} \subseteq T\right] \leqslant M \cdot (t/M)^\tau$, where the probability is taken over uniformly random permutation π.*

Proof. Fix $i \in [M]$. Note that $S_{(i,\tau)} \subseteq T$ if and only if for every $j \in [0, \tau - 1]$, we have $S_{i+j \bmod M} \in T$. For a fixed j we have that

$$\Pr_\pi\left[S_{i+j \bmod M} \in T\right] = \Pr\left[\exists k \in T \colon \pi(i + j \bmod M) = k\right] \leqslant (t/M)\,.$$

Note that all events $S_{i+j \bmod M} \in T$ are independent for all j. Therefore we have

$$\Pr_\pi\left[S_{(i,\tau)} \subseteq T\right] = \prod_{j=0}^{\tau-1} \Pr_\pi\left[S_{i+j \bmod M}\right] \leqslant (t/M)^\tau\,.$$

Applying Union Bound over indices $i \in [M]$ yields the desired probability. □

Correctness. By Proposition 2.1, as long as the permutation sampled by $\mathsf{Setup}_{\text{hmp}}$ is not a bad permutation, then the interaction between $\boldsymbol{P}_{\text{hmp}}$ and V_{hmp} is identical to the interaction between P_{arg} and V_{arg}. In particular, if Π_{arg} has δ_{arg}-correctness, then Π_{hmp} has δ_{arg}-correctness conditioned on the permutation being good. By Proposition 4.9, the probability that a good permutation is sampled is at least $1 - M \cdot (t/M)^\tau$. Let $p := p(M, t, \tau) = 1 - M \cdot (t/M)^\tau$. Then the probability that $\langle \boldsymbol{P}_{\text{hmp}}(w), V_{\text{hmp}} \rangle(\mathsf{pp}, x) = 1$ is at least $(1 - \delta_{\text{arg}}) \cdot p$. This gives our stated correctness error $\delta_{\text{hmp}} = 1 - (1 - \delta_{\text{arg}}) \cdot (1 - M \cdot (t/M)^\tau)$.

Soundnesss. As discussed in Section 2.1, the soundness of our HMMP model protocol directly reduces to the soundness of the underlying interactive argument. This is because by our definition, we require soundness to hold with respect to M corrupt provers $\widetilde{P}_1, \ldots, \widetilde{P}_M$. Notice that any set of M corrupt provers that can break the soundness of our HMMP model argument can directly be used to break the soundness of the underlying argument with identical probability. The reduction is simple: first run the corrupt provers and wait for them to broadcast their answers. Take the M values broadcast by the provers, take the majority, and directly forward this to the verifier for the argument. By our construction of V_{hmp}, it is easy to see that if $\widetilde{P}_1, \ldots, \widetilde{P}_M$ break soundness with V_{hmp}, then our constructed adversary breaks soundness with V_{arg}.

Zero-Knowledge. Given a zero-knowledge argument $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$, we modify our algorithm $\boldsymbol{P}_{\text{hmp}}$ to obtain zero-knowledge for the transformed HMMP model argument. For this transformation, we first observe that the circuit representing the next message function of P_{arg} during round i can be abstracted as two circuits C_i and D_i such that for any input $x \in \{0,1\}^*$: (1) $C_i(x)$ computes the desired functionality; and (2) $D_i(C_i(x); r)$ gives the zero-knowledge property, given uniformly random string $r \xleftarrow{\$} \{0,1\}^*$. Intuitively, we achieve zero-knowledge for $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$ by additionally requiring that

1. All provers P_i engage in a protocol to generate a shared (almost) uniformly random string $r \xleftarrow{\$} \{0,1\}^*$ (or simply use a randomness beacon); and
2. Compute $y^{(i)} = D_i\big(y'; r)\big)$ in the final step of $\boldsymbol{P}_{\text{hmp}}$ in Protocol 1, where y' is the derived output of $C_i(x)$ given all the values $y_1^{(i)}, \ldots, y_M^{(i)}$.

Thus with high probability over the choice of permutation π, so long as the honest majority of provers all receive the same random string r, the verifier V_{hmp} receives the correct zero-knowledge message. The random string r can be generated in every round either by assuming access to a randomness beacon [Rab83, SJK+17] that broadcasts random bits to all provers, or by any MPC protocol for collective coin tossing with an honest majority of provers [BOGW88, RBO89]. In the MPC case, since we are in the honest majority setting, adversaries can only bias the output of the coins by a negligible amount [BOGW88, RBO89]. Further, in the case of abort, parties run an Audit with the set A containing the index of the aborting party, then re-run the randomness generation protocol.

Non-Interactive Arguments. To obtain non-interactivity via the Fiat-Shamir transform [FS87], we assume all provers in $\boldsymbol{P}_{\text{hmp}}$ have access to the same random oracle H. Let $\mathsf{pp}, x, w, \tau, \Pi$ be the input to $\boldsymbol{P}_{\text{hmp}}$ during any round of the computation, where the public parameters are generated as $\mathsf{pp} \leftarrow \mathsf{Setup}_{\text{hmp}}(1^\lambda, \text{aux}, 1^M, 1^\tau)$, $(x, w) \in \mathcal{R}_{\mathcal{L}}$, and Π is a partial transcript containing all prior messages sent by $\boldsymbol{P}_{\text{hmp}}$ and messages generated by H (i.e., the simulated messages of V_{arg}). Then the provers $\boldsymbol{P}_{\text{hmp}}$ generate the next message α of V_{hmp} as follows:

1. upon the broadcast of all $y^{(i)}$ for $i \in [M]$, every prover P_j computes $y = \text{majority}(y^{(1)}, \ldots, y^{(M)})$; and
2. every P_j computes $\alpha \leftarrow H(\mathsf{pp}, x, (\Pi, y))$ and uses this α as the verifier message.

It is clear that this transformation is identical to the case of transforming $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$ into a non-interactive argument. Note that if any malicious prover of the protocol $\boldsymbol{P}_{\text{hmp}}$ deviates from defining α as described above, with high probability over the choice of random oracle H these malicious provers are caught in the proceeding rounds of the protocol.

Witness-Extended Emulation. We show that the transformation of Theorem 1.1 also preserves witness-extended emulation. We capture this via the following proposition.

Proposition 4.10. *If $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$ is an interactive argument with witness-extended emulation, then $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$ obtained by applying Theorem 1.1 has witness-extended emulation.*

Proof. Suppose that $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$ is an interactive argument with witness-extended emulation. Let $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$ be the HMMP model interactive argument obtained by applying Theorem 1.1. We argue that this HMMP model argument has witness-extended emulation (as per Definition 4.6). We show this by constructing an emulator E_{hmp} such that Eq. (2) holds. Suppose E_{hmp} has oracle access to $\mathsf{Rec}(\widetilde{\boldsymbol{P}}_{\text{hmp}}, \mathsf{pp}, x, st)$ where $\mathsf{pp} \leftarrow \mathsf{Setup}_{\text{hmp}}(1^\lambda, \text{aux})$ and $(x, st) \leftarrow A(\mathsf{pp})$. Then given pp, x as input, E_{hmp} does the following

1. Parse pp as $(\mathsf{pp}', \pi, M, \tau)$, where $\mathsf{pp}' \leftarrow \mathsf{Setup}_{\text{arg}}(1^\lambda, \text{aux})$.
2. E_{hmp} initializes an empty transcript tr.
3. Let E be the emulator for argument $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$. E_{hmp} runs $\mathsf{E}(\mathsf{pp}', x)$ and simulates oracle access to $\mathsf{Rec}(P^*, \mathsf{pp}', x, st)$, where P^* is some arbitrary prover strategy. In particular, E_{hmp} simulates this oracle access by directly forwarding the transcript derived from its oracle $\mathsf{Rec}(\widetilde{\boldsymbol{P}}_{\text{hmp}}, \mathsf{pp}, x, st)$ with the following modifications:
    - Whenever $\widetilde{\boldsymbol{P}}_{\text{hmp}}$ sends M messages to V_{hmp}, E_{hmp} replaces these M messages in the transcript with the majority of these messages.
    - Any message from V_{hmp} that is broadcast to the M provers of $\widetilde{\boldsymbol{P}}_{\text{hmp}}$ is instead replaced with a single message.
    - Any messages generated due to a call to Audit are not given as part of the interaction.

   Note that this simulation is otherwise unchanged; whenever E rewinds to round i, then E_{hmp} similarly rewinds to round i. Note that E_{hmp} stores the unmodified transcript in tr.
4. Upon receiving $(tr', w) \leftarrow \mathsf{E}(\mathsf{pp}, x)$, output (tr, w).

