SNARK Lower Bounds via Communication Complexity*

Rishabh Bhadauria[†1], Alexander R. Block[2], Prantar Ghosh[3], and Justin Thaler[4]

[1]rishabh.bhadauria@gmail.com
[2]University of Illinois at Chicago. alexander.r.block@gmail.com
[3]Tennessee Tech University. pghosh@tntech.edu
[4]a16z crypto research and Georgetown University. justin.r.thaler@gmail.com

December 18, 2025

Abstract

We initiate the study of lower bounding the verification time of Succinct Non-interactive ARguments of Knowledge (SNARKs) built in the Polynomial Interactive Oracle Proof + Polynomial Commitment Scheme paradigm. The verification time of these SNARKs is generally dominated by the polynomial commitment scheme, and so we want to understand if polynomial commitment schemes admit lower bounds on the verification time. By recognizing that polynomial commitment schemes are also often built by applying cryptography to some information-theoretic core protocol, we seek to separate this core from the cryptography in a way that meaningfully captures the verification time required by the polynomial commitment scheme verifier.

We provide strong evidence that several polynomial commitment schemes have (nearly) *optimal* verifier times. Our evidence comes from connecting polynomial commitment schemes to certain information-theoretic protocols known as *communication protocols* from the field of communication complexity, a link which we believe to be of independent interest. Through this lens, we model the verifier work in the cryptographic protocols as information (i.e., number of bits) exchanged between parties in the communication protocols, allowing us to leverage lower bounds from communication complexity. These lower bounds give strong evidence that the verifier time in these polynomial commitment schemes must be at least the number of bits exchanged in the communication protocol.

We extract the communication protocol cores of three polynomial commitment schemes and lower bound the bits exchanged in these cores. The lower bounds we obtain match (up to poly-logarithmic factors) the best-known (asymptotic) verification times of the polynomial commitment schemes we examine in this work. Specifically, we show that for univariate/multilinear polynomials of size $N = 2^n$:

- the communication core of Hyrax PCS (Wahby et al., S&P 2016) requires $\Omega(\sqrt{N})$ bits to be exchanged;
- the communication core of Bulletproofs PCS (Bootle et al., EUROCRYPT 2016; Bünz et al., S&P 2018) requires $\Omega(N)$ bits to be exchanged; and
- the communication core of Dory PCS (Lee, TCC 2021) requires $\Omega(\log(N))$ bits to be exchanged.

Our results strongly suggest a negative answer to a longstanding open question on whether the Bulletproofs verifier can run in sublinear time.

[†]Work partially done while at Georgetown University.

Contents

1 Introduction

Succinct Non-interactive ARguments of Knowledge (SNARKs) allow a computationally powerful prover to convince a computationally weak verifier of the truth of some (e.g., **NP**) statement via a (non-interactive) proof of sublinear size. It is well-known that modern efficient SNARK constructions can be divided into two components: an information-theoretically secure component (typically a polynomial interactive oracle proof (PIOP) [BFS20]), and a cryptographic protocol (typically a polynomial commitment scheme (PCS) [KZG10]). These two interactive components are combined to obtain a succinct interactive argument, which is then rendered non-interactive via the Fiat-Shamir transformation [FS86].

Examining SNARKs built from this combination of PIOPs and PCSs, what is less recognized is that the PCS itself can typically be divided into two of the same components: an information-theoretic one and a cryptographic one. Notable exceptions (i.e., ones which put this display at the forefront of their design) include PCSs based on (or inspired by) the FRI protocol [BBH+18, ZCF24, ACF+24], and those based on Ligero/Brakedown [AHI+23, BFH+20, BBH+22, GLS+23, BFK+24]. With these notable exceptions, the vast majority of other PCSs in the literature (e.g., discrete-log based schemes [BCC+16, BBB+18, WTS+18, BHR+20], pairing-based schemes [KZG10, XZZ+19, Lee21, BHV+23], groups of unknown order based schemes [BFS20, BHR+21], and lattice-based schemes [ACL+22, BCF+22, AFL+24, CMN+24, NS24, HSS24, FMN24]) do not highlight or examine the PCS from this same perspective.

1.1 Our Contributions

In this work, we attempt to elucidate this viewpoint by isolating the information-theoretic cores of three well-studied polynomial commitment schemes. This perspective and goal bring two benefits. First is expository: understanding the information-theoretic core of these protocols gives us insights into what is "really happening" in these commitment schemes; i.e., what is the cryptography guaranteeing? Second, this view allows us to provide evidence that certain commitment schemes are *optimal*, in terms of *verification time*. Specifically, we can show *unconditionally* that (certain components of) the information-theoretic cores of these commitment schemes cannot be improved (up to constant factors), so building "better" commitment schemes to circumvent our results will have to deviate significantly from the current design paradigms.

We examine three polynomial commitment schemes through this lens: Hyrax [WTS+18], Bulletproofs [BCC+16, BBB+18], and Dory [Lee21]. We separate the information-theoretic cores from the cryptography of these protocols, allowing us to understand what the cryptography is actually achieving; i.e., which properties are needed in the underlying cores that the cryptography guarantees. Moreover, by isolating these cores, we obtain strong evidence that the verification time of all the aforementioned schemes is (nearly) optimal. We do this by translating the verifier work in the polynomial commitment schemes to the *number of bits exchanged* between parties in the information-theoretic cores, then obtain *lower bounds* on this number of bits.

The information-theoretic cores we extract from these polynomial commitment schemes are information-theoretic protocols known as *communication protocols* from the field of *communication complexity* (see Section 1.2.1). Communication protocols are information-theoretic protocols with computationally unbounded parties that seek to understand how many bits are required to be exchanged between parties in order to compute some discrete function (with inputs spread across parties). That is, the complexity measure of a communication protocol is explicitly this number of exchanged bits.

Our results frame Hyrax, Bulletproofs, and Dory as communication protocols by stripping away the cryptography and understanding what guarantees the cryptography provides, then translating those guarantees into a suitable communication protocol. This framing allows us to leverage known and new lower bounds for communication protocols; in particular, we show that the communication cores of Hyrax, Bulletproofs,

and Dory require a certain number of bits to be exchanged between parties (otherwise soundness of the core is violated). The lower bounds we obtain nearly match the verification time of the specified schemes, giving strong evidence that the verification times of Hyrax, Bulletproofs, and Dory are *(nearly) optimal*. In other words, to achieve more efficient verification time in these schemes, our results suggest that significant modifications would be needed: any modification that can be captured using the same communication protocols we obtain for Hyrax, Bulletproofs, and Dory is subject to our lower bounds. Informally, we show the following results.

Theorem 1.1 (Informal; see Corollaries 4.2, 5.2 and 6.2)**.** *For $N = 2^n$ and any n-variate multilinear or degree-N univariate polynomial over a finite field $\mathbb{F}$,*

1. *The communication protocol core of Hyrax-PCS requires $\Omega(\sqrt{N})$ bits of communication;*
2. *The communication protocol core of Bulletproofs-PCS requires $\Omega(N)$ bits of communication;*
3. *The communication protocol core of Dory-PCS requires $\Omega(\log(N))$ bits of communication.*

Note that each scheme under consideration operates using group-based assumptions, with Hyrax and Bulletproofs utilizing the discrete-logarithm assumption, and Dory additionally employing pairing-friendly groups. In light of this, the verification costs of Hyrax, Bulletproofs, and Dory nearly match the lower bounds of Theorem 1.1, up to polylog $|\mathbb{G}|$ factors, assuming single $\mathbb{G}/\mathbb{F}$ operations (including pairings) require polylog $|\mathbb{G}|$ bit-operations and $|\mathbb{G}| \geqslant |\mathbb{F}|$ for group $\mathbb{G}$ with scalar field $\mathbb{F}$.

Notably, (2) in Theorem 1.1 gives evidence towards a resolution of a long-standing open question on whether the Bulletproofs verifier is inherently linear-time. Our results show that linear communication is inherent to the underlying information-theoretic core, and any attempt to improve the Bulletproofs verifier time will require significant deviation from this core; Dory is one such deviation from Bulletproofs, which we highlight in Section 1.2.5. In particular, our results suggest that prior work on obtaining a sublinear-time Bulletproofs verifier (e.g., [Lu22, Lu24], both of which have been withdrawn) was doomed to fail. Indeed, [Gab24] gave an attack against the work presented in [Lu24], and this attack matches (in spirit) the proof of our communication lower bound.

1.1.1 Evidence vs. Provable Optimality of Verification Times

We are very careful in how we state our results: we are giving (what we believe to be) *strong evidence* that the aforementioned protocols have optimal verifier time. Our evidence is strong in the sense that the communication protocols we extract from Hyrax, Bulletproofs, and Dory *nearly match* the communication lower bounds, and these lower bounds hold for *any communication protocol* which computes the same discrete functions as the communication cores of Hyrax, Bulletproofs, and Dory. However, this is not a formal proof, and any such formal proof would require proving a statement of the form "for every polynomial commitment scheme satisfying a set of properties P, there exists a communication protocol which satisfies a set of properties P'" (here, P' is a function of P, the protocol, etc.), where the properties P, P' capture the efficiency, communication, etc., of these stated protocols. Such a statement seems difficult to even formally write down, let alone prove; nonetheless, we leave this as exciting future work to explore, as it would allow us to derive lower bounds on many more polynomial commitment schemes in a straightforward manner.

1.1.2 Polynomial Commitment Schemes from any Communication Protocol?

One may naturally wonder if we could reverse our results: namely, build a polynomial commitment scheme from any specified communication protocol, possibly subject to certain restrictions. Our results make

no claims of this form, and we strongly believe this to not be possible. It is quite likely one could cook up a communication protocol, even under some possible restrictions (e.g., like ones we introduce for our communication cores), that look nothing like a polynomial commitment scheme (as in, impossible to translate into a meaningful cryptographic protocol). This makes our results and evidence even stronger since communication complexity lower bounds are of the form "for a function f, *any* communication protocol computing f has a communication lower bound," and the communication cores we derive from the aforementioned polynomial commitment schemes have (nearly) matching communication complexity.

1.1.3 Verifiers with PCP Proof String Access?

One natural question to ask is if our lower bounds can be circumvented by a polynomial commitment scheme verifier with query access to a PCP proof string, since in this case the actual work done by the verifier is much less than reading the proof string, which could potentially circumvent our lower bounds. However, we do not believe this would circumvent our results. This is because any of the homomorphic commitment schemes we model in this work (and for which our lower bounds apply to) can also be used to implement PCP-query access for cryptographic verifiers: e.g., commit to the (multilinear extension of) the PCP proof string, and now queries to the PCP proof string now become evaluations to the committed polynomial. In fact, any polynomial commitment scheme built via PCPs is likely to already have a cryptographic prover commit to the string (e.g., via a Merkle tree), and provide openings to the verifier queries.

1.2 Technical Overview

Proving Theorem 1.1 relies on showing suitable lower bounds on the information-theoretic cores of the three polynomial commitment schemes we examine in this work. The first step is, thus, understanding *what* these information-theoretic cores are. A natural attempt might be to simply strip away the cryptography from these PCSs and obtain some information-theoretic interactive (oracle) proof; however, to the best of our knowledge, we do not know of any lower bounds on the verifier time of interactive (oracle) proof systems, so this path does not seem viable. The only result we are aware of that provide some evidence of lower bounds on the verifier time are due to Bangalore et al. [BBH+22], who show evidence of a $\text{poly}(\lambda)$ upper bound on the *space* required by certain verifiers for interactive arguments based on symmetric-key primitives (e.g., hash functions). They also provide evidence of a proof-size lower bound based on existing techniques, which translates to a lower bound on the verification time. In light of these limited results, we take a different approach and model the information-theoretic cores of these PCSs as communication protocols, allowing us to leverage tools from the rich field of communication complexity to obtain our lower bounds.

1.2.1 Communication Complexity and Merlin-Arthur Communication

The field of communication complexity, originally introduced by Yao [Yao79], studies the communication costs of computing discrete functions with inputs split between two or more parties. The most relevant communication model to us is *(online) Merlin-Arthur Communication* [BFS86, CCM+14]. A communication protocol in this model is parameterized by a function $f\colon X \times Y \to Z$, two (randomized) parties Alice and Bob, and an omniscient but untrusted helper Merlin. Alice and Bob, given inputs $x \in X$ and $y \in Y$ respectively, wish to compute $f(x, y)$ and can settle for some (constant) probability of error. Merlin-Arthur communication asks: if Alice and Bob must communicate $R(f)$ bits to compute $f(x, y)$ on their own, can Merlin, who knows (x, y), convince "Arthur" (defined as Alice and Bob together) that $f(x, y) = z$ while keeping the total communication between *all* parties $o(R(f))$?

Online Merlin-Arthur (OMA) communication modifies the above communication model and allows Bob and Merlin to interact arbitrarily, with the restriction that Bob's messages do not depend on the input y, and restricts Alice to only send a *single message* to Bob after Bob and Merlin's interaction (in particular, Alice cannot receive communication from either Bob or Merlin). Notably in these models, Alice and Bob are always honest, trusted parties, and all parties are computationally unbounded.

Why Communication Complexity? Intuitively, communication complexity seeks to measure the amount of *information that must flow* between parties in a protocol in order to compute a function f. As all parties within the protocol are computationally unbounded, the cost of such a protocol is strictly determined by the number of bits sent. Thus, if we can frame (the underlying interactive protocol of) SNARKs as communication protocols, it gives us a tangible connection to SNARK performance in two distinct ways: (1) the SNARK proof is explicitly a transfer of information from the SNARK prover to the SNARK verifier; and (2) the SNARK verifier work can be isolated and modeled as the number of bits the party "Arthur" needs to receive in order to verify this proof (i.e., verify the computation of some function).

Communication complexity, then, is a clean information-theoretic setting to formulate the information-theoretic cores of (certain) cryptographic protocols. As all parties are computationally unbounded, the "work" being done by cryptographic parties can be explicitly measured as the number of bits sent in the underlying communication protocol. This, then, allows us to utilize well-established (online) communication lower bounds for many functions, including inner product and set disjointness [BFS86, KS92, Kla03, AW09], matrix-vector products [KS93, SWY+21, CMT13], and (sparse) index [Abl96, CCG+14, GS24]. In this work, we will use OMA communication (and an extension that we introduce) to cleanly isolate the work being performed by parties in Hyrax [WTS+18], Bulletproofs [BCC+16, BBB+18], and Dory [Lee21], and establish lower bounds via known communication complexity lower bounds.

1.2.2 Review: Polynomial Commitment Schemes

We briefly give a high-level overview of polynomial commitment schemes, originally introduced by Kate, Zaverucha, and Goldberg [KZG10]. A polynomial commitment scheme, or a PCS in short, is a special cryptographic commitment scheme which first allows a committer/prover to commit to a polynomial f as some value C. Then, a receiver/verifier, given C, can request C be opened to a specific evaluation x of the underlying polynomial f via an evaluation protocol, which (informally) proves the statement "$y = f(x)$ and $f(x)$ is consistent with C." See Section 2 for the full definition.

All three polynomial commitment schemes we consider in this work utilize group-based hardness assumptions (specifically, discrete-log and pairings). For ease of presentation, we let $\mathbb{G}$ be a finite cyclic group written in additive notation with scalar finite field $\mathbb{F}$. Throughout this section, we assume that $N = 2^n$. Moreover, we discuss all polynomial commitment schemes as *multilinear* PCSs; that is, as polynomial commitment schemes for multilinear polynomials. For any function $f\colon \{0,1\}^n \to \mathbb{F}$, the *multilinear extension of* f, denoted as $\widetilde{f}\colon \mathbb{F}^n \to \mathbb{F}$ is the unique polynomial such that $\deg(\boldsymbol{X}_i) \leqslant 1$ for all $i \in [n]$ and $f(\boldsymbol{c}) = \widetilde{f}(\boldsymbol{c})$ for all $\boldsymbol{c} \in \{0,1\}^n$. Similarly, for any vector $\boldsymbol{u} \in \mathbb{F}^N$, we let $\widetilde{\boldsymbol{u}}$ denote the multilinear extension of $\boldsymbol{u}$, where we equivalently view $\boldsymbol{u}$ as a function from $\{0,1\}^n$ to $\mathbb{F}$ in the natural way. We also naturally define extensions of vectors/functions over $\mathbb{G}$ with scalar field $\mathbb{F}$: for $\boldsymbol{g} \in \mathbb{G}^N$, $\widetilde{\boldsymbol{g}}\colon \mathbb{F}^n \to \mathbb{G}$ is the multilinear extension of $\boldsymbol{g}$.

1.2.3 Hyrax: Review and Information-Theoretic Core

The Hyrax PCS [WTS+18] is built from a more general commitment scheme for verifying *vector-matrix-vector* products; we call this scheme *Hyrax-VMV*. In Hyrax-VMV, the prover is given a matrix $\mathbf{A} \in \mathbb{F}^{a \times b}$ and

commits to this matrix by computing a Pedersen commitment [Ped91] per row of $\mathbf{A}$ with random generators $\boldsymbol{g} \xleftarrow{\$} \mathbb{G}^b$; that is, the commitment is the vector $\boldsymbol{C} = \mathbf{A}\boldsymbol{g}^\top \in \mathbb{G}^a$. Then, for vectors $\boldsymbol{L} \in \mathbb{F}^a$ and $\boldsymbol{R} \in \mathbb{F}^b$ given by the verifier after receiving a commitment $\boldsymbol{C}$, Hyrax-VMV takes as input $(\boldsymbol{C}, \boldsymbol{L}, \boldsymbol{R})$ and is a proof of knowledge that the prover knows matrix $\mathbf{A}$ such that $\boldsymbol{C} = \mathbf{A}\boldsymbol{g}^\top$ and $y = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$. The proof consists of a single message from the prover: a vector $\boldsymbol{L}'$ (the purported value of $\boldsymbol{L}\mathbf{A}$) and a value y' (the purported value of $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$). Verification proceeds in 3 steps: (1) sample $\alpha \xleftarrow{\$} \mathbb{F}$; (2) check $\alpha\langle \boldsymbol{L}, \boldsymbol{C}\rangle \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{g}\rangle$, and (3) check $\alpha y' \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{R}\rangle$. Hyrax-VMV can be used as a PCS for any polynomial f such that evaluation of f at any point can be expressed as a vector-matrix-vector product; examples include multilinear polynomials and univariate polynomials (see Section 4.2 for full details).