We now argue that Eq. (2) holds. In particular, we show that adversary A on input tr cannot distinguish between whether tr was output by $\mathsf{Rec}(\widetilde{\boldsymbol{P}}_{\text{hmp}}, \mathsf{pp}, x, st)$ or E_{hmp}. Further, if tr output by E_{hmp} is accepting then $(x, w) \in \mathcal{R}$. First note that A cannot distinguish whether tr is generated by Rec or E_{hmp} since the transcript generated by E_{hmp} is identical to a transcript generated by Rec, except for that E_{hmp} might rewind and resample *verifier* randomness. Since the verifier is public-coin, as long as E_{hmp} samples new coins the same way the verifier does, the transcripts are indistinguishable. Next suppose that tr is accepting. Note that a valid arbitrary prover strategy P^* is to simulate $\widetilde{\boldsymbol{P}}_{\text{hmp}}$ honestly, send the majority of M messages received from $\widetilde{\boldsymbol{P}}_{\text{hmp}}$ to V_{arg}. This implies that if tr is accepting then tr' is also accepting. Further, by Theorem 1.1, if $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$ is an interactive argument for $\mathcal{R}$, then so is $(\mathsf{Setup}_{\text{hmp}}, \boldsymbol{P}_{\text{hmp}}, V_{\text{hmp}})$. By witness-extended emulation of $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$, since tr' is accepting we have that $(x, w) \in \mathcal{R}$; this implies that tr is accepting implies that $(x, w) \in \mathcal{R}$. Finally, we establish indistinguishably by observing that if Eq. (2) does not hold, then we immediately have a prover P^* such that Eq. (1) does not hold, breaking witness-extended emulation of $(\mathsf{Setup}_{\text{arg}}, P_{\text{arg}}, V_{\text{arg}})$. □

5 Efficient Interactive Arguments for NP in the HMMP Model

We dedicate this section to proving Theorem 1.2, which we recall here.

Theorem 1.2. *Let $M, \tau, t, \lambda \in \mathbb{N}$ be parameters such that $t < M/2$ and $\tau \leqslant t$. For every* NP *relation with statements of size* $\text{poly}(\lambda)$ *and witnesses of size N, assuming the hardness of the discrete log problem for prime order groups there exists a HMMP model interactive argument Π_{hmp} with M provers and the following efficiency properties:*

- *the individual prover computation is $\widetilde{O}_\lambda(\tau \cdot (N/M))$;*
- *the communication complexity between provers is $\widetilde{O}_\lambda(\tau \cdot (N/M))$ bits;*

- *the verifier computation is $O_\lambda(M + N)$; and*
- *the verifier receives $O_\lambda(M \cdot \log(N))$ bits from all provers and sends $O_\lambda(\log(N))$ bits to all provers.*

To show Theorem 1.2, it suffices to prove Propositions 5.1, 2.3 and 2.5. Then by our discussion in the overview, we obtain Theorem 1.2.

5.1 HMMP Model Circuit SAT

In Section 2.2, we outlined a HMMP model protocol for circuit satisfiability. We give the formal protocol here and add some discussion.

One key condition is the need for a circuit C to be partitionable into $C_1, \ldots, C_M$ circuits each of size N/M such that for any $j \in [M]$, partition C_j depends only on wires from C_i for any $i \leqslant j$ and the input x. We remark that a suitable topological sorting will define a sequence of individual gates with this property, after which the partitions $C_1, \ldots, C_M$ can be suitably defined.

Given the above discussion, the discussion in Section 2.2, and Protocol 5, Proposition 2.3 follows.

Protocol 5: HMMP Model Circuit SAT

```
Parties : $M$ provers $\boldsymbol{P}_{\text{hmp}} = (P_1, \ldots, P_M)$.
Input   : An arithmetic circuit $C\colon \{0,1\}^n \to \mathbb{F}$ of size $N$, an input to the circuit $x$, an output $y$, and
          a partition of $C$ into $C_1, \ldots, C_M$.
Output  : A wire transcript $w$ certifying that $C(x) = y$.
Each prover sets wire transcript $w_i = \emptyset$ for $i \in [M]$.
foreach $i \in [M]$ do
    foreach $j \in [i, i + \tau - 1]_M$ do
        $P_{\pi(j)}$ uses $w_k$ for all $k < i$ and $x$ to compute and broadcast a wire transcript $w_{i,j}$ for $C_i$.
    if all $w_{i,j}$ are not equal for $j \in [i, i + \tau - 1]_M$ then
        All provers set $w_i = \mathsf{Audit}(C, x, \{w_k\}_{k<i}, \{w_{i,j}\}_{j \in [i, i+\tau-1]_M})$.
    else
        All provers set $w_i := w_{i,i}$.
Each prover defines $w = (w_1, \ldots, w_M)$.
return $w$.
```

5.2 HMMP Model Sum-Check

We give the formal description of our HMMP model sum-check protocol and prove the following proposition.

Proposition 5.1. *There exists a HMMP model protocol for the sum-check protocol with the following properties: for a finite field $\mathbb{F}$, an n-variate polynomial $f \in \mathbb{F}[X_1, \ldots, X_n]$ of individual degree at most d, a subset $H \subset \mathbb{F}$, and parameters τ, M such that $M \leqslant |H|^n$,*

- *the individual prover computation is $O(\tau \cdot ((n \cdot |H|^n)/M + |H|))$ polynomial evaluations and $O(\tau \cdot ((n \cdot |H|^n)/M + d \cdot |H|))$ field additions; and*
- *each prover broadcasts $O(\tau \cdot d \cdot (n + |H|))$ field elements.*

Note that setting $H = \{0, 1\}$ and $d = \Theta(1)$ in Proposition 5.1 directly yields Corollary 2.4.

```
Protocol 6: Sum-Check Protocol
Parties          : Prover $P$ and verifier $V$.
Prover Input     : A multi-variate polynomial $g \in \mathbb{F}[X_1, \dots, X_n]$ of individual degree at most $d$, a
                   subset $H \subset \mathbb{F}$, and a value $y \in \mathbb{F}$.
Verifier Input   : Individual degree upper bound $d$, a subset $H \subset \mathbb{F}$, and value $y \in \mathbb{F}$.
Verifier Output  : Accept or Reject.
Oracle           : The verifier $V$ has oracle access to evaluations of the polynomial $g$.
$P$ computes and sends to $V$ the univariate polynomial $g_1(X_1) := \sum_{x \in H^{n-1}} g(X_1, \boldsymbol{x})$.
$V$ checks that $g_1$ is univariate with degree at most $d$ and that $y = \sum_{x \in H} g_1(x)$, aborting with Reject
  if not.
$V$ samples and sends $r_1 \xleftarrow{\$} \mathbb{F}$ to $P$.
foreach $1 < j < n$ do
    $P$ computes and sends to $V$ the univariate polynomial $g_j(X_i) := \sum_{x \in H^{n-j}} g(r, X_j, x)$, where
      $r := (r_1, \dots, r_{j-1})$.
    $V$ checks that $g_j$ is univariate with degree at most $d$ and that $g_{j-1}(r_{j-1}) = \sum_{x \in H} g_j(x)$, aborting
      with Reject if not.
    $V$ samples and sends $r_j \xleftarrow{\$} \mathbb{F}$ to $P$.
$P$ computes and sends to $V$ the univariate polynomial $g_n(X_n) := g(r_1, \dots, r_{n-1}, X_n)$.
$V$ checks that $g_n$ is univariate with degree at most $d$ and that $g_{n-1}(r_{n-1}) = \sum_{x \in H} g_n(x)$, aborting
  with Reject if not.
$V$ samples $r_n \xleftarrow{\$} \mathbb{F}$ and queries the oracle at $g(r_1, \dots, r_n)$. $V$ checks that $g_n(r_n) = g(r_1, \dots, r_n)$,
  aborting with Reject if not.
$V$ outputs Accept.
```

5.2.1 Single-Prover Sum-Check Protocol

We first recall the single-prover sum-check protocol. We present it in Protocol 6, and give an overview here. For a finite field $\mathbb{F}$, polynomial $g \in \mathbb{F}[X_1, \dots, X_n]$ of individual degree at most d, and subset $H \subset \mathbb{F}$, the sum-check protocol [LFKN92] is an interactive proof certifying that $\sum_{x \in H^n} g(x) = y$ for some $y \in \mathbb{F}$. The protocol assumes that the verifier has oracle access to evaluations of the polynomial f and proceeds in n rounds: in the first round, the prover sends y and the polynomial $g_1(X_1) := \sum_{x \in H^{n-1}} g(X_1, x)$. The verifier checks that (1) g_1 has degree at most d; and (2) $y = \sum_{b \in H} g_1(b)$, rejecting if either do not hold. The verifier then samples $r_1 \in \mathbb{F}$ uniformly at random and sends it to the prover, and additionally sets $y' = g_1(r_1)$. Then, for every $j = 2, \dots, n$

- the prover computes and sends polynomial $g_j(X_j) := \sum_{x \in H^{n-j}} g(r, X_j, x)$ to the verifier, where $r = (r_1, \dots, r_{j-1})$;
- the verifier checks that g_j has degree at most d and $y' = \sum_{b \in H} g_j(b)$, rejecting if either do not hold; and
- the verifier samples $r_j \in \mathbb{F}$ uniformly at random, sets $y' = g_j(r_j)$, and sends r_j to the prover. If $j = n$, then the verifier instead queries $g(r_1, \dots, r_n)$ and checks if it is equal to $g_n(r_n)$, rejecting if not.

In the sum-check protocol, generating the polynomial g_j for $j = 1, \dots, n$ requires evaluating the polynomial g at $|H|^{n-j}$ points (keeping the variable X_j free) and computing the same number of polynomial additions over $\mathbb{F}$.

5.2.2 HMMP Model Sum-Check Protocol

We outline how to obtain Proposition 5.1 given the single-prover sum-check protocol and present our HMMP model sum-check in Protocol 7. Since g_j is a polynomial and is a linear function, we can easily sub-divide the computation between multiple parties, then reconstruct the complete polynomial via addition.

Suppose we are in round j of the computation and for simplicity assume that $|H|^{n-j}/M$ is an integer. Partition the set H^{n-j} into M subsets $H'_1, \ldots, H'_M$ each of size $|H|^{n-j}/M$ Then the computation of g_j is divided as follows. For $i = 1, \ldots, M$, audit window $P_{\pi(i)}, P_{\pi(i+1)}, \ldots, P_{\pi(i+\tau-1)}$ computes the polynomial $g_{j,i}(X_j) := \sum_{x \in H'_i} g(r_1, \ldots, r_{j-1}, X_j, x)$. The audit window then broadcasts $g_{j,i}(X_j)$ and invokes a global audit and re-balance as necessary. At the end of this round of computation, each party obtains $g_j(X_j) := \sum_{i=1}^{M} g_{j,i}(X_j)$ and sends this polynomial to the verifier.

Thus during this round j of the computation, each audit window performs $O(|H|^{n-j}/M)$ evaluations of g and $O(|H|^{n-j}/M+M)$ additions over over $\mathbb{F}$, yielding $O(\tau \cdot (|H|^{n-j}/M))$ evaluations and $O(\tau \cdot (|H|^{n-j}/M+M))$ additions for each prover. Moreover, each prover broadcasts $O(d \cdot \tau)$ field elements since each $g_{j,i}$ is a univariate polynomial of degree at most d. As before with the circuit satisfiability algorithm, if $|H|^{n-j}/M$ is not an integer (but is at least 1), we can let $k = \lfloor |H|^{n-j}/M \rfloor$ and partition the computation of g_j into at most $2M$ polynomials $g_{j,1}, \ldots, g_{j,2M}$ by partitioning H^{n-j} into at most $2M$ subsets $H'_1, \ldots, H'_{2M}$ each of size at most k. Then each audit window computes at most 2 polynomials in round j, yielding the same asymptotic complexity.

Things are slightly more complicated if $M > |H|^{n-j}$. Let j^* be the index such that $|H|^{n-j^*} < M \leqslant |H|^{n-j^*+1}$. From this same inequality, we know that $|H|^{n-j^*+1} < |H| \cdot M$. Moreover, it holds that $\sum_{j=j^*}^{n} |H|^{n-j} = (|H|^{n-j^*+1} - 1)/(|H| - 1) < |H| \cdot M$. Thus we can sub-divide the computation of the final $n - j^*$ polynomials with only a $O(|H|)$ overhead for each prover as follows. In a round-robin fashion, the first $|H|^{n-j^*}$ audit windows compute the polynomial $g_{j^*,i}$ for $i = 1, \ldots, |H|^{n-j^*}$, followed by the next $|H|^{n-j^*-1}$ audit windows each computing the polynomial $g_{j^*+1,i}$ for $i = 1, \ldots, |H|^{n-j^*-1}$, and so on until the final round. Each audit window participates in the round-robin computation at most $O(|H|)$ times, which gives our $O(|H|)$ overhead and $O(\tau \cdot |H|)$ evaluations and $O(\tau \cdot |H| \cdot d)$ field additions in total over all rounds $j = j^*, \ldots, n$.

As before, let $j^* \in \{1, \ldots, n\}$ be the round such that $|H|^{n-j^*} < M \leqslant |H|^{n-j^*+1}$. By our previous discussion, the total computation done by each prover in all rounds $j = j^*, j^* + 1, \ldots, n$ is $O(\tau \cdot |H|)$ evaluations and $O(\tau \cdot |H| \cdot d)$ field additions. Moreover, for any round $j = 1, \ldots, j^* - 1$, each prover performs $O(\tau \cdot (|H|^{n-j}/M))$ evaluations and $O(d \cdot \tau \cdot (|H|^{n-j}/M + M))$ field additions. This gives $O(n \cdot \tau \cdot (|H|^n/M))$ evaluations and $O(n \cdot \tau \cdot (|H|^n/M + |H|))$ field additions over all rounds $j = 1, \ldots, j^* - 1$. Thus we have obtained Proposition 5.1. desired.

5.3 HMMP Model Polynomial Commitment Schemes

We formally show Proposition 2.5 in this section. For our purposes, we utilized the Bulletproofs-based Block et al. [BHR+20] polynomial commitment scheme tailored for multi-linear polynomials.

5.3.1 Polynomial Commitment Scheme Preliminaries

Much of this section is borrowed from Block et al. [BHR+20]. We let $\mathbb{F} := \mathbb{F}_p$ denote a finite field of prime order p. When clear from context we omit p above. For finite cyclic group $\mathbb{G}$, we assume that $\mathbb{G}$ is a multiplicative group. For $n \in \mathbb{N}$ and bit-string $b = (b_n, \ldots, b_1) \in \{0,1\}^n$, we assume that b_n is the most significant bit and b_1 is the least significant bit. We let "$\circ$" denote the string concatenation operator; that is, for $b \in \{0,1\}^n$ and $c \in \{0,1\}$, we have $b \circ c := (b_n, \ldots, b_1, c) \in \{0,1\}^{n+1}$. Often, we index a vector and/or sequence using a binary string. That is, for $N = 2^n$ and $Y \in \mathbb{F}^N$, for $b \in \{0,1\}^n$ we have that Y_b is the b-th element of Y, where we interpret b naturally as an integer in the range $\{0, 1, \ldots, N-1\}$.

Discrete-log Assumption. Let GGen be a randomized algorithm that on input 1^λ for security parameter λ returns $(\mathbb{G}, p, g)$ such that $\mathbb{G}$ is the description of a finite cyclic group of prime order p, where p is a λ-bit prime, and g is a generator of $\mathbb{G}$.

Assumption 5.2 (Discrete-log Assumption)**.** *The* discrete-log assumption *holds for* GGen *if for all PPT adversaries A there exists a negligible function μ such that*

$$\Pr\left[\alpha' = \alpha \;:\; (\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda),\ \alpha \xleftarrow{\$} \mathbb{Z}_p,\ \alpha' \leftarrow A(\mathbb{G}, g, g^\alpha)\right] \leqslant \mu(\lambda).$$

We use the following variant of the discrete-log assumption, which is equivalent to the assumption above.

Assumption 5.3 (Discrete-log Relation Assumption [BCC+16])**.** *The* discrete-log relation assumption *holds for* GGen *if for all PPT adversaries A and for all $n \geqslant 2$ there exists a negligible function μ such that*

$$\Pr\left[\exists \alpha_i \neq 0 \ \wedge \ \prod_{i=1}^{n} g_i^{\alpha_1} = 1 \ : \ \begin{array}{l} (\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda), \ g_1, \dots, g_n \xleftarrow{\$} \mathbb{G}, \\ (\alpha_1, \dots, \alpha_n) \in \mathbb{Z}_p^n \leftarrow A(\mathbb{G}, g_1, \dots, g_n). \end{array}\right] \leqslant \mu(\lambda).$$

For α_i not all equal to 0, we say that $\prod_i g_i^{\alpha_i}$ is a non-trivial discrete-log relation between group elements $g_1, \dots, g_n$. The discrete-log relation assumption states that any PPT adversary cannot find a non-trivial relation between randomly chosen group elements.