Information-Theoretic Core of Hyrax. The above discussion leads us to frame the information-theoretic (IT) core of Hyrax as an online Merlin-Arthur (OMA) communication protocol for verifying (random) vector-matrix-vector products, which consists of two parts: Merlin convince "Arthur" that (1) $\boldsymbol{L}' = \boldsymbol{L}\mathbf{A}$, and (2) that $y = \langle \boldsymbol{L}', \boldsymbol{R}\rangle$. In particular, vector-matrix-vector products are verified by simultaneously verifying a matrix-vector product (the claim "$\boldsymbol{L}' = \boldsymbol{L}\mathbf{A}$") and an inner product (the claim "$y = \langle \boldsymbol{L}\mathbf{A}, \boldsymbol{R}\rangle$"). Somehow, we need to account for the cryptographic commitment $\boldsymbol{C}$ of Hyrax-VMV in the communication model. Moreover, since parties are computationally unbounded in the communication model, there is no such thing as a cryptographic commitment. Therefore, we first must understand what the commitment $\boldsymbol{C}$ is actually contributing in Hyrax-VMV.

The commitment $\boldsymbol{C} = \mathbf{A}\boldsymbol{g}^\top$ is actually a random fingerprint of $\mathbf{A}$ using public randomness $\boldsymbol{g}$, which cryptographic verifier uses to cross-check the proof. Traditionally, random fingerprinting of a matrix $\mathbf{A}$ consists of computing $\mathbf{A}\boldsymbol{r}^\top$ for $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$ (see [Fre77]), where the fingerprint is computed and given by a *trusted* party. It is not difficult to see in Hyrax-VMV that if the (untrusted) prover were to commit to $\mathbf{A}$ with a public random vector $\boldsymbol{r} \in \mathbb{F}^b$, then Hyrax-VMV no longer remains sound via the following attack. The prover can honestly send $\boldsymbol{C} = \mathbf{A}\boldsymbol{r}^\top$ and later convince the verifier that $\widetilde{y} = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$ for some $\widetilde{y} \neq y$ (the true value of $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$) by computing $\widetilde{\boldsymbol{L}} \neq \boldsymbol{L}\mathbf{A}$ such that $\langle \boldsymbol{L}, \boldsymbol{C}\rangle = \langle \widetilde{\boldsymbol{L}}, \boldsymbol{r}\rangle$ and $\langle \widetilde{\boldsymbol{L}}, \boldsymbol{R}\rangle \neq \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$.

However, in Hyrax-VMV, the discrete-log assumption prevents this attack and allows the prover to compute the fingerprint $\mathbf{A}\boldsymbol{r}^\top$ *without explicitly knowing* $\boldsymbol{r}$. The commitment $\boldsymbol{C} = \mathbf{A}\boldsymbol{g}^\top$ acts exactly like a random fingerprint of the matrix $\mathbf{A}$, where the randomness $\boldsymbol{r}$ is hidden within $\boldsymbol{g}$. To see this, we can reimagine sampling the random generators $\boldsymbol{g}$ as follows: (1) sample generator $g \in \mathbb{G}$; (2) sample random vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$; (3) define $\boldsymbol{g}_i = g \cdot \boldsymbol{r}_i$ for all i. Since g is a generator, each $\boldsymbol{g}_i$ is a generator. Now, by hardness of discrete-log, the prover cannot uncover the hidden $\boldsymbol{r}$, which allows the prover to provide the fingerprint of $\mathbf{A}$ without explicit access to the randomness, avoiding the above attack.

By this above discussion, we conclude that in the OMA protocol for Hyrax-VMV, Bob will need a random fingerprint of the matrix $\mathbf{A}$ in order to cross-check Merlin's claims. Moreover, the randomness used for this fingerprint must be hidden from Merlin. This is handled by having Alice and Bob sample shared private randomness $\boldsymbol{r}$ and having Alice send the fingerprint $\boldsymbol{C} = \mathbf{A}\boldsymbol{r}^\top$ after Merlin's interaction with Bob. Intuitively, Alice being honest and sending a correct fingerprint is capturing the cryptographic prover's hands being tied by the discrete-log assumption and the knowledge soundness of Hyrax-VMV: if the prover tries to cheat, he either has to break the discrete-log assumption, or we can extract $\mathbf{A}$ from an accepting transcript with a dishonest prover by the knowledge soundness property.

We now give the full OMA protocol for Hyrax-VMV in Figure 1 and discuss the protocol here. First, in Hyrax-VMV, let $\mathbf{A} \in \mathbb{F}^{a \times b}$ be the matrix that the prover has, as its private input, and let $\boldsymbol{L} \in \mathbb{F}^a$, $\boldsymbol{R} \in \mathbb{F}^b$ be the (public) inputs of the verifier. In the OMA protocol, Alice is given $\mathbf{A}$, Bob is given $\boldsymbol{L}, \boldsymbol{R}$, and Merlin receives both of these inputs. Before the start of Bob and Merlin's interaction, Alice and Bob sample shared private

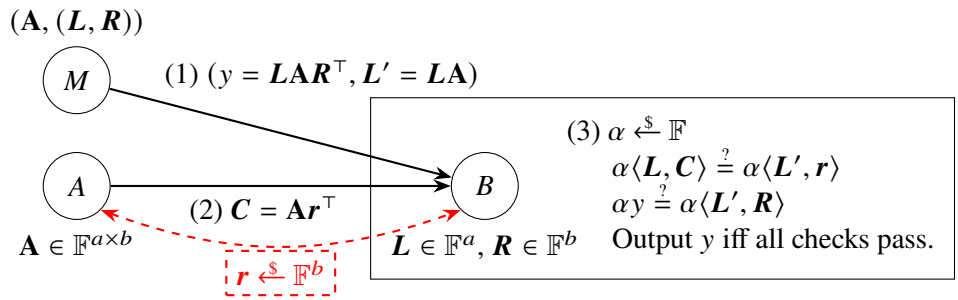


Figure 1: Hyrax-VMV as an OMA communication protocol. Dashed/red lines indicate private shared randomness that is independent of parties' inputs. The protocol executes steps (1) through (3) in order.

randomness $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$. Now, intuitively, the OMA protocol executes Hyrax-VMV in reverse: the protocol begins with Merlin sending $(y = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top, \boldsymbol{L}' = \boldsymbol{L}\mathbf{A})$ to Bob. Then, Alice sends the fingerprint of her input $\boldsymbol{C} = \mathbf{A}\boldsymbol{r}^\top$ to Bob. Finally, Bob performs the same verification procedure as the Hyrax-VMV verifier.

How does this OMA protocol capture Hyrax-VMV? As mentioned previously, the cryptographic protocol is utilizing a random fingerprint, hidden within the discrete-log of $\boldsymbol{g}$, to force the prover to send the correct values; i.e., the soundness of the protocol relies on the randomness hidden within $\boldsymbol{g}$. In the described OMA protocol, Alice plays the role of the committer, and Merlin plays the role of the prover. Crucially, Merlin *does not know the vector* $\boldsymbol{r}$, as it is private randomness shared between Alice and Bob. This ties Merlin's hands: if a dishonest Merlin is trying to convince Bob that $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top = \widetilde{y}$ for some incorrect value $\widetilde{y}$, then with high probability (over the choice of $\boldsymbol{r}$) Bob will reject the interaction. See Section Section 4 for full details.

Evidence for Optimality of the Hyrax-PCS Verifier via OMA Lower Bounds. With Hyrax-VMV framed as an OMA protocol (which is the most difficult part of our results), we can leverage the rich line of works lower bounding communication protocols to lower bound the amount of work needed by the Hyrax-VMV verifier. As stated previously, Hyrax-VMV and its OMA protocol simultaneously prove a matrix-vector product ($\boldsymbol{L}' = \boldsymbol{L}\mathbf{A}$) and an inner product ($y = \langle \boldsymbol{L}', \boldsymbol{R} \rangle$). In Figure 1, let hcost denote the number of bits Merlin sends to Bob, and let vcost denote the number of bits Alice sends to Bob (note here we are not counting the shared randomness in the costs). By prior results, we have the following lower bounds: (1) For $\mathbf{A} \in \mathbb{F}^{a \times b}$ and $\boldsymbol{x} \in \mathbb{F}^b$, any OMA protocol for verifying $f(\mathbf{A}, \boldsymbol{x}) = \mathbf{A}\boldsymbol{x}^\top$ requires $\mathsf{hcost} \cdot \mathsf{vcost} = \Omega(\min(a, b)^2)$ [CMT13] (see Theorem 3.4); (2) For $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^m$, any OMA protocol for verifying $f(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{u}, \boldsymbol{v} \rangle$ requires $\mathsf{hcost} \cdot \mathsf{vcost} = \Omega(m)$ [Kla03, CCM+14] (see Theorem 3.5).

Let the total cost of the OMA protocol be defined as $\mathsf{hcost} + \mathsf{vcost}$. Setting $m = b$, we minimize $\mathsf{hcost} + \mathsf{vcost}$ subject to $\mathsf{hcost} \cdot \mathsf{vcost} = \Omega(\max(b, \min(a, b)^2))$. With a bit of manipulation and calculus, we can minimize this as $\mathsf{hcost} + \mathsf{vcost} = \Omega(\sqrt{\max(b, \min(a, b)^2)}) = \Omega(\max(\sqrt{b}, \min(a, b)))$. This implies that the cryptographic verifier in Hyrax-VMV must perform $\Omega(\max(\sqrt{b}, \min(a, b)))$ bit-operations. Now, for Hyrax-PCS, let $N = 2^n$ be the instance size (i.e., a PCS for n-varite multilinear polynomials or degree $N - 1$ univariate polynomials). Then, Hyrax-PCS sets $a = b = \sqrt{N}$, which yields a lower bound of $\mathsf{hcost} + \mathsf{vcost} = \Omega(\sqrt{N})$. This gives us strong evidence that the Hyrax-PCS verifier must perform $\Omega(\sqrt{N})$ bit-operations. Notice that in Hyrax-VMV, the verifier computes 2 inner products over $\mathbb{G}$ of length a and one inner product over $\mathbb{F}$ of length b, leading to $O(\sqrt{N})$ $\mathbb{G}$-operations in Hyrax-PCS, strongly suggesting that the verifier is optimal up to polylog $|\mathbb{G}|$ factors; see Theorem 4.1 and Corollary 4.2 in Section 4 for full details.

1.2.4 Bulletproofs: Review and Information-Theoretic Core

The heart of the Bulletproofs polynomial commitment scheme is a proof of knowledge (PoK) of an opening to a Pedersen vector commitment. Given random generators $\boldsymbol{g} \xleftarrow{\$} \mathbb{G}^N$ as public parameters, the prover gives some commitment C and proves it knows a vector $\boldsymbol{w} \in \mathbb{F}^N$ such that $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$. Bulletproofs utilizes a "split-and-fold" recursive technique to continuously reduce the claim $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$ for length N vectors $\boldsymbol{g}, \boldsymbol{w}$ to a new claim $C' = \langle \boldsymbol{g}', \boldsymbol{w}' \rangle$ for length $N/2$ vectors $\boldsymbol{g}', \boldsymbol{w}'$, until the final vectors are of length 1 and the prover simply sends the final "folded" witness. Here, $\boldsymbol{g}'$ and $\boldsymbol{w}'$ are a random linear combination of the left and right halves of the (respective) vectors $\boldsymbol{g}$ and $\boldsymbol{w}$, where the randomness is provided by the verifier during the interaction. Bulletproofs readily extend to a PCS for any polynomial where polynomial evaluation can be expressed as an inner product (e.g., multilinear and univariate polynomials); see Section 5.3 for full details.

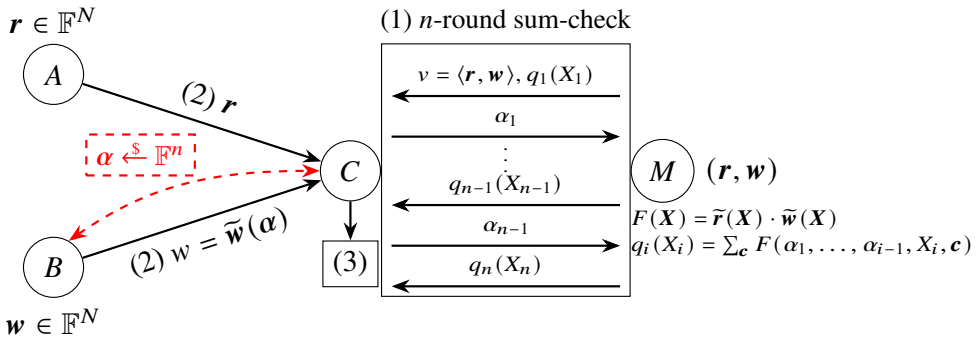
Bootle et al. [BCS21] originally observed that the Bulletproofs PoK is an instantiation of the sum-check protocol [LFK+92] for proving the inner product of two vectors $\boldsymbol{w}, \boldsymbol{g}$. The sum-check proves $C = \sum_{\boldsymbol{c} \in \{0,1\}^n} \widetilde{\boldsymbol{g}}(\boldsymbol{c}) \widetilde{\boldsymbol{w}}(\boldsymbol{c})$, where $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{g}}$ are appropriately defined multilinear extensions of $\boldsymbol{w}$ and $\boldsymbol{g}$, respectively.[1] The Bulletproofs sum-check allows us to do something impossible in an interactive proof: it allows the prover and verifier to run the sum-check with respect to a hidden vector $\boldsymbol{r}$. As with our discussion of Hyrax, this vector $\boldsymbol{r}$ is hidden in the group generators $\boldsymbol{g}$. But because of hardness of discrete-log, $\boldsymbol{r}$ is hidden, and due to group homomorphism, the prover and verifier can still operate the sum-check over the claim $v = \langle \boldsymbol{r}, \boldsymbol{w} \rangle$ without either party receiving access to $\boldsymbol{r}$ directly.

As just mentioned, this is *impossible* in an information-theoretic setting: any information-theoretic prover needs the vector $\boldsymbol{r}$ to perform the sum-check protocol. In Hyrax, the hidden random vector in $\boldsymbol{g}$ was utilized as a random fingerprint that the prover had no control over, and when translating Hyrax to a communication protocol, we preserved this property: the randomness was hidden from Merlin. However, any interactive proof or communication protocol for proving $\langle \boldsymbol{w}, \boldsymbol{r} \rangle$ needs to give the prover (or Merlin) explicit access to $\boldsymbol{r}$, else they could never run the protocol in the first place!

This leads us to the following question: if the vector hidden in $\boldsymbol{g}$ of Bulletproofs isn't operating as a random fingerprint hidden from the prover, what is the "crypto" actually achieving here? To answer this question, we first recall what is needed to achieve soundness in the sum-check protocol. Consider a sum-check protocol for proving $C = \langle \boldsymbol{w}, \boldsymbol{r} \rangle$, which performs the sum-check over the polynomial $F(\boldsymbol{X}) = \widetilde{\boldsymbol{r}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X})$ (the multilinear extensions of $\boldsymbol{r}, \boldsymbol{w}$). Crucial to soundness of the sum-check protocol is that the verifier can evaluate $F(\boldsymbol{\alpha})$ at some random point $\boldsymbol{\alpha}$ at the end of the protocol, which in our context means that the verifier needs to evaluate $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ and $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$.

In Bulletproofs, the verifier can compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ implicitly by computing $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ (i.e., $\boldsymbol{r}$ is hidden in the exponents of $\boldsymbol{g}$), but *cannot* compute $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$, as $\boldsymbol{w}$ is the prover's private input. Instead, the prover sends a value w which is claimed to be $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. In the information-theoretic setting, such a sum-check protocol would be completely unsound, as the prover can always send some w^* to pass the final check of the sum-check protocol: the final check is of the form $C^* \stackrel{?}{=} \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha}) \cdot w^*$, and the prover knows both C^* and $\widetilde{\boldsymbol{r}}(\alpha)$ before sending w^*, so can trivially pass this check. However, in the cryptographic setting, the discrete-log assumption gives us the powerful knowledge soundness property: in order to send some w^* which would violate soundness of Bulletproofs, the cryptographic prover would need to break the discrete-log assumption, or one could extract the entire witness $\boldsymbol{w}$ from the transcript of the protocol. Thus, the cryptographic prover is forced to send the correct value $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. This tells us that any information-theoretic core of Bulletproofs must enforce this invariant: the verifier must receive the correct and honest $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$.

[1] [BCS21] utilize a weighted sum-check protocol and non-standard definitions for multilinear extensions; the reason seems to be so that the resulting sum-check protocol is identical to the original Bulletproofs protocol.



(3) C does the following.

- Set $v_1 = v$.
- For each $i \in [n]$: (a) Check $v_i = q_i(0) + q_i(1)$ and $\deg(q_i) \leqslant 2$; (b) Set $v_{i+1} = q_i(\alpha_i)$.
- Compute $r = \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$. Check $v_{i+1} = r \cdot w$. Accept (i.e., output v_1) if and only if all checks pass.