Multi-linear Polynomials. An n-variate polynomial $f \in \mathbb{F}[X_1, \dots, X_n]$ is a *multi-linear* polynomial if every variable has individual degree at most 1. Any multi-linear polynomial is uniquely represented by its evaluations over the Boolean hypercube. We capture this in Fact 5.4.

Fact 5.4. *For n-variate multi-linear polynomial f, for any $\zeta \in \mathbb{F}^n$ we have*

$$f(\zeta) = \sum_{b \in \{0,1\}^n} f(b) \cdot \beta(\zeta, b), \qquad \beta(\zeta, b) := \prod_{i=1}^{n} \zeta_i \cdot b_i + (1 - \zeta_i) \cdot (1 - b_i). \tag{3}$$

Note that any sequence $Y \in \mathbb{F}^N$ for $N = 2^n$ uniquely defines a n-variate multi-linear polynomial Eq. (3). For sequence $Y \in \mathbb{F}^N$ and evaluation point $\zeta \in \mathbb{F}^n$, we define $\mathsf{MLE}(Y, \zeta) := \sum_b Y_b \cdot \beta(\zeta, b)$.

Definition 5.5 ([BHR+20])**.** *A tuple of protocols* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ *is a* multi-linear polynomial commitment scheme *if it satisfies the following*

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ *takes as input the unary representations of security parameter $\lambda \in \mathbb{N}$ and size parameter $N = 2^n$ corresponding to $n \in \mathbb{N}$, and produces public parameter* pp. *We allow* pp *to contain the description of the field $\mathbb{F}$ over which the multi-linear polynomials will be defined.*

2. $(C; d) \leftarrow \mathsf{Com}(\mathsf{pp}, Y)$ *takes as input public parameter* pp *and sequence $Y = (Y_b)_{b \in \{0,1\}^n} \in \mathbb{F}^N$ that defines the multi-linear polynomial to be committed, and outputs public commitment C and secret decommitment d.*

3. $b \leftarrow \mathsf{Open}(\mathsf{pp}, C, Y, d)$ *takes as input* pp*, a commitment C, sequence committed Y and a decommitment d and returns a decision bit $b \in \{0, 1\}$.*

4. $\mathsf{Eval}(\mathsf{pp}, C, \zeta, \gamma; Y, d)$ *is a public-coin interactive protocol between a prover P and a verifier V with common inputs—public parameter* pp*, commitment C, evaluation point $\zeta \in \mathbb{F}^n$ and claimed evaluation $\gamma \in \mathbb{F}$, and prover has secret inputs Y and d. The prover then engages with the verifier in an interactive argument system for the relation*

$$\mathcal{R}_{\mathsf{mle}}(\mathsf{pp}) = \{(C, \zeta, \gamma; Y, d) : \mathsf{Open}(\mathsf{pp}, C, Y, d) = 1 \wedge \gamma = \mathsf{MLE}(Y, \zeta)\}. \tag{4}$$

The output of V is the output of Eval *protocol.*

We require the following three properties.

1. *Computational Binding. For all PPT adversaries A and $n \in \mathbb{N}$*

$$\Pr\left[b_0 = b_1 \neq 0 \wedge \mathcal{Y}_0 \neq \mathcal{Y}_1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ (C, \mathcal{Y}_0, \mathcal{Y}_1, d_0, d_1) \leftarrow A(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, C, Y^{(0)}, d_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, C, Y^{(1)}, d_1) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

2. *Perfect Correctness. For all $n, \lambda \in \mathbb{N}$ and all $Y \in \mathbb{F}^N$ and $\zeta \in \mathbb{F}^n$,*

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, C, \mathcal{Z}, \gamma; Y, d) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N), \\ (C; d) \leftarrow \mathsf{Com}(\mathsf{pp}, Y),\ \gamma = \mathsf{MLE}(Y, \zeta) \end{array}\right] = 1 \ .$$

3. *Witness-extended Emulation. We say that the polynomial commitment scheme has witness-extended emulation if* Eval *has a witness-extended emulation as an interactive argument for the relation ensemble* $\{\mathcal{R}_{\mathsf{mle}}(\mathsf{pp})\}_{\mathsf{pp}}$ *(Eq. (4)) except with negligible probability over the coins of* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$.

5.3.2 Multi-Linear Polynomial Commitment Scheme in the HMMP Model

We dedicate this section to proving Proposition 2.5. Since the prover of a polynomial commitment scheme is responsible for computing three functionalities—the commitment Com, the opening Open, and the evaluation Eval—we shall define the above functions for each of these phases for simplicity. We first recall the multi-linear polynomial commitment scheme of Block et al. [BHR+20], which we refer to as the BHRRS polynomial commitment scheme.

BHRRS Polynomial Commitment Scheme. We recall the BHRRS polynomial commitment scheme. Note that the BHRRS polynomial commitment scheme focuses on *space-efficiency* of the prover; that is, the prover uses space which is poly-logarithmic in the description size of the multi-linear polynomial. They achieve this space-efficiency in the random oracle model by assuming multi-pass streaming access to the description of the multi-linear polynomial that is being committed. For simplicity, we present the BHRRS polynomial commitment scheme without its space-efficient implementation.

We now specify the BHRRS polynomial commitment scheme.

1. $\mathsf{Setup}(1^\lambda, 1^N)$: on input security parameter 1^λ and size parameter 1^N for $N = 2^n$, where $n \in \mathbb{N}$, sample $(\mathbb{G}, p, \boldsymbol{g} \in \mathbb{G}^N)$, set $\mathbb{F} := \mathbb{F}_P$, and return $\mathsf{pp} := (\mathbb{G}, \mathbb{F}, N, p, \boldsymbol{g})$. Here, $\boldsymbol{g}$ is a vector of N distinct generators of $\mathbb{G}$.

2. $\mathsf{Com}(\mathsf{pp}, Y)$: return commitment $C = \prod_{b \in \{0,1\}^n} \boldsymbol{g}_b^{Y_b}$.

3. $\mathsf{Open}(\mathsf{pp}, C, Y)$: return 1 if and only if $C = \mathsf{Com}(\mathsf{pp}, Y)$.

4. $\mathsf{Eval}(\mathsf{pp}, C, \zeta, \gamma; Y)$ is an interactive protocol $\langle P, V \rangle$ which begins with V sending $g_V \xleftarrow{\$} \mathbb{G}$. Then both P and V compute $C_\gamma = C \cdot g_V^\gamma$ to bind claimed evaluation γ. P and V then engage in EvalReduce on input $(C_\gamma, Z, \boldsymbol{g}, g_V, \gamma; Y)$ where P proves knowledge of Y such that

$$C_\gamma = \mathsf{Com}(\mathsf{pp}, Y) \cdot g_V^\gamma \ \wedge \ \langle Y, Z \rangle = \gamma,$$

where $\zeta \in \mathbb{F}^n$ and $Z := \left(z_b = \beta(\zeta, b)\right)_{b \in \{0,1\}^n}$ for function β defined in Eq. (3). The protocol is presented in Protocol 8

BHRRS Polynomial Commitment Scheme in the HMMP Model. We specify how to construct each function Com, Open, and Eval in our HMMP model. Excluding the Open function, the specification and analysis of Com and Eval directly yield Proposition 2.5.

HMMP Model Com. The algorithm Com receives $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathrm{hmp}}(1^\lambda, 1^N, 1^M, 1^\tau)$ as input along with sequence of evaluations $Y \in \mathbb{F}^N$ defining a multi-linear polynomial. Here $\mathsf{pp} = (\mathsf{pp}', \pi, M, \tau)$ where $\mathsf{pp}' \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^N)$ for Setup algorithm of the BHRRS polynomial commitment scheme. By assumption $N = 2^n$ and $M = 2^m$ for $m \leqslant n$.

Let $g_0, \ldots, g_{N-1}$ be generators of a prime-order group $\mathbb{G}$. Then the commitment to f is $C = \prod_{i=0}^{N-1} g_i^{f(i)}$, where i is interpreted as an element in $\{0,1\}^n$ in the natural way when evaluating $f(i)$. To implement this commitment in the HMMP model, first partition the set $\{0, 1, \ldots, N-1\}$ into M subsets $S_1, \ldots, S_M$ each of size N/M. Then for every $i \in \{1, \ldots, M\}$, the audit window $P_{\pi(i)}, \ldots, P_{\pi(i+\tau-1)}$ computes the partial commitment $C_i = \prod_{j \in S_i} g_j^{f(j)}$, and broadcasts C_i, invoking a global audit and re-balance as necessary. After the computation of C_M, each party combines all commitments into the final commitment $C = \prod_i C_i$ and broadcast this to the verifier. Here, each prover performs $O(\tau \cdot (N/M))$ group multiplications and exponents, with at most a 2 fold increase if N/M is not an integer, and broadcasts $O(\tau)$ group elements.

HMMP Model Open. Each prover in the Open algorithm must send its entire witness Y. In particular, this is identical to the single-prover case, with the modification that V now needs to take majority of all these vectors.

HMMP Model Eval. To produce proofs of evaluations, the BHRRS polynomial commitment scheme utilizes the Bulletproofs inner-product argument in conjunction with a Pedersen commitment. The inner-product argument certifies that (1) $\gamma = \langle X, Y \rangle$, where $X \in \mathbb{F}^N$ is a public vector and $Y \in \mathbb{F}^N$ is a private vector held by the prover; and (2) the vector Y used to compute $\langle X, Y \rangle$ is consistent with some Pedersen commitment C. The inner-product argument employs a "half-and-fold" technique to reduce the claim $\gamma = \langle X, Y \rangle$ to a claim $\gamma' = \langle X', Y' \rangle$, where $X', Y' \in \mathbb{F}^{N/2}$.

We outline the reduction from size N to size $N/2$ vectors. In a bit more detail, let C be a Pedersen commitment to the vector Y using generators $G = (g_0, \ldots, g_{N-1})$ and suppose γ is the claimed inner-product $\langle X, Y \rangle$. The verifier first samples and sends a random group generator g, and both the prover and verifier update the commitment C to $C \cdot g^{\gamma}$. Let $X_o, X_e \in \mathbb{F}^{N/2}$, $Y_o, Y_e \in \mathbb{F}^{N/2}$, and $G_o, G_e \in \mathbb{G}^N$ denote the odd and even halves of the vectors X, Y, and G, respectively. That is, $X_o := (X_{2i})_{i \in [0, N-1]}$ and $X_e := (X_{2i+1})_{i \in [0, N-1]}$, with Y_o, Y_e and G_o, G_e being defined analogously. The prover computes $\gamma_0 = \langle X_o, Y_e \rangle$ and $\gamma_1 = \langle X_e, Y_o \rangle$, and additionally computes two commitments C_0 and C_1, where C_0 is g^{γ_0} times the Pedersen commitment to Y_e using generators G_o and C_1 is g^{γ_1} times the Pedersen commitment to Y_o using generators G_e. Upon receiving C_0, C_1, the verifier samples and sends uniformly random challenge $\alpha \in \mathbb{F}$ to the prover. The prover then sends $\gamma' = \alpha^2 \cdot \gamma_0 + \gamma + \alpha^{-2} \cdot \gamma_1$ to the verifier, and both parties compute new commitment $C' = (C_0)^{\alpha^2} \cdot C \cdot (C_1)^{\alpha^{-2}}$, new public vector $X' = \alpha^{-1} \cdot X_e + \alpha \cdot X_o$, and new generator vector $G' = (G_e)^{\alpha^{-1}} * (G_o)^{\alpha}$, where "$*$" denotes entry-wise multiplication and the exponents are computed for every entry. Additionally, the prover computes $Y' = \alpha \cdot Y_e + \alpha^{-1} \cdot Y_o$. This reduction continues until the vectors X', Y', G' all have length 1, at which point the prover simply sends Y' to the verifier, yielding $n = \log(N)$ rounds for the protocol.

Like the sum-check protocol, we take advantage of the above reduction's structure to give an efficient HMMP model protocol. Suppose that $N/(2M)$ is an integer. For $i \in \{o, e\}$, partition vectors X_i, Y_i, G_i into vectors $X_{i,j}, Y_{i,j}, G_{i,j}$ each of size $N/(2M)$ for $j \in \{1, \ldots, M\}$. Then each audit window $P_{\pi(j)}, \ldots, P_{\pi(j+\tau-1)}$ computes and broadcasts the values

$$\gamma_{0,j} = \langle X_{o,j}, Y_{e,j} \rangle \qquad C_{0,j} = \mathsf{Com}(G_{o,j}, Y_{e,j})$$
$$\gamma_{1,j} = \langle X_{e,j}, Y_{o,j} \rangle \qquad C_{1,j} = \mathsf{Com}(G_{e,j}, Y_{o,j})$$

where $\mathsf{Com}(u, v) := \prod_i u_i^{v_i}$ (i.e., a Pedersen commitment). After all audit windows have broadcast their values, every prover computes

$$\gamma_0 = \sum_{j=1}^{M} \gamma_{0,j} \qquad C_0 = g^{\gamma_0} \cdot \prod_{j=1}^{M} C_{0,j}$$
$$\gamma_1 = \sum_{j=1}^{M} \gamma_{1,j} \qquad C_1 = g^{\gamma_1} \cdot \prod_{j=1}^{M} C_{1,j}$$

and broadcasts (C_0, C_1) to the verifier.

Upon receiving α from the verifier, each prover locally computes the values γ' and C' as specified. The provers then jointly compute the vectors X', Y', and G' as follows. Each audit window $P_{\pi(j)}, \ldots, P_{\pi(j+\tau-1)}$

computes and broadcasts

$$X'_j = \alpha^{-1} X_{e,j} + \alpha X_{o_j} \qquad Y'_j = \alpha Y_{e,j} + \alpha^{-1} Y_{o,j} \qquad G'_j = (G_{e,j})^{\alpha^{-1}} * (G_{o,j})^{\alpha}.$$

Once all provers have broadcast their values, each prover obtains X', Y', G' by concatenating X'_j, Y'_j, G'_j in order.

For any round $r \in \{1, \ldots, n\}$ such that $2^{n-r-1}/M \geqslant 1$, every prover performs $O(\tau \cdot (2^{n-r}/M))$ field multiplications and group exponents, and performs $O(\tau \cdot (2^{n-r}/M + M))$ field additions and group multiplications. Additionally, each prover broadcasts $O(\tau \cdot (2^{n-r}/M))$ field and group elements. Moreover, for any round r such that $2^{n-r-1} < M$, we can divide up the remaining computation the same way we did for the sum-check protocol. If r^* is the round such that $2^{n-r^*-1} < M \leqslant 2^{n-r^*}$, then over all rounds $r = r^*, \ldots, n$ each prover performs $O(\tau)$ field and group operations, and a total of $O(\tau \cdot M)$ field and group elements are broadcast. Additionally, the complexity of each prover over all rounds $r = 1, \ldots, r^* - 1$ is $O(\tau \cdot (N/M))$ field multiplications and group exponents and $O(\tau \cdot (N/M + n \cdot M))$ field additions and group multiplications. This yields total prover complexity $O(\tau \cdot (N/M))$ field multiplications and group exponents and $O(\tau \cdot (N/M + n \cdot M))$ field additions and group additions. Moreover, each prover broadcasts a total of $O(\tau \cdot n \cdot (N/M))$ field elements. This gives Proposition 2.5.

6 Extensions of the Honest Majority Multi-Prover Model

One can easily re-imagine the HMMP model and interactive arguments in the HMMP model by tweaking any of the moving parts. We discuss some potential modifications below, and leave it as future work to achieve HMMP model constructions with such modifications.

Adaptive Corruption. It is crucial for our construction of $\boldsymbol{P}_{\text{hmp}}$ in Protocol 1 that adversarial corruption is static and occurs before the generation of the permutation π. In our construction, *any* adaptive corruption break both correctness and soundness. Though we believe our static corruption is reasonable to assume in the real world, giving a HMMP model argument that is secure against adaptive corruptions would greatly increase the applicability of this model, as well as give interesting new transformations and protocols in this setting.

Communication Between Provers. For simplicity, we assume an authenticated sender broadcast channel with which the provers use for communication. This allows provers to easily identify who sent which messages and easily perform audits as necessary. One could also change this communication assumption (e.g., secure point-to-point, asynchronous sending, etc.) and define the HMMP model with respect to this communication channel. Of course, this could simplify some aspects of the HMMP model as well as complicate other aspects.