Figure 2: BP-Sum-Check as an n-round SMH protocol for $N = 2^n$. The red dashed line indicates private randomness shared between the respective parties. The protocol executes (1), followed by (2), followed by (3).

Information-Theoretic Core of Bulletproofs. With this in mind, we turn to modeling Bulletproofs as a communication protocol. However, the OMA model, which we used for Hyrax, will not suffice (see the end of this section and Section 5.4 for discussion). Instead, we will need a new communication model to properly model Bulletproofs (and Dory in Section 1.2.5). This model, which we call the *simultaneous messaging with a helper* (SMH) model, is a generalization of the OMA model with four parties: Alice, Bob, Charlie, and Merlin. Alice has input a, Bob has input b, Merlin receives (a, b), and the goal is to help Charlie compute $f(a, b)$. Notice here that Charlie has no input; moreover, Alice and Bob cannot speak to each other and can only send a single message to Charlie. Meanwhile, Merlin and Charlie can interact via an interactive proof. This model is readily captured by OMA model where "Bob" in the OMA protocol is simply a super player of Bob and Charlie (or Alice and Charlie) in the SMH protocol.

With our new model in hand, we can specify a SMH communication protocol for Bulletproofs. We give the protocol in Figure 2 and give an overview here. As with Hyrax, framing Bulletproofs as a communication protocol rearranges the execution of the protocol. To begin, for $N = 2^n$, Alice has input $\boldsymbol{r} \in \mathbb{F}^N$ and *is deterministic* (we'll come back to this later); Bob has input $\boldsymbol{w} \in \mathbb{F}^N$ and additionally shares private randomness $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}^n$ with Charlie; Merlin receives both $(\boldsymbol{r}, \boldsymbol{w})$ but does not receive $\boldsymbol{\alpha}$. Then, the communication protocol begins by having Charlie and Merlin execute the sum-check protocol to prove that $\langle \boldsymbol{r}, \boldsymbol{w} \rangle = v$ for some value $v \in \mathbb{F}$. At the end of the sum-check protocol, Alice sends Charlie the whole vector $\boldsymbol{r}$ and Bob sends Charlie $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ (the multilinear extension of $\boldsymbol{w}$ evaluated at $\boldsymbol{\alpha}$). Finally, with all of this information, Charlie can verify whether Merlin was honest during the execution of the sum-check protocol.

Why does this capture Bulletproofs? As we discussed previously, Bulletproofs is the sum-check protocol where the verifier computes $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ implicitly from the public randomness $\boldsymbol{g}$, and *honestly* receives $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ from the prover, as otherwise, either the discrete-log assumption is broken or knowledge soundness allows us to extract the entire witness $\boldsymbol{w}$ from the interaction. In our SMH protocol, we model the verifier receiving this correct $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ by having the honest party Bob send it after Charlie and Merlin's interaction. Key here is that the randomness $\boldsymbol{\alpha}$ is hidden from Merlin before the start of the sum-check, otherwise the protocol is unsound. Moreover, Bob and Charlie need this shared randomness, otherwise if Charlie used some different randomness during the sum-check (say some $\boldsymbol{\beta}$), then Bob would have no way of sending $\widetilde{\boldsymbol{w}}(\boldsymbol{\beta})$ to Charlie.

As with the Bulletproofs verifier, Charlie must also compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ from Alice's input $\boldsymbol{r}$. Here, this is trivial as Alice sends $\boldsymbol{r}$ to Charlie. In particular, we believe this is the correct modeling of Bulletproofs; i.e., Alice is deterministic and simply sends her entire input. We have two main reasons for this belief. First, in the actual prescribed Bulletproofs protocol, the verifier simply reads $\boldsymbol{g}$ from the public parameters and computes $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ locally after running the sum-check protocol with the prover. Second, consider what it would mean if Alice, Bob, and Charlie all shared the random string $\boldsymbol{\alpha}$. In this case, Alice could send $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ to Charlie at the end of the protocol yielding very succinct communication. However, how would this translate to the cryptographic setting? It is not clear; at best, this seems to be a variant of Bulletproofs with a trusted setup that includes $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ with $\boldsymbol{\alpha}$ known by the verifier then later revealed to the prover during the interactive protocol. Such a protocol is inherently different from Bulletproofs. Another possibility is to ask Merlin in the SMH protocol to send $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ as the last message, but the information-theoretic dishonest Merlin can construct a purported value r^* to verify any the entire transcript, even if Bob sends the correct $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. Moreover, such a Merlin sending $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ does not match the prescribed Bulletproofs prover, leading to a mismatch between the cryptographic protocol and the information-theoretic core. See Section 5 for full discussion.

All together, Charlie in our SMH protocol for Bulletproofs gets from Alice and Bob all the necessary information needed in order to verify the sum-check protocol executed with Merlin. Charlie receives $\boldsymbol{r}$ and $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ from Alice and Bob, respectively, computes $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$, and uses this to verify the final claim of the sum-check protocol. This establishes the necessary conditions for the soundness of the sum-check protocol, yielding a valid SMH protocol for the inner product function.

Evidence for Optimality of the Bulletproofs-PCS Verifier via New SMH Lower Bounds. With our SMH protocol for Bulletproofs established, we now turn towards our goal of lower bounding the communication needed for parties in our protocol. However, the SMH model is a new communication model we introduce, so there are no prior known lower bounds we can leverage. Therefore, we prove new lower bounds on the communication complexity of computing inner product in the SMH model.

Theorem 1.2 (Informal; see Theorem 3.10)**.** *Any SMH protocol for computing $f(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for vectors of length n such that Alice is deterministic requires Alice to send $\Omega(n)$ bits.*

Intuitively, the above proof follows since if Alice has input $\boldsymbol{x}$ and sends a $n-1$ bit message m_A, then Merlin can find another vector $\boldsymbol{x}' \neq \boldsymbol{x}$ and $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \neq \langle \boldsymbol{x}', \boldsymbol{y} \rangle$ (for Bob's input $\boldsymbol{y}$) such that if Alice has input $\boldsymbol{x}'$, her message would *also* be m_A, due to the Pigeonhole Principle and because Alice is deterministic. So Charlie would accept the same interaction with Merlin if Alice had input $\boldsymbol{x}$ or $\boldsymbol{x}'$, violating soundness of the protocol.

Letting hcost denote the number of bits exchanged between Charlie and Merlin, bcost denote the number of bits Bob sends to Charlie, and acost denote the number of bits Alice sends to Charlie. In our SMH protocol for Bulletproofs, we have specified that Alice is deterministic, thus in accordance with the above lower bound, we have $\mathsf{acost} = \Omega(N)$, which implies that $\mathsf{hcost} + \mathsf{acost} + \mathsf{bcost} = \Omega(N)$. This gives us strong evidence that the Bulletproofs verifier must perform $\Omega(N)$ bit-operations, otherwise we could translate the Bulletproofs verifier doing $o(N)$ work a communication protocol where Alice sends $o(N)$ bits but remains deterministic, violating our lower bound. Now, prescribed Bulletproofs verifier reads $O(N \log |\mathbb{G}|)$ bits (the vector $\boldsymbol{g}$) from the public input, then computes $\boldsymbol{g}(\boldsymbol{\alpha})$, which can be done in roughly $N \log |\mathbb{G}| / (\log(N) + \operatorname{loglog} |\mathbb{G}|)$ group operations via Pippenger's algorithm [Pip80], which gives roughly $O(N \operatorname{polylog} |\mathbb{G}|)$ bit-operations. This strongly suggests that the Bulletproofs verifier is optimal up to $\operatorname{polylog} |\mathbb{G}|$ factors; in particular, there is strong evidence to support that the Bulletproofs verifier is inherently *linear*.

More Evidence of Optimality: Framing an Attack on Prior Work in Our Model. As mentioned previously, two withdrawn papers [Lu22, Lu24] attempted to obtain a sublinear Bulletproofs verifier. This

was done by attempting to augment Bulletproofs by changing how the "foldings" were performed, and by having the prover send some additional information to help the verifier efficiently compute the final folding of the public parameters $\boldsymbol{g}$ in sublinear time. However, Gabizon [Gab24] showed an attack against the protocol where the prover could commit to some vector $\boldsymbol{w}$ and open to some *other vector* $\boldsymbol{c} \neq \boldsymbol{w}$. Such an attack directly translates to the proof of our SMH lower bound: in the proof of Theorem 1.2 (see Theorem 3.10), if Alice sends less than n bits when she is deterministic and her input $\boldsymbol{x}$ is of length n, Merlin can find another $\boldsymbol{x}' \neq \boldsymbol{x}$ such that Alice's message on input $\boldsymbol{x}$ or $\boldsymbol{x}'$ remains the same.

Why Does OMA Not Suffice for Bulletproofs? A natural question to ask is why did we introduce our new SMH communication model instead of framing Bulletproofs as an OMA model protocol, as we did with Hyrax. We give a high-level overview on why we believe the OMA model is not the correct model for capturing Bulletproofs; see Section 5.4 for the full discussion.

In the OMA model, we only have 2 parties, Alice and Bob, and the helper Merlin. The first thing to think about in this model is which parties receive which inputs? In Figure 2, there is the input $\boldsymbol{w}$, representing the cryptographic prover's private witness, and there is $\boldsymbol{r}$, representing the "hidden" group exponents. Now, in an OMA protocol, Bob plays the role of the cryptographic verifier, and we want to translate the amount of bits communicated to Bob to a lower bound on the time the cryptographic verifier needs for verification.

We have two possible scenarios: (1) Alice has input $\boldsymbol{r}$ and Bob has input $\boldsymbol{w}$; (2) Alice has input $\boldsymbol{w}$ and Bob has input $\boldsymbol{r}$. For (1), we do not believe this makes sense for modeling the cryptographic verifier, because it is, in essence, giving the verifier the prover's private witness $\boldsymbol{w}$. In the cryptographic protocol, if the verifier has this witness, then there is no need for the verifier to even talk to the prover!

So, intuitively, that only leaves us with situation (2), where Alice has input $\boldsymbol{w}$ and Bob has input $\boldsymbol{r}$. Here, we have the choice of letting Alice be deterministic or randomized. In the deterministic case, we actually run into the same lower bound as Theorem 1.2, but in the OMA model (see Corollary 3.11): Alice will need to send her entire input to Bob. But, this is now just the trivial proof system where the prover sends his witness $\boldsymbol{w}$, so the verifier does not need to interact with the prover in the first place. Now suppose we let Alice and Bob share randomness $\boldsymbol{\alpha}$, Bob and Merlin run the sum-check protocol (i.e., Bulletproofs), and Alice sends $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ to Bob. This communication protocol sends $O(\log(N))$ field elements, which is much less than the linear lower bound we are trying to argue. Notably, Bob still needs to compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ in this case, but we can no longer capture the cost of this computation using communication complexity.

For these reasons, which are more deeply explored in Section 5.4, we do not believe that the OMA model is the right model for capturing the verifier time of Bulletproofs.

1.2.5 Dory: Review and Information-Theoretic Core

The heart of Dory is nearly identical to Bulletproofs: it is a proof of knowledge of an opening to a Pedersen vector commitment, but now over a *pairing-friendly group* $\mathbb{G}$;[2] such a commitment, due to Abe et al. [AFG+16, Gro09], is also known as an AFGHO commitment. Given random generator $\boldsymbol{g} \xleftarrow{\$} \mathbb{G}^N$ as public parameters, the prover gives some commitment C and proves it knows a vector $\boldsymbol{w} \in \mathbb{G}^N$ such that $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle_e := \sum_i e(\boldsymbol{g}, \boldsymbol{w}) \in \mathbb{G}_T$, where $e(\cdot, \cdot)$ is the pairing function of $\mathbb{G}$ and $\mathbb{G}_T$ is the target group.

Pairings are used to enable *logarithmic verification*, as pairings allow parties to commit to elements of $\mathbb{F}$ *and* $\mathbb{G}$. In particular, pairings allow Dory to achieve what Bulletproofs could not: to force the cryptographic prover to send the correct value of $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ to the verifier, whereas in Bulletproofs, the verifier needed to compute

[2] For our purposes, it suffices to only consider symmetric pairings. See Section 2 for a review of pairing-friendly groups and Section 6 for a full discussion of Dory.

this value themselves. Dory achieves this as follows. The first round of Dory is identical to the first round of Bulletproofs: using verifier randomness, the claim $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle_e$ about vectors of size N is reduced to a new claim $C' = \langle \boldsymbol{g}', \boldsymbol{w}' \rangle_e$ about vectors of size $N/2$, where $\boldsymbol{g}'$ and $\boldsymbol{w}'$ are random linear combinations of the left and right halves of $\boldsymbol{g}, \boldsymbol{w}$. Since we are operating over a pairing-friendly group $\mathbb{G}$, we can force the prover to *commit to* $\boldsymbol{g}'$ using some new (public) commitment key $\boldsymbol{\Gamma} \in \mathbb{G}^{N/2}$, resulting in a new claim $D = \langle \boldsymbol{g}', \boldsymbol{\Gamma} \rangle$. At the start of the second round of Dory, the prover sends this new claim D to the verifier, and then the prover and verifier run two instances of the Bulletproofs PoK protocol in parallel for C' and D. This process continues every round, requiring a new public commitment key per round and resulting in i parallel invocations of the Bulletproofs PoK for inputs of size $N/2^{i-1}$ in round i.

At the end of the protocol, the verifier has $n + 1$ claims to verify for $N = 2^n$. Assuming that the intermediate commitment keys (e.g., $\boldsymbol{\Gamma}$) are part of the public parameters, logarithmic verification is achievable with (linear-time) *preprocessing*. Let $\boldsymbol{g} = \boldsymbol{\Gamma}^{(1)}$ and let $\boldsymbol{\Gamma}^{(i)} \in \mathbb{G}^{N/2^{i-1}}$ be the commitment key for round i of the protocol. Then, preprocessing of the form $(\Delta_{i,L}, \Delta_{i,R}) = (\langle \boldsymbol{\Gamma}_L^{(i)}, \boldsymbol{\Gamma}^{(i+1)} \rangle_e, \langle \boldsymbol{\Gamma}_R^{(i)}, \boldsymbol{\Gamma}^{(i+1)} \rangle_e)$ gives the verifier enough information to perform only a *logarithmic* amount of work to verify the interaction, where the subscripts L, R respectively denote the left and right halves of the vectors.

The keen-eyed readers may notice that the above protocol actually has $O(\log^2(N))$ proof size/verification time; indeed, the above protocol is a simpler variant of Dory which we call *Proto-Dory*.[3] Obtaining the full Dory protocol with logarithmic verification from Proto-Dory essentially amounts to taking a random linear combination of the new claim introduced in round i with the claim from the previous round; we refer the reader to Section 6 for our treatment of the full Dory protocol and to [Tha22] for an excellent overview of both these protocols as this overview, while sufficient for understanding Dory at a high-level, is oversimplified and removes many of the subtle (and necessary) details for the actual protocol. However, for the purposes of the overview, Proto-Dory suffices for understanding the information-theoretic core of Dory. We also remark that, like Bulletproofs, (Proto-)Dory readily yields a PCS for any polynomials that can express polynomial evaluation as an inner product (e.g., multilinear and univariate); see Section 6.3 for full details.

IT Core of Proto-Dory. As with Bulletproofs, we model Proto-Dory as a communication protocol in our new SMH model. From the above discussion, Proto-Dory is many invocations of Bulletproofs in parallel over pairing-friendly groups. In particular, in round i of the protocol, the prover and verifier are running i parallel Bulletproofs invocations over vectors of size $N/2^{i-1}$. This observation allows us to make a new connection: Proto-Dory is *also* an instantiation of the sum-check protocol (or, rather, many parallel invocations of the sum-check). To the best of our knowledge, we are the first to make this connection and observation explicit; moreover, in Section 6, we show how the full Dory protocol is also an instantiation of (a slight variant) of the standard sum-check protocol.[4]

This yields a clear way to model Proto-Dory as a communication protocol, which we present in Figure 3 and review here. Proto-Dory looks nearly identical to Bulletproofs (Figure 2), with a crucial difference that Alice is now *randomized*. First, Bob and Charlie both share a private random string $\boldsymbol{\alpha}$; this is the sum-check randomness used by Charlie and enables Bob to send $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ at the end of the protocol. Now, the randomization of Alice is capturing the preprocessing of Proto-Dory that both the prover needs to run the sum-check, and the verifier needs to verify the sum-check. The vectors $\boldsymbol{\gamma}^{(i+1)}$ are taking the role of the commitment keys in Proto-Dory, whereas the message sent by Alice to Charlie represents the preprocessing (expressed as polynomials) performed before the start of the cryptographic protocol (during the setup phase).

Why does this capture Proto-Dory? We first focus on Bob and Charlie's specification. (Proto-)Dory is

[3] Proto-Dory actually appears in [Lee21] as a build-up to the "full" Dory protocol.

[4] In this light, it is possible to apply the results of [BCS21] to Dory; however, we leave this as future work to explore.