Communication with the Verifier. Again for simplicity, we assume that the provers communicate with the verifier via the same authenticated broadcast channel. This allows the verifier to easily send its message to all provers simultaneously, as well as allowing the prover to identify which provers are sending which messages. The second point above helps a verifier ignore multiple messages sent by the same prover during any round (e.g., a malicious prover trying to flood the verifier with messages).

7 Future Work

We have demonstrated the efficiency gains of collaborative proof generation when privacy is not a concern. Even for small values of M, our HMMP model directly yields concrete efficiency gains; e.g., for $M = 2^3$ and $\tau = 2^2$, our provers experience a roughly 2× decrease in overall computational costs, at the cost of (a modest amount of) communication. For another setting of $M = 2^5$ and $\tau = 2^3$, we obtain a 4× decrease.

Our HMMP model can easily be tweaked by modifying different parts of it. Some potential modifications, left as future work, include modifying the model to handle adaptive corruptions, modifying the communication channels used by the provers, and modifying the communication channels used between the provers and the verifier. Additionally, one can imagine adjusting the adversarial model to trying to protect against adversaries with the goal of forcing other provers to perform computation proportional to a single prover. Our framework is modular and lends itself well to different aspects being tweaked to handle many situations.

8 Acknowledgements

Alexander R. Block was supported in part by NSF CCF #1910659. Christina Garman was supported in part by Intelligence Advanced Research Projects Activity (IARPA) contract #2019-19020700004.

References

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, 2018.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 1988.

[BCC$^+$16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology – EUROCRYPT*, 2016.

[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 *Innovations in Theoretical Computer Science Conference*, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *ACM Symposium on Theory of Computing*, 2013.

[BCG$^+$20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *IEEE Symposium on Security and Privacy*, 2020.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In *ITCS*, pages 401–414. ACM, 2013.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: elastic snarks for diverse environments. In *Advances in Cryptology – EUROCRYPT (to appear)*, 2022.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography*, 2013.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In *Advances in Cryptology – EUROCRYPT*, 2020.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Advances in Cryptology — CRYPTO*, 1992.

[BHR$^+$20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In *Theory of Cryptography*, 2020.

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In *Advances in Cryptology – CRYPTO*, 2021.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing*, 1988.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *International Colloquium on Automata, Languages, and Programming*, 2018.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, 2014.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography*, 2016.

[BSGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling Outside the Box Improves Soundness. In *Theoretical Computer Science Conference*, 2020.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology – EUROCRYPT*, 2020.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science Conference*, 2012.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography*, 2012.

[DPP+22] Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any np statement jointly? efficient distributed-prover zero-knowledge protocols. Privacy Enhancing Technologies Symposium, 2022.

[Eth] Etherium Foundation. Zk-rollups. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO*, 1987.

[GGM16] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security*, 2016.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology – EUROCRYPT*, 2013.

[GI08] Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology – EUROCRYPT*, 2008.

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum snarks for R1CS. *IACR Cryptol. ePrint Arch.*, page 1043, 2021.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing*, 1987.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *ACM Symposium on Theory of Computing*, 2011.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *ACM Symposium on Theory of Computing*, 2007.

[Kil91] Joe Kilian. A general completeness theorem for two party games. In *ACM Symposium on Theory of Computing*, 1991.

[KMS$^+$16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy*, 2016.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. Redshift: Transparent snarks from list polynomial commitment iops. Cryptology ePrint Archive, Report 2019/1400, 2019.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT*, 2010.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography*, 2021.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 1992.

[Lin03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 2003.

[LNS20] Jonathan Lee, Kirill Nikitin, and Srinath Setty. Replicated state machines without replicated execution. In *IEEE Symposium on Security and Privacy*, 2020.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 1982.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 2000.

[NPY20] Moni Naor, Merav Parter, and Eylon Yogev. *The Power of Distributed Verifiers in Interactive Proofs.* 2020.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-snarks: Zero-knowledge proofs for distributed secrets. USENIX Security Symposium, 2022.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO*, 1991.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 1980.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography*, 2013.

[Rab83] Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.

[RBO89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *ACM Symposium on Theory of Computing*, 1989.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *ACM Symposium on Theory of Computing*, 2016.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

[SG11] Aviad Rubinstein Shafi Goldwasser, Huijia Lin. Delegation of computation without rejection problem from designated verifier cs-proofs. Cryptology ePrint Archive, Report 2011/456, 2011.

[SJK$^+$17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, 2017.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zksnarks. Cryptology ePrint Archive, Report 2020/1275, 2020.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology – CRYPTO*, 2013.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography*, 2008.

[WBT$^+$17] Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *IEEE Symposium on Security and Privacy*, 2017.

[WTS$^+$18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *IEEE Symposium on Security and Privacy*, 2018.

[WZC$^+$18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *USENIX Security Symposium*, 2018.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. *IACR Cryptol. ePrint Arch.*, page 1010, 2022.

[XZZ$^+$19] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology – CRYPTO*, 2019.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Symposium on Foundations of Computer Science*, 1982.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE Symposium on Security and Privacy*, 2020.

Protocol 7: HMMP Model Sum-Check Protocol

Parties : M provers $\boldsymbol{P}$ and verifier V.

Prover Input : A multi-variate polynomial $g \in \mathbb{F}[X_1, \ldots, X_n]$ of individual degree at most d, a subset $H \subset \mathbb{F}$, and a value $y \in \mathbb{F}$.

Verifier Input : Individual degree upper bound d, a subset $H \subset \mathbb{F}$, and value $y \in \mathbb{F}$.

Verifier Output : Accept or Reject.

Oracle : The verifier V has oracle access to evaluations of the polynomial g.

```
$\boldsymbol{P}$ initializes $r = \emptyset$ and $\mathsf{count} = 1$.
$V$ initializes $y' = y$.
foreach $j \in [n]$ do
    if $|H|^{n-j} \geqslant M$ then
        $\boldsymbol{P}$ partitions $H^{n-j}$ into $M$ subsets $H_1, \ldots, H_M$ each of size $H^{n-j}/M$.
        foreach $i \in [M]$ do
            foreach $k \in [i, i + \tau - 1]_M$ do
                $P_{\pi(k)}$ computes and broadcasts $g_{j,i,k}(X_j) := \sum_{x \in H_k^{n-j}} g(r, X_j, x)$.
    else
        foreach $i = 1, \ldots, |H|^{n-j}$ do
            foreach $k \in [\mathsf{count}, \mathsf{count} + \tau - 1]_M$ do
                $P_{\pi(k)}$ computes and broadcasts $g_{j,i,k}(X_j) := g(r, X_j, h_i)$, where $h_i$ is the $i^{th}$ element of
                  $H^{n-j}$.
                $\boldsymbol{P}$ updates $\mathsf{count} = \mathsf{count} + 1 (\bmod M)$.
    if not all $g_{j,i,k}$ are equal then
        All provers set $g_{j,i}(X_j) := \mathsf{Audit}(i, \{g_{j,i,k}\}_k)$.
    else All provers set $g_{j,i}(X_j) := g_{j,i,i}(X_j)$.
    foreach $i \in [M]$ do $P_i$ computes $g_j^{(i)}(X_j) := \sum_k g_{j,k}(X_j)$ and broadcasts it to $V$.
    $V$ sets $g_j(X_j) = \text{majority}(g_j^{(1)}(X_j), \ldots, g_j^{(M)}(X_j))$.
    $V$ checks that $g_j(X_j)$ has degree at most $d$ and $y' = \sum_{x \in H} g_j(x)$, outputting reject if either do not
      hold.
    $V$ samples $r_j \xleftarrow{\$} \mathbb{F}$.
    if $j < n$ then
        $V$ sets $y' = g_j(r_j)$, and broadcasts $r_j$ to $\boldsymbol{P}$.
        $\boldsymbol{P}$ updates $r = (r, r_j)$.
    else $V$ queries the oracle and checks $g_n(r_n) = g(r_1, \ldots, r_n)$, aborting with Reject if not.
$V$ outputs Accept.
```