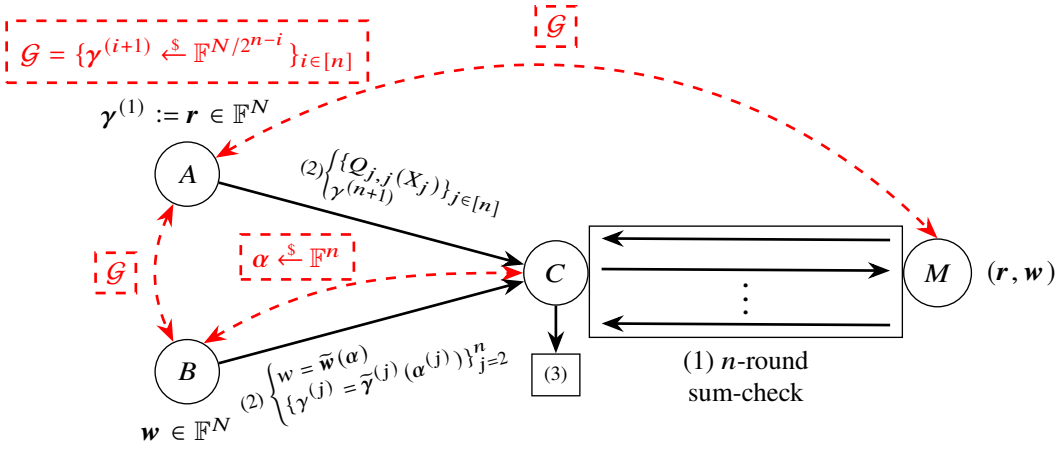
also a proof of knowledge under the discrete-log assumption in pairing-friendly groups. At the end of the actual protocol (see Algorithm 6.1), as with Bulletproofs the prover provides the value w claimed to be $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$, which the verifier needs in order to verify the sum-check protocol since there is no oracle access to $\widetilde{\boldsymbol{w}}$ and the verifier cannot evaluate this polynomial at random points. By the proof of knowledge and discrete-log assumption (in pairing-friendly groups), the cryptographic prover is forced to send the correct w. In the communication setting, this correct value is enforced by having Bob send it; to do so, Bob and Charlie need to share the randomness ahead of time. Second, the interaction between Charlie and Merlin is exactly the interaction that occurs in Proto-Dory: during round i of the sum-check, Merlin and Charlie are running i invocations of the sum-check protocol in parallel. In Figure 3, the polynomial S_i is the "main" sum-check polynomial for verifying the inner product $\langle \boldsymbol{r}, \boldsymbol{w} \rangle$, whereas the $(i-1)$ polynomials $Q_{j,i}$ for $j \in [i-1]$ are the parallel sum-check invocations. Third, Alice's message to Charlie is exactly the preprocessing performed during the setup phase, just expressed as polynomials, which Charlie needs to verify the sum-check claims from the interaction with Merlin.

Now, why do Alice, Bob, and Merlin share random $\boldsymbol{\gamma}^{(i)}$ for all $i \in [n]$? In the actual (Proto-)Dory protocol (see Algorithms 6.1 and 6.2), these vectors are actually recursively defined as the left halves of the previous vector: i.e., $\boldsymbol{\gamma}^{(i+1)} = \boldsymbol{\gamma}^{(i)}_L$ for $i \in [n]$, where the subscript L denotes the left half of the vector. First, if we let Alice share these vectors in this way, then Alice is now *deterministic*, and we run into the lower bound of Theorem 1.2. Moreover, under such a setting, we are able to demonstrate an attack that Merlin can perform to violate the soundness of the protocol (in line with the proof of Theorem 3.10, see Remark 6.5 for details). Finally, this violates the very model we are introducing, where we enforce that shared randomness be independent of the parties' inputs. So these vectors need to be random in order for the SMH protocol to be sound and in order for Merlin to execute the parallel sum-check with Charlie. Another natural question, as with Bulletproofs, is why do Alice, Bob, and Charlie not simply share $\boldsymbol{\alpha}$? As mentioned in Section 1.2.4, it is not clear how this translates to the cryptographic setting, as it would require the setup of Dory to have the vector $\boldsymbol{\alpha}$ hidden and only known to the verifier before running the proof of knowledge with the prover, leading to some trusted setup that later becomes public.

The final discrepancy to discuss is Merlin's final message to Charlie and Bob's new messages. First, we additionally require Merlin to include in his final message a degree-1 polynomial $\hat{g}$, claimed to be the polynomial $\widetilde{\boldsymbol{\gamma}}^{(1)}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$, the multilinear extension of Alice's input. Second, we additionally require Bob to send $\boldsymbol{\gamma}^{(j)} = \widetilde{\boldsymbol{\gamma}}^{(j)}(\alpha_j, \ldots, \alpha_n)$ for $j \in \{2, \ldots, n\}$. Intuitively, Bob's messages are again capturing the proof of knowledge property of the protocol: these values, in the cryptographic protocol, are forced to be sent honestly by the prover, unless the discrete-log assumption is broken, or we can extract the prover's full witness from the protocol transcript. However, Bob can send every value honestly except for $\gamma^{(1)} = \widetilde{\boldsymbol{\gamma}}^{(1)}(\boldsymbol{\alpha})$, as this value depends solely on Alice's input. If Bob could compute this value, Bob would need Alice's entire input and make the problem trivial: Bob just sends $\langle \boldsymbol{w}, \boldsymbol{\gamma}^{(1)} \rangle$ to Charlie.

Therefore, we delegate the responsibility of providing $\gamma^{(1)}$ to Merlin in the SMH protocol. We do this by forcing Merlin to send the polynomial $\hat{g}(X_n)$: crucially, Merlin sends this before ever knowing the value α_n, so (by soundness of the sum-check protocol), the only way for Merlin to pass the final verification checks performed by Charlie if $\hat{g}$ is dishonest is for Charlie to sample some bad randomness. This happens with probability at most $1/|\mathbb{F}|$, so soundness remains in the protocol, effectively forcing Merlin to send the correct value, otherwise be caught with high probability.

Evidence for Optimality of the Dory-PCS Verifier. While Proto-Dory is a $(\log(N) + 1)$-round protocol with $O(\log^2(N))$ verification time, the full Dory protocol is $(\log(N) + 1)$-rounds with $O(\log(N))$ verification time (ignoring polylog $|\mathbb{G}|$ factors). In both cases, when extending Proto-Dory and Dory to a polynomial



(0) B and C sample shared private randomness $\alpha \in \mathbb{F}^n$. A, B, and M sample shared private randomness $\gamma^{(i+1)} \in \mathbb{F}^{N/2^{n-i}}$ for $i \in [n]$.
(1) M and C engage in the n-round sum-check protocol over the polynomial $F(X) = \widetilde{r}(X)\widetilde{w}(X)$. For $i \in [n]$:

- For all $j \in [i-1]$, M defines $S_i(X_i) = \sum_{c} F(\alpha_1, \ldots, \alpha_{i-1}, X_i, c)$, $P_{i-1}(X_{[i-1,n]}) = \widetilde{\gamma}^{(i-1)}(X_{[i-1,n]})\widetilde{\gamma}^{(i)}(X_{[i,n]})$, and $Q_{j,i}(X_i) = \sum_{c} P_j(\alpha_j, \ldots, \alpha_{i-1}, X_i, c)$.
- M sends $S_i(X_i)$ and $\{Q_{j,i}(X_i)\}_{j\in[i-1]}$. If $i = 1$, then send $v = \langle r, w\rangle$. If $i = n$, then send $\hat{g}(X_n) = \widetilde{\gamma}^{(1)}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$.
- C sends α_i.

(2) A computes $Q_{j,j}(X_j) = \sum_{c\in\{0,1\}^{n-j}} \widetilde{\gamma}^{(j)}(X_j, c) \cdot \widetilde{\gamma}^{(j+1)}(c)$ for all $j \in [n]$ and sends $(\{Q_{j,j}(X_j)\}_{j\in[n]}, \gamma^{(n+1)})$. B computes and sends: $\gamma^{(j)} = \widetilde{\gamma}^{(j)}(\alpha^{(j)})$ for $j \in [2, n]$ and $\alpha^{(j)} := (\alpha_j, \ldots, \alpha_n)$, and $w = \widetilde{w}(\alpha)$.
(3) C does the following checks.

- Set $v_1 = v$. For each $i \in [n]$:
  - Check if $v_i \stackrel{?}{=} S_i(0) + S_i(1)$; reject if not.
  - For each $j \in [i-1]$, check $D_{j,i} \stackrel{?}{=} Q_{j,i}(0) + Q_{j,i}(1)$ and $\deg(Q_{j,i}) \leqslant 2$.
  - Set $v_{j+1} = q_j(\alpha_j)$ and $D_{j,i+1} = Q_{j,i}(\alpha_i)$ for $j \in [i]$
- Check that $\deg(\hat{g}) \leqslant 1$ and compute $\gamma^{(1)} = \hat{g}(\alpha_n)$.
- Check if $v^{(n+1)} \stackrel{?}{=} w \cdot \gamma^{(1)}$ and $D_{j,n+1} \stackrel{?}{=} \gamma^{(j)} \cdot \gamma^{(j+1)}$ for all $j \in [n]$.
- Accept (i.e., output v_1) if and only if all checks pass.

Figure 3: Proto-Dory (Algorithm 6.1) as a n-round SMH protocol for $N = 2^n$. The dashed/red lines indicate private randomness shared before C and M's interaction. The protocol executes (1) through (3) in order.

commitment scheme, these costs are only a constant factor larger; in particular, extending to a polynomial commitment scheme just requires a single addition of another parallel sum-check instance in the first round of the protocol. This translates to an n-round SMH protocol nearly identical to Figure 3 (and Figure 7) with some minor modifications (see Section 6.3 for full details). In some sense, our lower bound is now trivially shown: we specify an n-round SMH protocol for computing inner product, and by definition, Charlie and Merlin need to exchange at least 1 bit per round, leading to $\Omega(n)$ communication. This gives us the $\Omega(\log(N))$ lower bound (for $N = 2^n$) for the SMH protocol representing (Proto-)Dory, whereas the verifier time in Dory is $O(\log(N)\,\mathrm{polylog}\,|\mathbb{G}|)$ bit-operations, giving strong evidence that Dory has optimal verifier time.

1.3 Related Work

As we mentioned previously, to the best of our knowledge, we are the first to make the connection between polynomial commitment schemes and communication complexity. Here, we discuss some related communication models and other seemingly related work.

Communication Complexity. The first and most popular communication model is the 2-player model by Yao [Yao79] as described earlier. For a comprehensive discussion on many other communication models, see

the book by Kushivelitz and Nisan [KN97]. We briefly describe the models that are most relevant to our work.

Chandra, Furst, and Lipton [CFL83] extended the 2-player communication model to the k-player or *multi-party* model, where k players communicate with each other to compute a function of their private inputs. Babai, Kimmel, and Lokam [BKL95, BGK+03] introduced the *Simultaneous Message* (SM) model, which can be seen as a restriction of the multi-party model: the players cannot interact with each other but simultaneously send messages to a "referee", who doesn't directly see any of the inputs but generates the output from the players' messages. We note that our SMH model should be seen as an extension of the SM model rather than the general multi-player model: SMH extends the 2-player SM model with a referee by introducing a helper in the standard Merlin-Arthur sense.

We now discuss some other "prover-enhanced" communication models studied in the literature. The *non-deterministic model* [KN97] is an analog of **NP**, where the prover sends a deterministic help message to the verifier (the pair Alice and Bob), who then communicate with each other using a deterministic protocol to compute a function of their inputs. Other variants of Merlin-Arthur communication (where randomization is allowed) include the setting where Merlin first sends a help message to Arthur (represented collectively by two or more players), which we can think of being written on a blackboard shared between the players, who then interact with each other (possibly based on Merlin's message) in unrestricted number of rounds to evaluate the function [Kla03, CCM+14]. Natural generalizations to allow multiple rounds of interaction between Merlin and Arthur have been studied: Chakrabarti et al. [CCM+19] defined the $\mathbf{OMA}[k]$ model that we use in this paper and also study the $\mathbf{OIP}[k]$ model, the online communication analog of $\mathbf{IP}[k]$ similar to $\mathbf{OMA}[k]$, except $\mathbf{OIP}[k]$ allows Bob and Merlin to interact based on Bob's input.

Other Work. A (seemingly) related work by Boyle et al. [BJS+25] introduces the concept of simultaneous-message and succinct secure computation. This concept seems similar to the simultaneous message model [BKL95], as it involves 3 parties, Alice, Bob, and Charlie, where Alice has some large input X, Bob has a small input y, and Charlie wishes to compute $f(X, y)$ while minimizing the bits exchanged between parties. However, this concept is introduced in the context of secure computation, where Alice and Bob wish to keep their inputs private. Moreover, Alice and Bob in this setting are allowed to share messages with each other prior to sending messages to Charlie, which is different from the SM model discussed previously and is therefore not directly related to our work or communication complexity. Another work by Cormode et al. [CDG+24] studies streaming zero-knowledge proofs that use streaming algebraic commitment protocols, which may be adapted to certain communication models using standard streaming-to-communication simulations. However, the motivation behind their work is in designing zero-knowledge streaming protocols for specific problems, which is entirely different from ours.

2 Preliminaries

For $n \in \mathbb{Z}^+$, we let $[n] := \{1, 2, \ldots, n\}$. For a set S, we let $x \xleftarrow{\$} S$ denote the process of sampling an element of S uniformly at random. We let $\mathbb{F}_q$ denote a finite field of size $q = p^a$ for some prime p and $a \in \mathbb{Z}^+$. When clear from context, we simply write $\mathbb{F}$. We let $\mathbb{G}$ denote a finite cyclic group of prime order. For ease of presentation, we express group operations as additive rather than multiplicative. We let lower case bold denote vectors of field and/or group elements; e.g., $\boldsymbol{a} \in \mathbb{F}^n$ or $\boldsymbol{b} \in \mathbb{G}^n$. For any vector $\boldsymbol{a}$ of length n, we let $\boldsymbol{a}_i$ denote the ith element of $\boldsymbol{a}$. For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$, we let $\langle \boldsymbol{a}, \boldsymbol{b} \rangle := \sum_i \boldsymbol{a}_i \cdot \boldsymbol{b}_i \in \mathbb{F}$ denote the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$. Moreover, for group $\mathbb{G}$ with scalar field $\mathbb{F}$, given $\boldsymbol{g} \in \mathbb{G}^n$ and $\boldsymbol{a} \in \mathbb{F}^n$, we let $\langle \boldsymbol{g}, \boldsymbol{a} \rangle = \langle \boldsymbol{a}, \boldsymbol{g} \rangle \in \mathbb{G}$ denote the inner product of $\boldsymbol{g}$ and $\boldsymbol{a}$.

Multilinear Polynomials and Extensions. Let $f\colon \{0,1\}^n \to \mathbb{F}$ be a function. Then the *multilinear extension* of f, denoted as $\widetilde{f}\colon \mathbb{F}^n \to \mathbb{F}$ is the unique polynomial such that $\widetilde{f}(\boldsymbol{c}) = f(\boldsymbol{c})$ for every $\boldsymbol{c} \in \{0,1\}^n$, and the individual degree of $\widetilde{f}$ is at most 1. Such a polynomial can be uniquely defined as

$$\widetilde{f}(X_1, \ldots, X_n) := \sum_{\boldsymbol{c} \in \{0,1\}^n} f(\boldsymbol{c}) \cdot \mathsf{eq}(\boldsymbol{X}, \boldsymbol{c}), \text{ where} \tag{1}$$

$$\mathsf{eq}(\boldsymbol{X}, \boldsymbol{c}) := \prod_{i=1}^{n} X_i c_i + (1 - X_i)(1 - c_i). \tag{2}$$

Here, for binary inputs $\boldsymbol{b}, \boldsymbol{c} \in \{0,1\}^n$, $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{c}) = 1$ if and only if $\boldsymbol{b} = \boldsymbol{c}$, otherwise $\mathsf{eq}(\boldsymbol{b}, \boldsymbol{c}) = 0$. We abuse notation and similarly let $\widetilde{\boldsymbol{u}}$ denote the multilinear extension of a vector $\boldsymbol{u} \in \mathbb{F}^N$, where we naturally interpret $\boldsymbol{u}$ as a function $\boldsymbol{u}\colon \{0,1\}^n \to \mathbb{F}$ via the mapping $i \mapsto \boldsymbol{u}_{i+1}$ for $i \in \{0,1\}^n$.

We also utilize functions $g\colon \{0,1\}^n \to \mathbb{G}$ (cf. [BCS21]) and analogously define multilinear extensions of g as above. This extension is denoted as $\widetilde{g}\colon \mathbb{F}^n \to \mathbb{G}$, where $\mathsf{eq}\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ is defined as in Eq. (2) and $\mathbb{F}$ is the scalar field of $\mathbb{G}$. Now, given two polynomials $F(\boldsymbol{X}) \in \mathbb{F}[\boldsymbol{X}]$ and $G(\boldsymbol{X}) \in \mathbb{G}[\boldsymbol{X}]$, $H(\boldsymbol{X}) = F(\boldsymbol{X}) \cdot G(\boldsymbol{X}) \in \mathbb{G}[\boldsymbol{X}]$ is also a polynomial. We similarly naturally interpret $\boldsymbol{g} \in \mathbb{G}^N$ as a function $\boldsymbol{g}\colon \{0,1\}^n \to \mathbb{G}$ and let $\widetilde{\boldsymbol{g}}\colon \mathbb{F}^n \to \mathbb{G}$ denote the multilinear extension of $\boldsymbol{g}$.

Bilinear Pairings and Pairing Polynomials. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be pairing-friendly additive groups of prime order p, with scalar field $\mathbb{F}$, generators g_1, g_2, g_T, and bilinear pairing function $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. For $\boldsymbol{a} \in \mathbb{G}_1^n, \boldsymbol{b} \in \mathbb{G}_2^n$, we let $\langle \boldsymbol{a}, \boldsymbol{b} \rangle_e := e(\boldsymbol{a}, \boldsymbol{b}) = \sum_i e(a_i, b_i) \in \mathbb{G}_T$. When clear from context, we drop the subscript e above. Similarly, for $a \in \mathbb{G}_1, b \in \mathbb{G}_2$, we let $a \cdot b := e(a, b)$.