Protocol 8: Eval protocol

```
Parties           : Prover $P$ and verifier $V$.
Prover Input      : Public parameters pp, commitment $C$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}$, and $Y \in \mathbb{F}^N$
Verifier Input    : Public parameters pp, commitment $C$, $\zeta \in \mathbb{F}^n$, and $\gamma \in \mathbb{F}$
Verifier Output   : Accept or Reject.
$V$ samples and sends $g_V \xleftarrow{\$} \mathbb{G}$.
$P$ and $V$ define $C_\gamma := C \cdot g_V^\gamma \in \mathbb{G}$.
$P$ and $V$ define sequence $Z = (Z_b = \beta(b, \zeta))_{b \in \{0,1\}^n} \in \mathbb{F}^N$.
$P$ and $V$ engage in $\mathsf{EvalReduce}(C_\gamma, Z, \gamma, \boldsymbol{g}, g_V, N; Y)$.
Procedure $\mathsf{EvalReduce}(C_\gamma, Z, \gamma, \boldsymbol{g}, g_V, N; Y)$
    if $N = 1$ then
        $P$ sends $Y \in \mathbb{F}$ to $V$.
        $V$ outputs Accept if and only if $C_\gamma = \boldsymbol{g}^Y \cdot g_V^{(Y \cdot Z)}$ and $\gamma = \langle Y, Z \rangle$, where $\boldsymbol{g} \in \mathbb{G}$ and $Z \in \mathbb{F}$; else
         outputs Reject.
    Set $n = \log(N)$.
    $P$ computes $\gamma_0, \gamma_1$ as
        $$\gamma_0 = \sum_{b \in \{0,1\}^{n-1}} Y_{b \circ 0} \cdot Z_{b \circ 1} \qquad \gamma_1 = \sum_{b \in \{0,1\}^{n-1}} Y_{b \circ 1} \cdot Z_{b \circ 0}.$$
    $P$ computes and sends $C_0, C_1$ to $V$, where
        $$C_0 = g_V^{\gamma_0} \cdot \prod_{b \in \{0,1\}^{n-1}} (\boldsymbol{g}_{b \circ 1})^{Y_{b \circ 0}} \qquad C_1 = g_V^{\gamma_1} \cdot \prod_{b \in \{0,1\}^{n-1}} (\boldsymbol{g}_{b \circ 0})^{Y_{b \circ 1}}.$$
    $V$ samples and sends $\alpha \xleftarrow{\$} \mathbb{F}$ to $P$.
    $P$ computes and sends $\gamma' = \alpha^2 \cdot \gamma_0 + \gamma + \alpha^{-2} \cdot \gamma_1$.
    $P$ and $V$ both compute
        $$C'_{\gamma'} = (C_0)^{\alpha^2} \cdot C_\gamma \cdot (C_1)^{\alpha^{-2}}$$
        $$Z' = \left(Z'_b = \alpha^{-1} \cdot Z_{b \circ 0} + \alpha \cdot Z_{b \circ 1}\right)_{b \in \{0,1\}^{n-1}}$$
        $$\boldsymbol{g}' = \left(\boldsymbol{g}'_b = (\boldsymbol{g}_{b \circ 0})^{\alpha^{-1}} \cdot (\boldsymbol{g}_{b \circ 1})^{\alpha}\right)_{b \in \{0,1\}^{n-1}}.$$
    $P$ computes $Y' = \left(Y'_b = \alpha \cdot Y_{b \circ 0} + \alpha^{-1} \cdot Y_{b \circ 1}\right)_{b \in \{0,1\}^{n-1}}$.
    $P$ and $V$ engage in $\mathsf{EvalReduce}(C'_{\gamma'}, Z', \gamma', \boldsymbol{g}', g_V, N/2; Y')$
```

Protocol 9: HMMP Model Eval Protocol

```
Parties         : $M$ provers $\boldsymbol{P}$ and verifier $V$.
Prover Input    : Public parameters $\mathsf{pp}$, commitment $C$, $\zeta \in \mathbb{F}^n$, $\gamma \in \mathbb{F}$, and $Y \in \mathbb{F}^N$
Verifier Input  : Public parameters $\mathsf{pp}$, commitment $C$, $\zeta \in \mathbb{F}^n$, and $\gamma \in \mathbb{F}$.
Verifier Output : Accept or Reject.
$V$ samples and sends $g_V \xleftarrow{\$} \mathbb{G}$.
$\boldsymbol{P}$ and $V$ define $C_\gamma := C \cdot g_V^\gamma \in \mathbb{G}$. $\boldsymbol{P}$ additionally initializes $\mathsf{count} = 1$.
$\boldsymbol{P}$ partitions $\{0,1\}^n$ into $M$ subsets $S_1, \ldots, S_M$ each of size $N/M$.
foreach $i \in [M]$ do
    foreach $j \in [i, i + \tau - 1]_M$ do
        $P_{\pi(j)}$ computes and broadcasts $Z_{i,j} = (\beta(b, \gamma))_{b \in S_i} \in \mathbb{F}^{N/M}$.
    if not all $Z_{i,j}$ are equal then
        All provers set $Z_i := \mathsf{Audit}(i, \{Z_{i,j}\}_j)$.
    else All provers set $Z_i := Z_{i,i}$.
while $N > 1$ do
    Set $n = \log(N)$.
    if $N/2 \geqslant M$ then
        Run Protocol 10 with inputs $(N/2, n, Y, Z, \boldsymbol{g})$ and obtain $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and $\{C_{1,i}\}_i$.
    else
        Run Protocol 11 with inputs $(N/2, Y, Z, \boldsymbol{g}, \mathsf{count})$ and obtain $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and
         $\{C_{1,i}\}_i$
    foreach $i \in [M]$ do
        $P_i$ computes $\gamma_0^{(i)} := \sum_i \gamma_{0,i}$, $\gamma_1^{(i)} := \sum_i \gamma_{1,i}$, $C_0^{(i)} := g_V^{\gamma_0} \cdot \prod_i C_{0,i}$, and $C_1^{(i)} := g_V^{\gamma_1} \cdot \prod_i C_{0,i}$
         and broadcasts $C_0^{(i)}, C_1^{(i)}$ to $V$.
    $V$ samples and broadcasts $\alpha \xleftarrow{\$} \mathbb{F}$ to $\boldsymbol{P}$
    foreach $i \in [M]$ do
        $P_i$ computes and broadcasts $\gamma' = \alpha^2 \cdot \gamma_0 + \gamma + \alpha^{-2} \cdot \gamma_1$ to $V$.
    $V$ and all provers locally compute $C_\gamma := (C_0)^{\alpha^2} \cdot C_\gamma \cdot (C_1)^{\alpha^{-2}}$. $V$ additionally computes
        $$Z' = (\alpha^{-1} \cdot Z_{b \circ 0} + \alpha \cdot Z_{b \circ 1})_{b \in \{0,1\}^{n-1}}$$
        $$\boldsymbol{g}' = ((\boldsymbol{g}_{b \circ 0})^{\alpha^{-1}} \cdot (\boldsymbol{g}_{b \circ 1})^{\alpha})_{b \in \{0,1\}^{n-1}}$$
    if $N/2 \geqslant M$ then
        Run Protocol 12 with inputs $(N/2, n, Y, Z, \boldsymbol{g}, \alpha)$ and obtain $Y, Z \in \mathbb{F}^{N/2}$ and $\boldsymbol{g} \in \mathbb{G}^{N/2}$.
    else
        Run Protocol 13 with inputs $(N/2, Y, Z, \boldsymbol{g}, \alpha, \mathsf{count})$ and obtain $Y, Z \in \mathbb{F}^{N/2}$ and $\boldsymbol{g} \in \mathbb{G}^{N/2}$.
    Update $N \leftarrow N/2$.
Each $P_i$ broadcasts $Y \in \mathbb{F}$ and $Z \in \mathbb{F}$ to $V$.
$V$ outputs Accept if and only if $C_\gamma = \boldsymbol{g}^Y \cdot g_V^{(Y \cdot Z)}$; else outputs Reject.
```