We also define the notion of polynomials over pairing friendly groups. A degree-d, n-variate polynomial $f_1(\boldsymbol{X}) \in \mathbb{G}_1[\boldsymbol{X}]$ is a function $f_1\colon \mathbb{F}^n \to \mathbb{G}_1^n$ with degree at most d in each variable. Analogously, $f_2(\boldsymbol{X}) \in \mathbb{G}_2[\boldsymbol{X}]$ is a function $f_2\colon \mathbb{F}^n \to \mathbb{G}_2$ where each variable has degree at most d. Notably, $h(\boldsymbol{X}) = f_1(\boldsymbol{X}) \cdot f_2(\boldsymbol{X}) \in \mathbb{G}_T[\boldsymbol{X}]$ is a map $h\colon \mathbb{F}^n \to \mathbb{G}_T$. Intuitively, the coefficients of h are obtained by "pairing" the coefficients of f_1 and f_2 (i.e., replacing $\cdot$ with e as above). Finally, we define the multilinear extension of functions $h_1\colon \{0,1\}^n \to \mathbb{G}_1$, $h_2\colon \{0,1\}^n \to \mathbb{G}_2$, and $h_3\colon \{0,1\}^n \to \mathbb{G}_T$ analogously as with the prior section, and denote these functions by $\widetilde{h}_1, \widetilde{h}_2, \widetilde{h}_3$, respectively. As above, we again naturally interpret $\boldsymbol{h}_1 \in \mathbb{G}_1^N$ and $\boldsymbol{h}_2 \in \mathbb{G}_2^N$ as functions $\boldsymbol{h}_1\colon \{0,1\}^n \to \mathbb{G}_1$, $\boldsymbol{h}_2\colon \{0,1\}^n \to \mathbb{G}_2$, and let $\widetilde{\boldsymbol{h}}_1\colon \mathbb{F}^n \to \mathbb{G}_1$, $\widetilde{\boldsymbol{h}}_2\colon \mathbb{F}^n \to \mathbb{G}_2$ denote the multilinear extensions of these vectors.

Interactive Protocols. An interactive protocol $\Pi = (P, V)$ consists of two players (i.e., interactive algorithms), a *prover* P and a *verifier* V. A protocol proceeds in *rounds*, where P and V each send a single message to each other in some order. In this work, we only consider protocols where during each round, the prover speaks first, followed by the verifier, and in the final round, only the prover sends a message. We say such a protocol is k-*rounds* (or has $(2k-1)$-*moves*) if P and V interact as described above for k-rounds. In particular, the prover sends k messages and the verifier sends $(k-1)$ messages.

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation (e.g., an **NP**-relation). We say that an interactive protocol Π is an *interactive proof* for R if for all $(x, w) \in R$, we have $\langle P(x, w), V(x) \rangle = 1$ with probability 1 (*completeness*), and if for any $x \notin L_R$ and any computationally unbounded prover P^*, the probability $\langle P^*(x), V(x) \rangle = 1$ is negligible (in the length of x) (*soundness*). Here, $\langle P(x, w), V(x) \rangle$ denotes the interaction between P and V according to protocol Π, where P is given (x, w) as input and V is given x as input, and L_R denotes the language of the relation R (i.e., $L_R = \{x\colon \exists w \text{ s.t. } (x, w) \in R\}$). We say that Π is an *interactive argument* if soundness holds with respect to any probabilistic polynomial time algorithm P^*. Finally, we say that Π is a *non-interactive argument* if $k = 1$. For completeness, we give the formal definition below.

Definition 2.1 (Interactive Proofs/Arguments)**.** *An* interactive proof *(resp.,* argument*) for a language $\mathcal{L}$ with relation $\mathcal{R}(\mathcal{L})$ is a tuple of algorithms (Gen, P, V) such that the following properties are satisfied:*

- Gen *is a non-interactive algorithm that on input 1^λ for security parameter $\lambda \in \mathbb{N}$ outputs public parameters* pp.
- ***Completeness****: For any $(x, w) \in \mathcal{R}(\mathcal{L})$ and $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, we have $\Pr[\langle P(w), V\rangle(\mathsf{pp}, x) = 1] = 1$, where the probability is taken over the generation of* pp *and the random coins of P and V, $\langle P, V\rangle(\mathsf{pp}, x)$ denotes the verifier output after P and V interact with common input (pp, x), and P additionally receives w as input.*
- ***ϵ-Soundness****: For any $x \notin \mathcal{L}$, $\mathsf{pp} \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$, and any unbounded (resp., polynomial-time) interactive algorithm P^*, we have $\Pr[\langle P^*, V\rangle(\mathsf{pp}, x) = 1] \leqslant \epsilon(|x|, \lambda)$.*
- ***Efficiency****: The verifier runs in time $\mathrm{poly}(|x|, \lambda)$. If (P, V) is an argument, then P runs in time $\mathrm{poly}(|x|, |w|, \lambda)$.*

The protocol (Gen, P, V) is said to be public coin *if each message sent by V are independent uniform random strings of some bounded length and the output of V does not depend on any secret state.*

Polynomial Commitment Schemes. A Polynomial Commitment Scheme (PCS) [KZG10] is a tuple of three algorithms $\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}$ and a PPT protocol Eval between two parties: a *prover* (or *committer*) P and a *verifier* V. The algorithm Setup, on input 1^λ for security parameter $\lambda \in \mathbb{N}$, outputs some set of public parameter pp, which all other algorithms take as input. In the Commit algorithm, P generates a commitment C to a polynomial $p(\cdot)$ (usually univariate or multilinear). In the Open algorithm, a party verifies that a given commitment C corresponds to a polynomial $p(\cdot)$, using the opening or randomness r during the commitment phase. In the Eval protocol, P and V engage in an interactive proof where P attempts to convince V that a claimed evaluation y is correct for a given input x, i.e., that $y = p(x)$. The verifier V outputs 1 (accepts) if the proof is successful, and 0 otherwise. Such an interactive proof often has the desirable *knowledge soundness* property. While we do not build polynomial commitment schemes in this work, for completeness we provide the formal definition of a polynomial commitment scheme below.

Definition 2.2 (Polynomial Commitment)**.** *A polynomial commitment scheme allows one party to commit to a polynomial. Subsequently, it enables them to convince another party that the polynomial, when evaluated at a particular point, results in a specific value. It consists of the following algorithms:*

- $\mathsf{pp} \leftarrow \mathsf{PC.setup}(1^\lambda, \mathcal{F})$*: Given the security parameter and a family of polynomials, output the public parameters* pp.
- $\mathsf{com} \leftarrow \mathsf{PC.commit}(\phi, \mathsf{pp})$*: On input a polynomial ϕ and the public parameter pp, output the commitment* C.
- $(y, \pi) \leftarrow \mathsf{PC.open}(\phi, \mathsf{pp}, x)$*: On input the polynomial, the public parameters, and the point to be evaluated at, output $y = \phi(x)$, and the proof.*
- $\{0, 1\} \leftarrow \mathsf{PC.verify}(\mathsf{pp}, \mathsf{com}, x, y, \pi)$*: Verify the proof against the claim $\phi(x) = y$, output 1 if verification passes, otherwise 0.*

A polynomial commitment scheme satisfies the following properties:

- *Completeness. For any polynomial $\phi \in \mathcal{F}$, and any evaluation point x. The following probability holds:*

$$\Pr\left[\mathsf{PC.verify}(\mathsf{pp}, \mathsf{com}, x, y\pi) = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{PC.setup}(1^\lambda, \mathcal{F}), \\ \mathsf{com} \leftarrow \mathsf{PC.commit}(\phi, \mathsf{pp}), \\ (y, \pi) \leftarrow \mathsf{PC.open}(\phi, \mathsf{pp}, x), \end{array}\right] = 1$$

- *Knowledge soundness. For any PPT adversary* $\mathbf{A}$*, there exists an expected PPT extractor* $\mathbf{E}$ *that on given oracle access to* $\mathbf{A}$*, outputs the polynomial such that*

$$Pr\left[\begin{array}{c|c} \phi' \leftarrow \mathbf{E}^{\mathbf{A}}(\mathsf{pp}, \mathsf{com}, x, y, \pi), & \mathsf{pp} \leftarrow \mathsf{PC.setup}(1^\lambda, \mathcal{F}), \\ \mathsf{PC.commit}(\phi') = \mathsf{com}, & (\mathsf{com}, y, x, \pi) \leftarrow \mathbf{A}(\mathsf{pp}), \\ \phi'(x) \neq y & \mathsf{PC.verify}(\mathsf{pp}, \mathsf{com}, x, y, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3 Communication Complexity

3.1 Online Merlin-Arthur Communication Protocols

Online Merlin-Arthur (OMA) protocols [BFS86, CCM+14] are a class of communication protocols which extend classical Merlin-Arthur protocols to simulate k-round streaming interactive proof (SIP) protocols. OMA protocols consist of a fixed, publicly known function f, two parties, Alice (A) and Bob (B) along with a helper party Merlin (M). Alice and Bob each have their own private inputs x_A, x_B, respectively, and may toss private random coins. Moreover, Alice and Bob may share random coins which are hidden from Merlin. Bob and Merlin may exchange k messages to help Bob compute $f(x_A, x_B)$, after which Alice may send a single message m_A to Bob.

Definition 3.1 (Online Merlin-Arthur Protocols)**.** *Let* $f\colon X \times Y \to Z$ *be an arbitrary function. We say that a protocol* Π *for computing* f *is an* $\mathbf{OMA}[k]$ *protocol if* Π *consists of two parties* A *and* B *and a helper* M *such that for all* $x \in X$ *and* $y \in Y$*, the following hold:*

1. *A receives x as input, B receives y as input, and M receives (x, y) as input;*
2. *M and B participate in a $(2k-1)$-move protocol, where B's messages do not depend on the input y and all parties send at least 1 bit per round, resulting in transcript τ;*
3. *A sends message m_A to B after this protocol; and*
4. *B outputs some $z \leftarrow B(\tau, m_A, y)$.*

Moreover, all parties may be randomized and sample both private and shared randomness prior to Step (1) above, where shared randomness *denotes random strings that are agreed upon between two or more parties.*

See Figure 4 for a pictorial overview of the $\mathbf{OMA}[k]$-model. Next, we define the security and cost of $\mathbf{OMA}[k]$-model protocols.

Definition 3.2 (Secure $\mathbf{OMA}[k]$ Protocol)**.** *We say that an* $\mathbf{OMA}[k]$ *protocol* Π *for a function* f *is* (γ, ϵ)-secure *if for all* $x \in X$ *and* $y \in Y$*, the following hold.*

1. *(Completeness) For all honest helpers M, it holds that* $\Pr_{z \leftarrow \Pi(x,y)}[z = f(x,y)] \geqslant 1 - \gamma$.
2. *(Soundness) For any dishonest helper M^*, it holds that* $\Pr_{z \leftarrow \Pi(x,y;M^*)}[z \neq f(x,y)] \leqslant \epsilon$.

Here, the probability is taken over any private or shared randomness sampled by parties. For brevity, we say that a $(1/3, 1/3)$*-secure* $\mathbf{OMA}[k]$ *protocol is simply a* secure $\mathbf{OMA}[k]$ protocol.

Definition 3.3 ($\mathbf{OMA}[k]$ Protocol Cost)**.** *Let* Π *be an* $\mathbf{OMA}[k]$ *protocol for a function* f*. We define the* verification cost *as* $\mathsf{vcost}(\Pi) := \max\{1, \log|m_A|\}$ *and the* help cost *as* $\mathsf{hcost}(\Pi) := \max\{1, \log|\tau|\}$*. Then, we define the* cost of Π *as* $C(\Pi) := \mathsf{vcost}(\Pi) + \mathsf{hcost}(\Pi)$*. Finally, we let* $C_{\mathbf{OMA}[k]}(f) := \min_{\Pi \in \mathbf{OMA}[k](f)}\{C(\Pi)\}$ *denote the* $\mathbf{OMA}[k]$-cost of f*, where* $\mathbf{OMA}[k](f)$ *denotes the set of all secure* $\mathbf{OMA}[k]$ *protocols for* f*.*

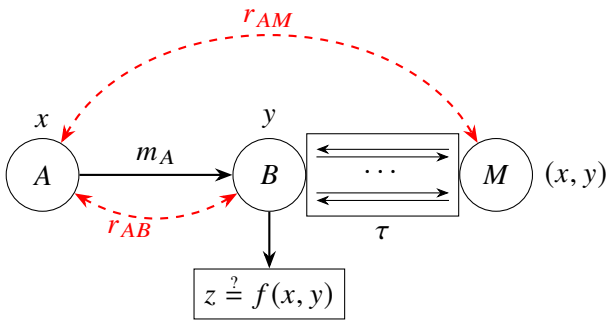


Figure 4: Graphical representation of a **OMA**$[k]$-model protocol for function $f\colon X \times Y \to Z$. Parties may or may not be randomized. If randomized, parties may sample private coins, or shared random coins. Red dashed lines indicate shared randomness between parties. τ represents the transcript of the $(2k-1)$-move protocol executed between B and M. Parties B and M never share randomness prior to the protocol execution.

3.1.1 OMA Lower Bounds

Here, we state two important lower bounds for the OMA model that we use in our work (see Section 4).

Theorem 3.4 ([CMT13])**.** *Given a (dense) matrix $\mathbf{A} \in \mathbb{F}^{a \times b}$ and a vector $\boldsymbol{x} \in \mathbb{F}^b$ for any finite field $\mathbb{F}$, any secure* **OMA**$[1]$ *protocol Π for $f(\mathbf{A}, \boldsymbol{x}) = \mathbf{A} \cdot \boldsymbol{x}^\top$ requires $\mathsf{hcost}(\Pi) \cdot \mathsf{vcost}(\Pi) = \Omega(\min(a, b)^2)$.*

Theorem 3.5 ([Kla03, CCM+19])**.** *Let $k \in \mathbb{N}$ be a constant. Given two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^m$ for any finite field $\mathbb{F}$, any secure* **OMA**$[k]$ *protocol Π for $f(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{u}, \boldsymbol{v} \rangle$ requires $\mathsf{hcost}(\Pi)^k \cdot \mathsf{vcost}(\Pi) = \Omega(m)$.*

3.2 Simultaneous Messaging with a Helper Communication Protocols

In this work, we introduce a new communication model called *Simultaneous Messaging with a Helper*, which we believe to be of independent interest. This model is a natural extension of the *Simultaneous Message* (SM) model [Yao79, BKL95, BK97]. In the SM model, there is a fixed and publicly known function f. Two parties Alice (A) and Bob (B) each have a private input x_A, x_B, respectively, and send messages m_A, m_B to a third party, the referee Charlie (C). The goal is for Charlie to correctly output $f(x_A, x_B)$ given m_A, m_B, while minimizing the quantity $|m_A| + |m_B|$. Note that in this model, all parties are computationally unbounded, may or may not be randomized, and may or may not be allowed to share randomness with other parties.

We naturally extend the SM model to the *Simultaneous Messaging with a Helper* (SMH) model. Inspired by the OMA model, we add a fourth party to the SM model, a helper Merlin (M), who receives both Alice and Bob's inputs and is allowed to communicate with Charlie for k rounds (or participate in a $(2k-1)$-move protocol) to help Charlie compute $f(x_A, x_B)$ correctly.

Definition 3.6 (Simultaneous Messaging with a Helper)**.** *Let $f\colon X \times Y \to Z$ be an arbitrary function. We say that a protocol Π for computing f is a* **SMH**$[k]$ protocol *if Π consists of two parties A and B, a referee C, and a helper M, such that for all $x \in X$ and $y \in Y$, the following hold:*

1. *A receives x as input, B receives y as input, and M receives (x, y) as input;*
2. *M and C participate in a $(2k-1)$-move protocol, where all parties send at least 1 bit per round, resulting in transcript τ;*
3. *After this interaction, A and B send messages m_A, m_B, respectively, to C; and*

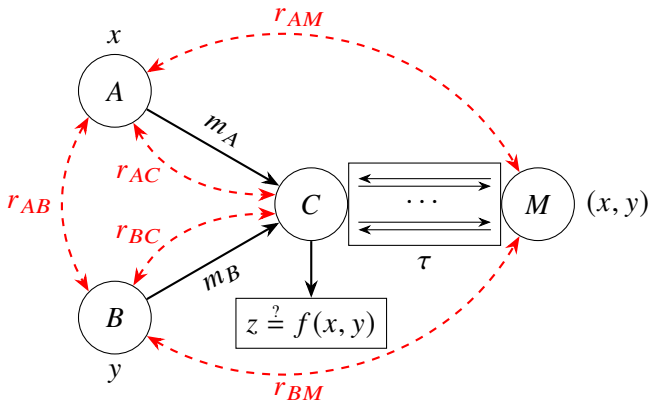


Figure 5: Graphical representation of a $\mathbf{SMH}[k]$-model protocol for function $f\colon X \times Y \to Z$. Parties may or may not be randomized. If randomized, parties may sample private coins, or shared random coins. Red dashed lines indicate shared randomness between parties. τ represents the transcript of the $(2k-1)$-move protocol executed between C and M. Parties C and M never share randomness prior to the protocol execution.

4. *C outputs $z \leftarrow C(\tau, m_A, m_B)$.*

Moreover, all parties may be randomized and may sample both private and shared randomness prior to Step (1) above, and C's output depends on any privately sampled randomness or randomness shared between C and other parties. Here, by shared randomness*, we mean that two (or more) parties agree on common random strings prior to Step (1) above.*

See Figure 5 for a pictorial overview of the $\mathbf{SMH}[k]$-model. Next we define the security and cost of $\mathbf{SMH}[k]$-model protocols.

Definition 3.7 (Secure **SMH** [k] Protocol)**.** *We say that a $\mathbf{SMH}[k]$ protocol Π for a function f is* (γ, ϵ)-secure *if for all $x \in X$ and $y \in Y$, the following hold.*

1. *(Completeness) There exists an honest M such that $\Pr_{z \leftarrow \Pi(x,y)}[z = f(x,y)] \geqslant 1-\gamma$.*
2. *(Soundness) For any dishonest helper M^*, it holds that $\Pr_{z \leftarrow \Pi(x,y;M^*)}[z \neq f(x,y)] \leqslant \epsilon$.*

Here, the probability is taken over any private or shared randomness sampled by the parties. For brevity, we say that a $(1/3, 1/3)$-secure $\mathbf{SMH}[k]$ protocol is simply a secure $\mathbf{SMH}[k]$ protocol*.*

Definition 3.8 (**SMH**$[k]$ Protocol Cost)**.** *Let Π be a $\mathbf{SMH}[k]$ protocol for a function f. We define* $\mathsf{acost}(\Pi) := \max\{1, \log|m_A|\}$, $\mathsf{bcost}(\Pi) := \max\{1, \log|m_B|\}$, *and* $\mathsf{hcost}(\Pi) = \max\{1, \log|\tau|\}$ *respectively as* Alice's cost, Bob's cost, *and the* help cost *of Π. Then, we let $C(\Pi) := \mathsf{acost}(\Pi) + \mathsf{bcost}(\Pi) + \mathsf{hcost}(\Pi)$ denote the* cost of Π*. Moreover, we let $C_{\mathbf{SMH}[k]}(f) := \min_{\Pi \in \mathbf{SMH}[k](f)}\{C(\Pi)\}$ the $\mathbf{SMH}[k]$-*cost of f, where $\mathbf{SMH}[k](f)$ denotes the set of all secure $\mathbf{SMH}[k]$ protocols for f.*

Note that any $\mathbf{SMH}[k]$ protocol can be simulated by an $\mathbf{OMA}[k]$ protocol.

Lemma 3.9. *Let Π be a (γ, ϵ)-secure $\mathbf{SMH}[k]$ protocol for a function f. Then there exists a (γ, ϵ)-secure $\mathbf{OMA}[k]$ protocol Π' for computing f.*

Proof. Let Π be such an $\mathbf{SMH}[k]$ protocol. Recall that by definition, this means for function $f\colon X \times Y \to Z$, the protocol Π has parties A, B, C, and M. Party A gets input $x \in X$, party B gets input $y \in Y$, and party M gets (x, y) as input. Parties M and C interact over k-rounds of communication, parties A and B may or may not share randomness with each other, with C, or with M.

We show how to simulate Π via an $\mathbf{OMA}[k]$ protocol. Let Π' be a protocol in the **OMA** model with parties A', B', and M'. These three parties will simulate the protocol Π as follows.

- Party B' will play the role of a "super party." In particular, B' will simulate both parties B and C in its head. Any messages and/or randomness B sends to C in protocol Π is simulated in the head of B'.
- Party M' will interact with party B' exactly as party M interacts with party C in Π. Moreover, if B or C shares randomness with M in Π, B' shares this same randomness with M'.
- Party A' will exactly simulate party A by treating M' as M and by treating B' as both B and C.

Clearly, if Π had completeness error γ and soundness error ϵ, then Π' also has the same completeness and soundness errors. □

3.3 SMH Model Lower Bounds

We introduce the $\mathbf{SMH}[k]$ model in order to appropriately model cryptographic protocols in an information-theoretic setting where we can lower bound verification costs via communication complexity. Thus, we need to establish lower bounds on the communication complexity of certain functions in the $\mathbf{SMH}[k]$ model. Key to our results is computing the inner product between two vectors in the $\mathbf{SMH}[k]$ model, subject to certain restrictions on parties in the protocol (we will highlight why these restrictions are needed in Section 5). We prove the following theorem.

Theorem 3.10. *Let Π be any $\mathbf{SMH}[k]$ protocol for $f(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$ for any finite field $\mathbb{F}$, subject to the following constraints:*

- *Alice is deterministic;*
- *Bob, Charlie, and Merlin may sample private random coins; and*
- *Bob and Charlie can share randomness (hidden from Merlin).*

Then, $C(\Pi) = \Omega(n)$. In particular, Alice must send $\Omega(n)$ bits to Charlie.

Proof. We show the result for $\mathbb{F} := \{0, 1\}$. Since one can always embed the binary field into large finite fields, the lower bound applies for any finite field. Let $\boldsymbol{x}$ denote Alice's input and $\boldsymbol{y}$ denote Bob's input and suppose that Alice is deterministic. Then for any $\boldsymbol{x} \neq \boldsymbol{x}' \in \{0, 1\}^n$, there exists $\boldsymbol{y} \in \{0, 1\}^n$ such that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \neq \langle \boldsymbol{x}', \boldsymbol{y} \rangle$, which can be constructed as follows. Suppose $\boldsymbol{x}$ and $\boldsymbol{x}'$ disagree at bit i; i.e., $\boldsymbol{x}_i \neq \boldsymbol{x}'_i$. Without loss of generality, assume that $\boldsymbol{x}_i = 1$ and $\boldsymbol{x}'_i = 0$. Then define $\boldsymbol{y}$ as $\boldsymbol{y}_i = 1$ and $\boldsymbol{y}_j = 0$ for $j \neq i$. Clearly, $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 1$, while $\langle \boldsymbol{x}', \boldsymbol{y} \rangle = 0$.

Now suppose that Alice sends at most $(n - 1)$ bits to Charlie in protocol Π. Since there are 2^n possible strings $\boldsymbol{x}$ that Alice can receive as input and there are only 2^{n-1} possible $(n - 1)$-bit messages m_A that Alice can send to Charlie, by the Pigeonhole Principle there must exist a pair of strings $\boldsymbol{x} \neq \boldsymbol{x}' \in \{0, 1\}^n$ such that $m_A(\boldsymbol{x}) = m_A(\boldsymbol{x}')$. That is, Alice sends the same message m_A on inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$. Without loss of generality, suppose that $x_i = 1$ and $x'_i = 0$ for some index $i \in [n]$. Now suppose Bob has input $\boldsymbol{y} \in \{0, 1\}^n$ such that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 1$ and $\langle \boldsymbol{x}', \boldsymbol{y} \rangle = 0$. By our previous argument above, such an input $\boldsymbol{y}$ must exist. Since Π is an $\mathbf{SMH}[k]$ protocol, we know that there exists a Charlie-Merlin transcript τ such that

$$\Pr_{\Pi}[C(\tau, m_A(\boldsymbol{x}), m_B(\boldsymbol{y})) = 1] \geqslant 2/3,$$

where the probability is taken over all random coins sampled by parties in the protocol. However, we know that $m_A(\boldsymbol{x}) = m_A(\boldsymbol{x}')$, so it must also hold that

$$\Pr_{\Pi}[C(\tau, m_A(\boldsymbol{x}'), m_B(\boldsymbol{y})) = 1] \geqslant 2/3.$$

This violates the soundness of Π since $\langle \boldsymbol{x}', \boldsymbol{y} \rangle = 0$, but Charlie outputs 1 with probability at least $2/3$. Hence, Alice must send at least n-bits of communication to Charlie, which implies $C(\Pi) = \Omega(n)$. □

As a direct corollary, the above lower bound also holds for $\mathbf{OMA}[k]$ protocols under the same restrictions.

Corollary 3.11. *Let Π be a secure $\mathbf{OMA}[k]$ protocol for $f(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, where $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$, such that party Alice is deterministic and parties Bob and Merlin can use private randomness. Then, $C(\Pi) = \Omega(n)$.*

Proof. The proof is identical to that of Theorem 3.10 with deterministic Alice and randomized Bob simulating parties B and C in the SMH protocol. □

4 Hyrax: Information-Theoretic Core and Evidence of Optimal Verifier Time

In this section, we provide strong evidence that the Hyrax polynomial commitment scheme, which we denote by Hyrax-PCS, has optimal verifier time. We do this by first extracting the generalized scheme for proving knowledge of vector-matrix-vector product, which we call *Hyrax-VMV*. Then, we extract the information-theoretic core of Hyrax-VMV as a communication protocol; specifically, an **OMA** protocol. We then show that any **OMA** protocol computing the same function as Hyrax-VMV has certain lower bounds on the number of bits exchanged between parties, giving evidence that the Hyrax-VMV verifier must perform at least the same number of bit-operations. Finally, we instantiate Hyrax-VMV to obtain Hyrax-PCS and show that the communication lower bounds of the **OMA** protocol nearly match the verification costs of Hyrax-PCS.

To begin, we present the formal Hyrax-VMV protocol in Algorithm 4.1 and give an overview of the protocol here. Hyrax-VMV allows a prover/committer to commit to a matrix $\mathbf{A} \in \mathbb{F}^{a \times b}$ and later prove that $y = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$ for vectors $\boldsymbol{L} \in \mathbb{F}^a$ and $\boldsymbol{R} \in \mathbb{F}^b$ supplied by the verifier/receiver. For a group of prime order $\mathbb{G}$ with scalar field $\mathbb{F}$, the public parameters of the scheme are randomly sampled group generators $\boldsymbol{g} := (g_1, \ldots, g_b) \in \mathbb{G}^b$. To commit to the matrix $\mathbf{A}$, the prover computes vector $\boldsymbol{C} = \mathbf{A} \cdot \boldsymbol{g}^\top \in \mathbb{G}^a$; or, equivalently, $\boldsymbol{C}_i = \langle \boldsymbol{A}_{i,*}, \boldsymbol{g} \rangle$ for $i \in [b]$, where $\boldsymbol{A}_{i,*}$ denotes the i^{th}row of $\mathbf{A}$.

After receiving a commitment $\boldsymbol{C}$, the verifier sends vectors $\boldsymbol{L}, \boldsymbol{R}$.[5] Given these vectors, the proof provided by the prover is a pair $(\boldsymbol{L}', y) \in \mathbb{F}^b \times \mathbb{F}$, the purported values of $\boldsymbol{L}\mathbf{A}$ and $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$, respectively. Verification proceeds by using $\boldsymbol{C}$ to check consistency between $\boldsymbol{L}$ and $\boldsymbol{L}'$, and by using $\boldsymbol{R}$ to check consistency between y and $\boldsymbol{L}'$. The verifier samples $\alpha \xleftarrow{\$} \mathbb{F}$ and checks: (1) $\alpha\langle \boldsymbol{L}, \boldsymbol{C} \rangle \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{g} \rangle$; and (2) $\alpha y \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{R} \rangle$.

Our goal is to extract the information-theoretic core of Hyrax-VMV as an **OMA** protocol and prove lower bounds on the communication needed in this protocol to compute the same function as Algorithm 4.1. We work towards this goal and proving the following theorem (the proof of which can be found in Section 4.1.1).

Theorem 4.1. *Let $a, b \in \mathbb{N}$, let $\boldsymbol{L} \in \mathbb{F}^a$, $\boldsymbol{R} \in \mathbb{F}^b$, and $\mathbf{A} \in \mathbb{F}^{a \times b}$ be inputs to Hyrax-VMV. Then, there exists an $\mathbf{OMA}[1]$ protocol Π which verifies the same function as Hyrax-VMV with inputs $\boldsymbol{L}, \boldsymbol{R}, \mathbf{A}$ such that $C(\Pi) = \Omega(\max(\sqrt{b}, \min(a, b)))$.*

As a direct corollary, we obtain strong evidence that Hyrax-PCS has (nearly) optimal verifier time.

[5] Note that the scheme in [WTS+18] is *not sound* if the vectors are given to the prover *before* he commits to his input.

Algorithm 4.1: Hyrax-VMV

Public Parameters: $a, b \in \mathbb{N}$, $\mathbb{G}$, $\boldsymbol{g} \in \mathbb{G}^b$.

Private Input: $\mathbf{A} \in \mathbb{F}^{a \times b}$.

Commit Phase: ($\mathbf{A} \in \mathbb{F}^{a \times b}$, $\boldsymbol{g} \in \mathbb{G}^b$)

  Output: $\boldsymbol{C} = \mathbf{A} \cdot \boldsymbol{g}^\top \in \mathbb{G}^a$.

Proof of Knowledge Phase: ($\boldsymbol{C} \in \mathbb{G}^a$, $\boldsymbol{L} \in \mathbb{F}^a$, $\boldsymbol{R} \in \mathbb{F}^b$, $\boldsymbol{g} \in \mathbb{G}^b$; $\mathbf{A} \in \mathbb{F}^{a \times b}$).

  Output: Accept or reject.

  Prover sends $y = \boldsymbol{L} \cdot \mathbf{A} \cdot \boldsymbol{R}^\top \in \mathbb{F}$ and $\boldsymbol{L}' = \boldsymbol{L} \cdot \mathbf{A} \in \mathbb{F}^b$ to verifier.

  Verifier samples $\alpha \xleftarrow{\$} \mathbb{F}$ and checks if $\alpha\langle \boldsymbol{L}, \boldsymbol{C}\rangle \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{g}\rangle$ and $\alpha y \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{R}\rangle$.

  Verifier outputs accept if and only if both checks pass.

Corollary 4.2. *Let $N = 2^n$ and $\mathbb{G}$ be a group with scalar field $\mathbb{F}$.*

- *For inputs $f : \{0,1\}^n \to \mathbb{F}$, $\boldsymbol{u} \in \mathbb{F}^n$, and $\boldsymbol{g} \in \mathbb{G}^{\sqrt{N}}$ to Hyrax-PCS for multilinear polynomials, there exists an* **OMA**$[1]$ *protocol Π which verifies the same function as Hyrax-PCS with inputs $f, \boldsymbol{u}, \boldsymbol{g}$ such that $C(\Pi) = \Omega(\sqrt{N})$.*
- *For inputs $f \in \mathbb{F}^{<N}[X]$, $u \in \mathbb{F}$, and $\boldsymbol{g} \in \mathbb{G}^{\sqrt{N}}$ to Hyrax-PCS for univariate polynomials, there exists an* **OMA**$[1]$ *protocol Π which verifies the same function as Hyrax-PCS with inputs $f, u, \boldsymbol{g}$ such that $C(\Pi) = \Omega(\sqrt{N})$.*

Moreover, the prescribed Hyrax-PCS verifier (for both multilinear and univariate polynomials) performs at most $O(\sqrt{N}\,\text{polylog}\,|\mathbb{G}|)$ bit-operations, assuming a single $\mathbb{G}/\mathbb{F}$ operation requires $\text{polylog}\,|\mathbb{G}|$ *bit-operations.*

Corollary 4.2 follows directly from Theorem 4.1 since Hyrax-PCS is an instantiation of Hyrax-VMV with $a = b = \sqrt{N}$ for some $N = 2^n$, structured vectors $\boldsymbol{L}, \boldsymbol{R}$, and structured matrix $\mathbf{A}$, and by noting that the proof size of Hyrax-VMV is $b + 1$ elements of $\mathbb{F}$ and $2a$ elements of $\mathbb{G}$. Here, we make the implicit assumption that $|\mathbb{G}| \geqslant |\mathbb{F}|$. The full proof of Corollary 4.2 is given in Section 4.2.

We build up to the proof of Theorem 4.1 by first extracting the information-theoretic core of Hyrax-VMV as an OMA protocol. Given this information-theoretic core, we apply prior known lower bounds to the problems of vector-matrix-vector products in the OMA model. Then, we show how these lower bounds directly translate to a lower bound on the verifier work of Hyrax-VMV, completing the proof. Finally, we show how to instantiate Algorithm 4.1 to obtain Hyrax-PCS, directly giving Corollary 4.2.

4.1 Information Theoretic Core of Hyrax-VMV

Hyrax-VMV is a proof of knowledge for proving vector-matrix-vector products, so the information-theoretic core of Hyrax-VMV must also be a protocol for verifying vector-matrix-vector products. Crucial here is first understanding (a) how Hyrax-VMV verifies vector-matrix-vector products, and (b) what the cryptography (in this case, the discrete-log assumption) is guaranteeing in the protocol. Understanding (1) gives us a direct mapping to a potential communication protocol for Hyrax-VMV, and understanding (2) is crucial to specifying the protocol, in particular, what guarantees the cryptography gives in Hyrax-VMV must also be guaranteed in the underlying information-theoretic protocol.

Towards (1), examining Algorithm 4.1 shows us that in order to prove vector-matrix-vector products, Hyrax-VMV proves/verifies in parallel 2 claims: a vector-matrix product claim (i.e., $\boldsymbol{L}', = \boldsymbol{L}\mathbf{A}$) and an inner product claim ($y = \langle \boldsymbol{L}', R\rangle$). This gives us a straightforward way of specifying Algorithm 4.1 as an **OMA** protocol; see Figure 1 in Section 1.2.3. At a high-level, Figure 1 captures Algorithm 4.1 as follows.

- Party M plays the role of the prover in the **Eval Phase** of Algorithm 4.1.
- Party A plays the role of the committer in the **Commit Phase** of Algorithm 4.1.
- Party B plays the role of the verifier of Algorithm 4.1.

There are two key differences between Figure 1 and Algorithm 4.1. In Figure 1:

1. Party A generates the "commitment" $\boldsymbol{C} = \mathbf{A}\boldsymbol{r}^\top$ using private shared randomness $\boldsymbol{r} \in \mathbb{F}^b$;
2. Party A sends the "commitment" $\boldsymbol{C}$ after parties M and B execute the **Eval Phase** of the protocol.