Protocol 10: Polycom Compute Partial Evals and Commitments 1

```
Input  : Integers $N/2$ and $n$, $Y, Z \in \mathbb{F}^N$, and $\boldsymbol{g} \in \mathbb{G}^N$
Output : Values $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and $\{C_{1,i}\}_i$
$\boldsymbol{P}$ partitions $\{0,1\}^{n-1}$ into $M$ subsets $B_1, \ldots, B_M$ each of size $N/(2M)$.
foreach $i \in [M]$ do
    foreach $j \in [i, i+\tau-1]_M$ do
        $P_{\pi(j)}$ computes and broadcasts
            $\gamma_{0,i,j} = \sum_{b \in B_i} Y_{b \circ 0} \cdot Z_{b \circ 1}$        $C_{0,i,j} = \prod_{b \in B_i} (\boldsymbol{g}_{b \circ 1})^{Y_{b \circ 0}}$
            $\gamma_{1,i,j} = \sum_{b \in B_i} Y_{b \circ 1} \cdot Z_{b \circ 0}$        $C_{1,i,j} = \prod_{b \in B_i} (\boldsymbol{g}_{b \circ 0})^{Y_{b \circ 1}}$.
    if not all $\{\gamma_{0,i,j}\}_j$ are equal then  All provers set $\gamma_{0,i} = \mathsf{Audit}(i, \{\gamma_{0,i,j}\}_j)$.
    else All provers set $\gamma_{0,i} = \gamma_{0,i,i}$.
    if not all $\{\gamma_{1,i,j}\}_j$ are equal then  All provers set $\gamma_{1,i} = \mathsf{Audit}(i, \{\gamma_{1,i,j}\}_j)$.
    else All provers set $\gamma_{1,i} = \gamma_{1,i,i}$.
    if not all $\{C_{0,i,j}\}_j$ are equal then  All provers set $C_{0,i} = \mathsf{Audit}(i, \{C_{0,i,j}\}_j)$.
    else All provers set $C_{0,i} = C_{0,i,i}$.
    if not all $\{C_{1,i,j}\}_j$ are equal then  All provers set $C_{1,i} = \mathsf{Audit}(i, \{C_{1,i,j}\}_j)$.
    else All provers set $C_{1,i} = C_{1,i,i}$.
return $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and $\{C_{1,i}\}_i$.
```

Protocol 11: Polycom Compute Partial Evals and Commitments 2

```
Input  : Integer $N/2$, $Y, Z \in \mathbb{F}^N$, $\boldsymbol{g} \in \mathbb{G}^N$, and integer $\mathsf{count}$.
Output : Values $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and $\{C_{1,i}\}_i$
foreach $i = 0, \ldots, N/2-1$ do
    foreach $j \in [\mathsf{count}, \mathsf{count}+\tau-1]_M$ do
        $P_{\pi(j)}$ computes and broadcasts $\gamma_{0,i,j} = Y_{i \circ 0} \cdot Z_{i \circ 1}$, $\gamma_{1,i,j} = Y_{i \circ 1} \cdot Z_{i \circ 0}$, $C_{0,i,j} = (\boldsymbol{g}_{i \circ 1})^{Y_{i \circ 0}}$,
         $C_{1,i,j} = (\boldsymbol{g}_{i \circ 0})^{Y_{i \circ 1}}$.
    if not all $\{\gamma_{0,i,j}\}_j$ are equal then  All provers set $\gamma_{0,i} = \mathsf{Audit}(i, \{\gamma_{0,i,j}\}_j)$.
    else All provers set $\gamma_{0,i} = \gamma_{0,i,i}$.
    if not all $\{\gamma_{1,i,j}\}_j$ are equal then  All provers set $\gamma_{1,i} = \mathsf{Audit}(i, \{\gamma_{1,i,j}\}_j)$.
    else All provers set $\gamma_{1,i} = \gamma_{1,i,i}$.
    if not all $\{C_{0,i,j}\}_j$ are equal then  All provers set $C_{0,i} = \mathsf{Audit}(i, \{C_{0,i,j}\}_j)$.
    else All provers set $C_{0,i} = C_{0,i,i}$.
    if not all $\{C_{1,i,j}\}_j$ are equal then  All provers set $C_{1,i} = \mathsf{Audit}(i, \{C_{1,i,j}\}_j)$.
    else All provers set $C_{1,i} = C_{1,i,i}$.
    $\mathsf{count} = \mathsf{count} + 1 (\bmod M)$.
return $\{\gamma_{0,i}\}_i$, $\{\gamma_{1,i}\}_i$, $\{C_{0,i}\}_i$, and $\{C_{1,i}\}_i$.
```

Protocol 12: Polycom Compute Partial Sequences 1

```
Input  : Integers $N/2$ and $n$, $Y, Z \in \mathbb{F}^N$, $\boldsymbol{g} \in \mathbb{G}^N$, and $\alpha \in \mathbb{F}$.
Output : Sequences $Y, Z \in \mathbb{F}^{N/2}, \boldsymbol{g} \in \mathbb{G}^{N/2}$.
$\boldsymbol{P}$ partitions $\{0,1\}^{n-1}$ into $M$ subsets $B_1, \ldots, B_M$ each of size $N/(2M)$.
foreach $i \in [M]$ do
    foreach $j \in [i, i+\tau-1]_M$ do
        $P_{\pi(j)}$ computes and broadcasts
            $Z'_{i,j} = (\alpha^{-1} \cdot Z_{b\circ 0} + \alpha \cdot Z_{b\circ 1})_{b \in B_i}$
            $Y'_{i,j} = (\alpha \cdot Y_{b\circ 0} + \alpha^{-1} \cdot Y_{b\circ 1})_{b \in B_i}$
            $\boldsymbol{g}'_{i,j} = ((\boldsymbol{g}_{b\circ 0})^{\alpha^{-1}} \cdot (\boldsymbol{g}_{b\circ 1})^{\alpha})_{b \in B_i}$
    if not all $\{Z'_{i,j}\}_j$ are equal then  All provers set $Z'_i = \mathsf{Audit}(i, \{Z'_{i,j}\}_j)$.
    else All provers set $Z'_i = Z'_{i,i}$.
    if not all $\{Y'_{i,j}\}_j$ are equal then  All provers set $Y'_i = \mathsf{Audit}(i, \{Y'_{i,j}\}_j)$.
    else All provers set $Y'_i = Y'_{i,i}$.
    if not all $\{\boldsymbol{g}'_{i,j}\}_j$ are equal then  All provers set $\boldsymbol{g}'_i = \mathsf{Audit}(i, \boldsymbol{g}'_{i,j}\}_j)$.
    else All provers set $\boldsymbol{g}'_i = \boldsymbol{g}'_{i,i}$.
return $Y = (Y'_1, \ldots, Y'_M)$, $Z = (Z'_1, \ldots, Z'_M)$, and $\boldsymbol{g} = (\boldsymbol{g}'_1, \ldots, \boldsymbol{g}'_M)$.
```

Protocol 13: Polycom Compute Partial Sequences 2

```
Input  : Integers $N/2$ and $n$, $Y, Z \in \mathbb{F}^N$, $\boldsymbol{g} \in \mathbb{G}^N$, $\alpha \in \mathbb{F}$, and integer $\mathsf{count}$.
Output : Sequences $Y, Z \in \mathbb{F}^{N/2}, \boldsymbol{g} \in \mathbb{G}^{N/2}$.
foreach $i = 0, \ldots, N/2 - 1$ do
    foreach $j \in [\mathsf{count}, \mathsf{count} + \tau - 1]_M$ do
        $P_{\pi(j)}$ computes and broadcasts $Z'_{i,j} = \alpha^{-1} \cdot Z_{i\circ 0} + \alpha \cdot Z_{i\circ 1}$, $Y'_{i,j} = \alpha \cdot Y_{i\circ 0} + \alpha^{-1} \cdot Y_{i\circ 1}$, and
          $\boldsymbol{g}'_{i,j} = (\boldsymbol{g}_{i\circ 0})^{\alpha^{-1}} \cdot (\boldsymbol{g}_{i\circ 1})^{\alpha}$.
    if not all $\{Z'_{i,j}\}_j$ are equal then  All provers set $Z'_i = \mathsf{Audit}(i, \{Z'_{i,j}\}_j)$.
    else All provers set $Z'_i = Z'_{i,i}$.
    if not all $\{Y'_{i,j}\}_j$ are equal then  All provers set $Y'_i = \mathsf{Audit}(i, \{Y'_{i,j}\}_j)$.
    else All provers set $Y'_i = Y'_{i,i}$.
    if not all $\{\boldsymbol{g}'_{i,j}\}_j$ are equal then  All provers set $\boldsymbol{g}'_i = \mathsf{Audit}(i, \boldsymbol{g}'_{i,j}\}_j)$.
    else All provers set $\boldsymbol{g}'_i = \boldsymbol{g}'_{i,i}$.
    $\mathsf{count} = \mathsf{count} + 1 (\mathrm{mod}\, M)$.
return $Y = (Y'_1, \ldots, Y'_M)$, $Z = (Z'_1, \ldots, Z'_M)$, and $\boldsymbol{g} = (\boldsymbol{g}'_1, \ldots, \boldsymbol{g}'_M)$.
```