First, item (2) is here due to the **OMA** model, which requires the interaction between parties B and M to be *independent* of the message sent by A and of B's input. Since B does not send anything to M, this second restriction is already satisfied. The first restriction is enforced by having A send their message after all interaction between B and M has been executed. For item (1), the vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$ is playing the role of the vector $\boldsymbol{g} \in \mathbb{G}^b$ of Algorithm 4.1. Recall that $\boldsymbol{g}$ is a randomly sampled vector of generators for $\mathbb{G}$. Notice that such a vector can be equivalently sampled as follows: (i) sample a single generator $g \in \mathbb{G}$; sample random $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$ (ii); and (iii) set generator $\boldsymbol{g}_i = \boldsymbol{r}_i \cdot g$. Since g is a generator, $\boldsymbol{g}_i$ is also a generator for all i.

In Algorithm 4.1, if the prover *knew* the vector $\boldsymbol{r} \in \mathbb{F}^b$, then it can easily break the binding of the commitment scheme. However, in the context of Algorithm 4.1, the prover does not know $\boldsymbol{r}$ due to the hardness of the discrete-log problem in $\mathbb{G}$. Thus, this vector $\boldsymbol{r}$ is hidden from the prover in Algorithm 4.1.

In the **OMA** model, all parties are computationally unbounded. If party M knew the vector $\boldsymbol{r}$ that parties A and B share, then party M can trivially break the soundness of the protocol and convince B that $y' = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$ for some $y' \neq y$, where y is the real value of $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$. So the vector $\boldsymbol{r}$ is hidden from party M by specifying it as the private randomness shared between A and B (before the interaction between M and B). In particular, we have answered (b) from the start of this section: the cryptography is forcing the prover in Hyrax-VMV to provide a correct random fingerprint of the matrix $\mathbf{A}$ using the vector $\boldsymbol{g}$, otherwise the hardness of discrete-log is broken, or we can extract the prover's true input $\mathbf{A}$ due to the proof of knowledge property of Hyrax-VMV. Thus, in our **OMA** protocol for Hyrax-VMV, this invariant must also be enforced; this is handled by having party A (who is honest) send the random fingerprint of $\mathbf{A}$ using randomness that is hidden from party M.

4.1.1 Hyrax-VMV Lower Bound

With the information-theoretic core of Algorithm 4.1 established in Figure 1, we now give a lower bound on the communication cost of Figure 1, which we then use to obtain Theorem 4.1.

Proof of Theorem 4.1. As in Definition 3.3, we let hcost denote the number of bits exchanged between M and B, and let vcost denote the number of bits A sends to B. Our goal is to minimize the quantity $\mathsf{hcost} + \mathsf{vcost}$. We do this by analyzing $\mathsf{hcost} \cdot \mathsf{vcost}$. However, we first establish that Figure 1 is a valid (i.e., secure according to Definition 3.2) **OMA** protocol via the following claim.

Claim 4.3. *Let $a, b \in \mathbb{N}$ and let $f(\mathbf{A}, \boldsymbol{L}, \boldsymbol{R}) := \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$ for $\boldsymbol{L} \in \mathbb{F}^a, \boldsymbol{R} \in \mathbb{F}^b$, and $\mathbf{A} \in \mathbb{F}^{a \times b}$ for any finite field $\mathbb{F}$. Then, Figure 1 is a $(0, 2/|\mathbb{F}|)$-secure* **OMA**$[1]$ *protocol for computing f.*

Perfect completeness is immediate. For soundness, suppose that M wishes to convince B that $y^* = \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$ for some $y^* \neq y := \boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top$. Thus, M must send some $\boldsymbol{L}^*$ such that $(y^*, \boldsymbol{L}^*)$ pass the checks of B. If party B samples $\alpha = 0$, then all of their checks pass; this happens with exactly $1/|\mathbb{F}|$ probability. Suppose this is not the case. Then, $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^b$ is uniformly sampled and hidden from M, so the value $\langle \boldsymbol{L}^*, \alpha \rangle$ is uniformly distributed in $\mathbb{F}$. So the probability $\langle \boldsymbol{L}, \boldsymbol{C} \rangle = \langle \boldsymbol{L}^*, \boldsymbol{r} \rangle$ is exactly $1/|\mathbb{F}|$, which is required to pass B's check for $\alpha \neq 0$. If this check does pass, then B checks if $\alpha y^* \stackrel{?}{=} \alpha \langle \boldsymbol{L}^*, \boldsymbol{R} \rangle$. Note that M can guarantee this check passes

Algorithm 4.2: Hyrax-PCS

Public Parameters: $N = 2^n$, $\mathbb{G}$, $\boldsymbol{g} \in \mathbb{G}^{\sqrt{N}}$.
Private Input: $f\colon \{0,1\}^n \to \mathbb{F}$ for multilinear PCS, $f \in \mathbb{F}^{<N}[X]$ for univariate PCS.

Commit Phase: $(f \in \mathbb{F}^N, \boldsymbol{g} \in \mathbb{G}^{\sqrt{N}})$
- **Output:** $\boldsymbol{C} \in \mathbb{G}^N$
- Define matrix $\mathbf{F}$ as follows:
  - If multilinear PCS, $\mathbf{F}[i,j] = f\left((i-1) + \sqrt{N}(j-1)\right)$ (i.e., evaluations of f) for $i, j \in [\sqrt{N}]$.
  - If univariate PCS, $\mathbf{F}[i,j] = f_{(i-1)+\sqrt{N}(j-1)}$ (i.e., coefficients of f) for $i, j \in [\sqrt{N}]$.

  Output $\boldsymbol{C} = \mathbf{F} \cdot \boldsymbol{g}^\top$.

Eval Phase: ($\boldsymbol{C} \in \mathbb{G}^{\sqrt{N}}$, $\boldsymbol{u} \in \mathbb{F}^n$ *(for multilinear PCS) or* $r \in \mathbb{F}$ *(for univariate PCS)*, $\boldsymbol{g} \in \mathbb{G}^{\sqrt{N}}$; f)
- **Output:** Verifier accepts or rejects.
- If the PCS is multilinear, the prover and verifier compute $\boldsymbol{L}, \boldsymbol{R}$ as

$$\boldsymbol{L} = \left(\chi^+(\boldsymbol{u}, 0), \chi^+(\boldsymbol{u}, 1), \ldots, \chi^+(\boldsymbol{u}, \sqrt{N}-1)\right) \quad \boldsymbol{R} = \left(\chi^-(\boldsymbol{u}, 0), \chi^-(\boldsymbol{u}, 1), \ldots, \chi^-(\boldsymbol{u}, \sqrt{N}-1)\right),$$

- where $\chi(X, Y) := X_i Y_i + (1 - X_i)(1 - Y_i)$ and

$$\chi^+(\boldsymbol{u}, \boldsymbol{c}) := \prod_{k=1}^{n/2} \chi(u_k, c_k) \quad \chi^-(\boldsymbol{u}, \boldsymbol{c}) := \prod_{k=1}^{n/2} \chi(u_{k+n/2}, c_k).$$

- If the PCS is univariate, the prover and verifier compute $\boldsymbol{L}, \boldsymbol{R}$ as :

$$\boldsymbol{L} = \left(1, r^{\sqrt{N}}, r^{2\sqrt{N}}, \ldots, r^{(\sqrt{N}-1)\sqrt{N}}\right) \quad \boldsymbol{R} = \left(1, r, r^2, \ldots, r^{\sqrt{N}-1}\right).$$

- The prover additionally computes $\mathbf{F} \in \mathbb{F}^{\sqrt{N} \times \sqrt{N}}$ as in the commit phase.
- The prover computes and sends $y = \boldsymbol{L} \cdot \mathbf{F} \cdot \boldsymbol{R}^\top \in \mathbb{F}$ and $\boldsymbol{L}' = \boldsymbol{L} \cdot \mathbf{F} \in \mathbb{F}^{\sqrt{N}}$.
- The verifier samples $\alpha \xleftarrow{\$} \mathbb{F}$ and checks the following two equations:

$$\alpha\langle \boldsymbol{L}, \boldsymbol{C}\rangle \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{g}\rangle \qquad \alpha y \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{R}\rangle.$$

- The verifier accepts if and only if both checks pass.

by setting $\boldsymbol{L}^*$ such that $y^* = \boldsymbol{L}^* \boldsymbol{R}^\top$, so in this case the check passes if the above check also passes. Otherwise, this check fails for $\alpha \neq 0$. So the total soundness error is upper bounded by $2/|\mathbb{F}|$, establishing Claim 4.3.

We now turn towards minimizing hcost+vcost. Notice that Figure 1 is an **OMA** protocol for simultaneously verifying one matrix-vector product and ($\boldsymbol{L}' = \boldsymbol{L}\mathbf{A}$), and one inner product ($y = \langle \boldsymbol{L}', \boldsymbol{R}\rangle$). In our protocol, B verifies the matrix-vector product $\boldsymbol{L}\mathbf{A}$ and the inner product $\langle \boldsymbol{L}', \boldsymbol{R}\rangle$, where $\boldsymbol{L} \in \mathbb{F}^a$ and $\boldsymbol{L}', \boldsymbol{R} \in \mathbb{F}^b$. To verify the matrix-vector product, Theorem 3.4 tells us that $\mathsf{vcost} \cdot \mathsf{hcost} = \Omega(\min(a,b)^2)$, whereas to verify the inner product, Theorem 3.5 tells us that $\mathsf{vcost} \cdot \mathsf{hcost} = \Omega(b)$. Combining these bounds, we have a lower bound of $\mathsf{hcost} \cdot \mathsf{vcost} = \Omega(\max(b, \min(a,b)^2))$.

Let $\ell = \max(b, \min(a,b)^2)$. Now, we minimize $\mathsf{hcost} + \mathsf{vcost}$ subject to the constraint that $\mathsf{hcost} \cdot \mathsf{vcost} = \Omega(\ell)$. This constraint implies $\mathsf{vcost} \geqslant \ell/\mathsf{hcost}$, which implies $\mathsf{hcost} + \mathsf{vcost} \geqslant \mathsf{hcost} + \ell/\mathsf{hcost}$. Taking the derivative of the right-hand side with respect to hcost, we see that the minimum value is obtained at $\mathsf{hcost} = \sqrt{\ell}$, and thus hcost + vcost is minimized at $\Omega(\sqrt{\ell}) = \Omega(\max(\sqrt{b}, \min(a,b)))$. This implies that party B must receive at least $\Omega(\max(\sqrt{b}, \min(a,b)))$ bits in any **OMA** protocol which simultaneously verifies a matrix-vector product and an inner product. □

4.2 Evidence that Hyrax-PCS has Optimal Verifier Time

With the Hyrax-PCS presented in Algorithm 4.2, we now show Corollary 4.2, giving us strong evidence that Hyrax-PCS (for both multilinear and univariate polynomials) has optimal verifier time (up to polylog $|\mathbb{G}|$ factors). In particular, this entails showing how to build Hyrax-PCS from Hyrax-VMV, then applying our lower bound in conjunction with analyzing the work done by the verifier in Hyrax-PCS.

Proof of Corollary 4.2. Hyrax-PCS is simply an instantiation of Hyrax-VMV with more structured matrices and vectors. We begin with Hyrax-PCS for multilinear polynomials; the construction for univariate polynomials is similar and given at the end of the proof. In the multilinear PCS setting, the prover has a function $f\colon \{0,1\}^n \to \mathbb{F}$ such that n is even. Setting $N = 2^n$, the prover then defines a $\sqrt{N} \times \sqrt{N}$ matrix $\mathbf{F}$ representing f as $\mathbf{F}[i,j] := f\left((i-1) + \sqrt{N} \cdot (j-1)\right)$ for $i, j \in [\sqrt{N}]$. Here, we interpret $(i-1)+\sqrt{N}(j-1)$ as an n-bit string in the natural way. So to commit to (the multilinear extension of) f, the prover outputs $\boldsymbol{C} \leftarrow$ Hyrax-VMV.**Commit**$(\mathbf{F}, \boldsymbol{g})$.

Now for the evaluation proofs, let $\boldsymbol{u} \in \mathbb{F}^n$ be the evaluation point specified by the verifier after being provided some commitment $\boldsymbol{C}$. Let $\chi(X,Y) := X_iY_i + (1-X_i)(1-Y_i)$ (e.g., see Eq. (2)). Then, for any $\boldsymbol{c} \in \{0,1\}^{n/2}$, the verifier defines

$$\chi^+(\boldsymbol{u}, \boldsymbol{c}) := \prod_{k=1}^{n/2} \chi(u_k, c_k) \qquad\qquad \chi^-(\boldsymbol{u}, \boldsymbol{c}) := \prod_{k=1}^{n/2} \chi(u_{k+n/2}, c_k).$$

Finally, vectors $\boldsymbol{L}, \boldsymbol{R}$ are defined as $\boldsymbol{L} = (\chi^+(\boldsymbol{u}, i))_{i=0}^{\sqrt{N}-1}$ and $\boldsymbol{R} = (\chi^-(\boldsymbol{u}, i))_{i=0}^{\sqrt{N}-1}$, where we interpret i as a vector in $\{0,1\}^{n/2}$ in the natural way. The prover and verifier then run Hyrax-VMV.**Eval**$(\boldsymbol{C}, \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{g}; \mathbf{F})$ with the above definitions. From Hyrax [WTS+18], it can be show that for $\boldsymbol{L}, \boldsymbol{R}$, and $\mathbf{A}$ defined this way, we have $\boldsymbol{L}\mathbf{A}\boldsymbol{R}^\top = \widetilde{f}(\boldsymbol{u})$, where $\widetilde{f}$ is the multilinear extension of f (defined in Eq. (1)).

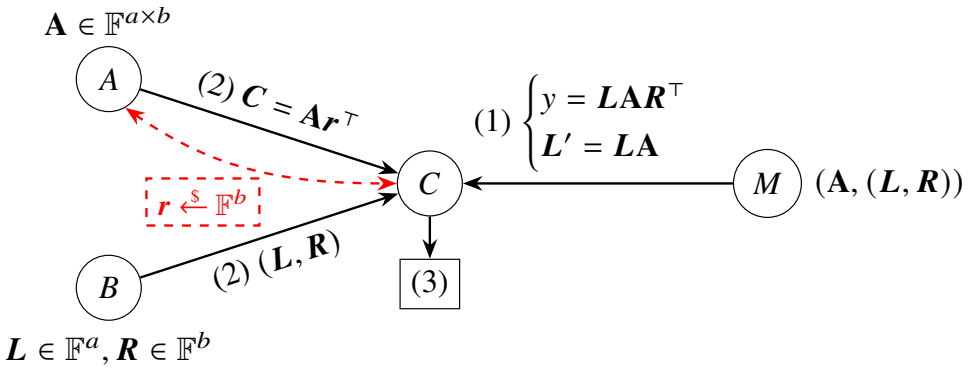
Now, translating Hyrax-PCS to the **OMA** model is nearly identical to Hyrax-VMV. The only changes are: (1) we specify $a = b = \sqrt{N}$ for $N = 2^n$ and even n; and (2) how the vectors $\boldsymbol{L}, \boldsymbol{R}$ are sampled by the verifier. These two changes are reflected by the following minor modifications to Figure 1.

1. Merlin's input is $(\mathbf{F}, \boldsymbol{u})$, Bob's input is $\boldsymbol{u} \in \mathbb{F}^n$, and Alice's input is $\mathbf{F} \in \mathbb{F}^{\sqrt{N} \times \sqrt{N}}$. Notice that $f\colon \{0,1\}^n \to \mathbb{F}$ uniquely defines $\mathbf{F}$ and vice versa.
2. Merlin, during step (1) of the protocol, will appropriately define $\boldsymbol{L}$ and $\boldsymbol{R}$ as in Hyrax-PCS (described above).
3. Bob, during step (3) of the protocol, will also appropriately define $\boldsymbol{L}, \boldsymbol{R}$ as in Hyrax-PCS.

In particular, the messages sent by both Alice and Merlin to Bob remain the same; they just now all have length $\Theta(\sqrt{N})$. Moreover, this is clearly just an instance of Figure 1 with $a = b = \sqrt{N}$, and has the same soundness error as Figure 1 (Claim 4.3) since each entry of $\boldsymbol{u}$ is uniformly random.

By Theorem 4.1, this implies a lower bound of $\Omega(\sqrt{N})$ for the **OMA** protocol representing Hyrax-PCS. Notice that in Hyrax-PCS, the verifier computes 3 inner products of size $\sqrt{N}$; two are inner products between a $\mathbb{G}$ and a $\mathbb{F}$ vector, and one is an inner product between two $\mathbb{F}$ vectors. This implies at most $O(\sqrt{N}$ polylog $|\mathbb{G}|)$ bit-operations for these computations, assuming $\mathbb{G}/\mathbb{F}$ operations can be done in polylog $|\mathbb{G}|$ bit-operations.

For univariate polynomials, the above argument is identical; we simply need to specify how the prover constructs the matrix $\mathbf{A}$ from a univariate polynomial and how the verifier constructs $\boldsymbol{L}, \boldsymbol{R}$ from an evaluation point. Let $p \in \mathbb{F}[X]$ be a univariate polynomial such that $\deg(p) < N$; write $p(X) = p_0 + p_1X + \cdots + p_{N-1}X^{N-1}$. Then, the prover defines a $\sqrt{N} \times \sqrt{N}$ matrix $\mathbf{P}$ representing p as $\mathbf{P}[i,j] = p_{(i-1)+\sqrt{N}(j-1)}$ for $i, j \in [\sqrt{N}]$. So to commit to p, the prover outputs $\boldsymbol{C} \leftarrow$ Hyrax-VMV.**Commit**$(\mathbf{P}, \boldsymbol{g})$.



(3) C does the following.
- Sample $\alpha \xleftarrow{\$} \mathbb{F}$.
- Check $\alpha\langle \boldsymbol{L}, \boldsymbol{C}\rangle \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{r}\rangle$ and $\alpha y \stackrel{?}{=} \alpha\langle \boldsymbol{L}', \boldsymbol{R}\rangle$.
- Output accept (i.e., output y) if and only if all checks pass.

Figure 6: Hyrax-VMV as a **SMH**[1] protocol. The red dashed line indicates private randomness shared between the respective parties. The protocol executes (1), followed by (2), followed by (3).

For the evaluation proofs, let $r \in \mathbb{F}$ be the evaluation point specified by the verifier after receiving some commitment $\boldsymbol{C}$. The verifier defines $\boldsymbol{R} = (1, r, r^2, \ldots, r^{\sqrt{N}-1})$ and $\boldsymbol{L} = (1, r^{\sqrt{N}}, r^{2\sqrt{N}}, \ldots, r^{(\sqrt{N}-1)\sqrt{N}})$. Then, the prover and verifier run Hyarx-VMV.**Eval**$(\boldsymbol{C}, \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{g}; \mathbf{P})$ with the above specifications. One can readily verify that $p(r) = \boldsymbol{L}\mathbf{P}\boldsymbol{R}^\top$ under these definitions of $\boldsymbol{L}, \boldsymbol{R}, \mathbf{P}$. The rest of the proof follows identically as in the case of multilinear Hyrax-PCS, except that the final **OMA** protocol for the univariate Hyrax-PCS has soundness error at most $n/|\mathbb{F}|$ due to using powers of r instead of uniformly random vectors. □

4.3 Hyrax-VMV as an SMH Protocol

In this section, we reformulate Hyrax-VMV as a **SMH**[1] protocol, mostly for completeness and to showcase that the protocol is essentially identical to Figure 1. However, we believe this to be a useful exercise to motivate our **SMH** model, showcasing that the **OMA** model is not special for Hyrax.

The protocol, presented in Figure 6, is nearly identical to Figure 1. The only change now is that we have parties A, B, and C, along with helper M. We now give vectors $\boldsymbol{L}, \boldsymbol{R}$ as input to B. Moreover, the protocol is simple: A and C share private random string $\boldsymbol{r} \in \mathbb{F}^b$, M sends the same messages to C as in the **OMA** protocol, and now we simply have B send the vectors $\boldsymbol{L}, \boldsymbol{R}$. In other words, B is deterministic; moreover, C simultaneously verifies both a vector-matrix product an inner product. The inner product claim has size b, so Theorem 3.10 implies that B must send $\Omega(b)$ bits to C. For $b = a = \sqrt{N}$, this matches the **OMA** lower bound we obtain for Hyrax-PCS.

5 Bulletproofs: Information-Theoretic Core and Evidence of Optimal Verifier Time

We now turn towards our second main result: giving strong evidence that the Bulletproofs' verification time *is inherently linear*. That is, the underlying information theoretic core of Bulletproofs requires a linear amount

of communication. We prove the following theorem in Section 5.2.1.

Theorem 5.1. *Let $\mathbb{G}$ be a prime-order group with scalar field $\mathbb{F}$, and let $N = 2^n$. Then, for private prover input $\boldsymbol{w} \in \mathbb{F}^N$ and public parameters $\boldsymbol{g} \in \mathbb{G}^N$ for the Bulletproofs proof of knowledge for opening a Pedersen vector commitment over $\mathbb{G}$, there exists a* $\mathbf{SMH}[n]$ *protocol Π which verifies the same function as Bulletproofs with inputs $\boldsymbol{w}, \boldsymbol{g}$ such that $C(\Pi) = \Omega(N)$.*

As a direct corollary, we obtain strong evidence that Bulletproofs-PCS has (nearly) optimal verifier time.

Corollary 5.2. *Let $N = 2^n$, $\mathbb{G}$ be a finite group with scalar field $\mathbb{F}$.*

- *For inputs $f\colon \{0,1\}^n \to \mathbb{F}$, $\boldsymbol{u} \in \mathbb{F}^n$, and $\boldsymbol{g} \in \mathbb{G}^N$ to Bulletproofs-PCS for multilinear polynomials, there exists a* $\mathbf{SMH}[n]$ *protocol Π which verifies the same function as Bulletproofs-PCS with inputs $f, \boldsymbol{u}, \boldsymbol{g}$ such that $C(\Pi) = \Omega(N)$.*
- *For inputs $f \in \mathbb{F}^{<N}[X]$, $u \in \mathbb{F}$, and $\boldsymbol{g} \in \mathbb{G}^N$ to Bulletproofs-PCS for univariate polynomials, there exists a* $\mathbf{SMH}[n]$ *protocol Π which verifies the same function as Bulletproofs-PCS for inputs $f, u, \boldsymbol{g}$ such that $C(\Pi) = \Omega(N)$.*

Moreover, the prescribed Bulletproofs-PCS verifier (for both multilinear and univariate polynomials) performs at most $O(N \operatorname{polylog} |\mathbb{G}|)$ bit-operations, assuming a single $\mathbb{G}/\mathbb{F}$ operation requires $\operatorname{polylog} |\mathbb{G}|$ bit-operations.

We prove Corollary 5.2 in Section 5.3. The rest of this section is dedicated to showing this lower bound. First, we cast Bulletproofs as an instantiation of the sum-check protocol. Then, we capture the information-theoretic core of Bulletproofs as a communication protocol. Finally, we show a linear communication lower bound in this information-theoretic core, giving strong evidence that the Bulletproofs verifier is optimal.

5.1 Bulletproofs is Sum-Check (Revisited)

Let $\boldsymbol{g} \in \mathbb{G}^N$ be (random) group generators and $\boldsymbol{w} \in \mathbb{F}^N$, where $\mathbb{F}$ is the scalar field of $\mathbb{G}$, and $N = 2^n$. At a high-level, Bulletproofs is a proof of knowledge of an opening of the Pedersen commitment $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$; i.e., the prover "knows" a vector $\boldsymbol{w}$ (hidden from the verifier) that is within the Pedersen commitment C under public parameters $\boldsymbol{g}$. It utilizes a "split-and-fold" recursive technique to continuously reduce the claim $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$ for length N vectors $\boldsymbol{g}, \boldsymbol{w}$ to a new claim $C' = \langle \boldsymbol{g}', \boldsymbol{w}' \rangle$ for length $N/2$ vectors $\boldsymbol{g}', \boldsymbol{w}'$, until the final vectors are of length 1 and the prover simply sends the final "folded" witness. Here, $\boldsymbol{g}'$ and $\boldsymbol{w}'$ are a random linear combination of the left and right halves of the (respective) vectors $\boldsymbol{g}$ and $\boldsymbol{w}$, where the randomness is given by the verifier in the protocol.

It was originally observed by Bootle et al. [BCS21] that the Bulletproofs PoK is simply an instantiation of the sum-check protocol [LFK+92] for proving the inner product of two vectors $\boldsymbol{w}, \boldsymbol{g}$. The sum-check proves $C = \sum_{\boldsymbol{c} \in \{0,1\}^n} \widetilde{\boldsymbol{g}}(\boldsymbol{c}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{c})$, where $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{g}}$ are appropriately defined multilinear extensions of $\boldsymbol{w}$ and $\boldsymbol{g}$, respectively. [BCS21] give a different definition of $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{g}}$ then what is "standard" (e.g., [LFK+92]). However, we are interested in presenting Bulletproofs as a sum-check protocol using the "standard" definition of multilinear extensions; i.e., Eqs. (1) and (2).

5.1.1 Bulletproofs as a Standard Sum-Check Protocol

Let $\boldsymbol{g} \in \mathbb{G}^N$ be the public parameters of the Bulletproofs protocol and let $\boldsymbol{w} \in \mathbb{F}^N$ be the private input (the witness) of the Bulletproofs prover. Now, the Bulletproofs protocol precisely corresponds to a sum-check over the polynomial $F(\boldsymbol{X}) = \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X})$. The initial commitment to the vector $\boldsymbol{w}$ under generators $\boldsymbol{g}$ is

Algorithm 5.1: BP-Sum-Check

Public Params: $N = 2^n$, $\mathbb{G}$ with scalar field $\mathbb{F}$, $\boldsymbol{g} \in \mathbb{G}^N$.
Private Input: $\boldsymbol{w} \in \mathbb{F}^N$.

Commit Phase: $(\boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{w} \in \mathbb{F}^N)$
  Output: $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{G}$

Proof of Knowledge Phase: $(C_1 \in \mathbb{G}, \boldsymbol{g} \in \mathbb{G}^N; \boldsymbol{w} \in \mathbb{F}^N)$
  Output: Accept or reject.
  The prover defines n-variate polynomial $F(\boldsymbol{X}) := \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{G}[\boldsymbol{X}]$.
  for $i \in [n]$ **do**
    The prover defines and sends polynomial $q_i(X_i) \colon \mathbb{F} \to \mathbb{G}$ as

$$q_i(X_i) := \sum_{\boldsymbol{c} \in \{0,1\}^{n-i}} F(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}).$$

    The verifier checks $C_i \stackrel{?}{=} q_i(0) + q_i(1)$ and $\deg(q_i) \leqslant 2$; reject if not.
    The verifier samples and sends $\alpha_i \xleftarrow{\$} \mathbb{F}$, and defines $C_{i+1} := q_i(\alpha_i)$.
  The prover sends $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha} \in \mathbb{F}^n$.
  The verifier computes $g = \widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$. The verifier checks $C_{n+1} \stackrel{?}{=} g \cdot w$ and rejects if not equal; otherwise the verifier outputs accept.

$C = \sum_{\boldsymbol{c}} F(\boldsymbol{c}) = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$, which follows by the definition of multilinear extensions: $F(\boldsymbol{c}) = \widetilde{\boldsymbol{g}}(\boldsymbol{c}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{c}) = \boldsymbol{g}_{\boldsymbol{c}} \cdot \boldsymbol{w}_{\boldsymbol{c}}$, where we index the vectors $\boldsymbol{g}$ and $\boldsymbol{w}$ using the binary vector $\boldsymbol{c} \in \{0,1\}^n$ in the natural way. Then, proving knowledge of $\boldsymbol{w}$ such that $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$ amounts to running a sum-check protocol using the polynomial F defined above. We present this formal sum-check protocol in Algorithm 5.1.

Remark 5.3 (Sum-Check with Knowledge Soundness). Bootle et al. [BCS21] explore which properties needed by instances of the sum-check protocol in order to imply knowledge soundness. We do not explore this further in this work, and we make no claims on the connection between soundness in our communication protocols and (knowledge) soundness in our cryptographic protocols.

Remark 5.4 (Optimizing the Proof Size and Verifier Checks). During each round of the sum-check, the prover sends 3 group elements to specify the polynomial q_i. This can be optimized using a standard sum-check optimization: (1) the prover sends $S_i = \{q_i(0), q_i(2)\}$; (2) the verifier sets $q_i(1) := C_i - q_i(0)$ and interpolates $\{q_i(0), q_i(1), q_i(2)\}$ to obtain $q_i(X_i)$; and (3) the verifier sets $C_{i+1} = q_i(\alpha_i)$. This saves one group element per round of the sum-check, and the verifier does not have to check consistency every round. Furthermore, this protocol has the same soundness guarantees as the standard sum-check, and, in fact, sends the same number of group elements per round as the original Bulletproofs protocol.

5.2 Information-Theoretic Core of Bulletproofs

Given Bulletproofs is a standard sum-check protocol, we now turn to distilling the information-theoretic core of Bulletproofs as a communication protocol. Indeed, one motivation for recapturing Bulletproofs as a (standard) sum-check is due to the prevalence of sum-check in interactive proofs, SNARK design, and communication complexity.

In Section 3, we discussed two flavors of communication protocols: the **OMA** model and our new **SMH** model. To capture the IT core of Bulletproofs, we use our new **SMH** model; that is, we'll present

an analogue of Algorithm 5.1 as a **SMH** protocol. Note that while it is possible to capture some form of the information-theoretic core of Bulletproofs as a **OMA** model protocol, we do not believe this framing meaningfully captured the properties of Bulletproofs (see Section 5.4).

5.2.1 Bulletproofs as a SMH Protocol

Recall that a $\mathbf{SMH}[k]$ communication protocol (Section 3.2) involves 4 parties: A, B, C, and M, where A and B communicate "one-way" with C, and C and M execute k-round (or $(2k-1)$-move) interactive proof. We present the formal **SMH** protocol for Algorithm 5.1 in Figure 2. Roughly speaking, the 4 parties in our protocol operate as follows.

- Party A receives input $\boldsymbol{r} \in \mathbb{F}^N$, representing the scalars of the group generators. Moreover, A will be *deterministic*. Intuitively, A represents the public parameters/setup of Bulletproofs.
- Party B receives input $\boldsymbol{w} \in \mathbb{F}^N$, representing the input to the prover of the cryptographic protocol.
- Party M receives both $(\boldsymbol{r}, \boldsymbol{w})$ as input and plays the role of the interactive prover from **Eval Phase** of Algorithm 5.1.
- Party C will play the role of the cryptographic verifier. It will run a n-round sum-check protocol with M, where M tries to convince C of the claim $v = \langle \boldsymbol{r}, \boldsymbol{w} \rangle$. Moreover, C and B will share private randomness $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}^n$ for $N = 2^n$, which is used in the sum-check.
- At the end of this sum-check, A sends their entire vector $\boldsymbol{r}$ to C. B sends the value $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ to C.
- C then performs all the appropriate sum-check verification checks, outputting accept or reject.

Notably, in this information theoretic setting, (1) the vector $\boldsymbol{r}$, which intuitively represents the "hidden" vector in the group generators, is given in the clear to both C and M; and (2) C is guaranteed the correct evaluation $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ from B. (2) captures exactly what happens in the cryptographic setting: in Bulletproofs, the prover sends the value w claimed to be $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. Now, by knowledge soundness and hardness of the discrete-log assumption, w will be the correct value, otherwise either the prover has broken the discrete-log assumption, or we can extract $\boldsymbol{w}$ from the transcript of the interaction. For (1), it would be impossible for C and M to execute the sum-check protocol without M knowing the vector $\boldsymbol{r}$. Similarly, C could not compute $\widetilde{r}(\boldsymbol{\alpha})$ without the vector $\boldsymbol{r}$ (moreover, A cannot send $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ since A does not receive $\boldsymbol{\alpha}$).

With our $\mathbf{SMH}[n]$ protocol (Figure 2) specified for $n = \log(N)$, we now prove Theorem 5.1.

Proof of Theorem 5.1. We begin by establishing the security of Figure 2; namely, it is a valid **SMH** protocol.

Claim 5.5. *Figure 2 is a* $(0, 2n/|\mathbb{F}|)$*-secure* $\mathbf{SMH}[n]$ *protocol for* N*-sized inner products.*

This claim immediately follows from the completeness and soundness of the sum-check protocol. In particular, C always receives the correct $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$, and will always be able to compute the correct $r = \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$, which gives C access to the correct value $F(\boldsymbol{\alpha})$ at the end of the sum-check. Since F has n variables, each with degree at most 2, the soundness error of the sum-check is at most $2n/|\mathbb{F}|$.

With the claim established, Figure 2 is a valid $\mathbf{SMH}[n]$ protocol for verifying the inner product of A's and B's inputs. By construction, party A is deterministic, so Theorem 3.10 tells us that for soundness to hold, party A must send $\Omega(N)$ bits of communication, where N is the length of A's input. □

5.3 Evidence that Bulletproofs-PCS has Optimal Verifier Time

Bulletproofs is readily adaptable into a polynomial commitment scheme for any polynomials which can express evaluation as an inner product; in particular, multilinear and univariate polynomials satisfy this

property. In fact, to obtain a PCS from Bulletproofs, we first give a more general protocol for proving an opening to a Pedersen commitment, and that the inner product of this opening and some public vector is some value. This allows us to prove Corollary 5.2.

Proof of Corollary 5.2. We modify Algorithm 5.1 to obtain a new scheme which satisfies the above property; we present this new algorithm in Algorithm 5.2. As with Algorithm 5.1, we let $\boldsymbol{w} \in \mathbb{F}^N$ be the prover's input and $\boldsymbol{g} \in \mathbb{G}^N$ be the public parameters of the scheme. The commitment phase remains the same: the prover provides $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$. Now, we modify the proof of knowledge phase as follows. First, there is an additional *public input*: $\boldsymbol{y} \in \mathbb{F}^N$, provided by the verifier, and $v \in \mathbb{F}$, provided by the prover. Now, for the proof of knowledge, the prover now proves it knows $\boldsymbol{w}$ such that $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$ *and* $v = \langle \boldsymbol{y}, \boldsymbol{w} \rangle \in \mathbb{F}$. This is actually simple to handle: another instantiation of the sum-check protocol is added in parallel, one for the polynomial $H(\boldsymbol{X}) := \widetilde{\boldsymbol{y}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X})$.

Translating Algorithm 5.2 to an **SMH** protocol, we augment Figure 2 by additionally giving party A the input $\boldsymbol{y}$ and by having party M send the claimed value $v = \langle \boldsymbol{y}, \boldsymbol{w} \rangle$ as part of the first message during the sum-check protocol and run an additional sum-check in parallel for this claim. Moreover, we also specify the function f computed by the $\mathbf{SMH}[n]$ protocol to be $f(\boldsymbol{w}, \boldsymbol{y}, \boldsymbol{r}) = (\langle \boldsymbol{w}, \boldsymbol{y} \rangle, \langle \boldsymbol{w}, \boldsymbol{r} \rangle)$. Then, party A also sends $\boldsymbol{y}$ along with $\boldsymbol{r}$ as her message, and party C additionally augments their checks in line with the checks performed by the verifier in Algorithm 5.2. Clearly, our lower bound still remains for this protocol as now the protocol verifies two inner products in parallel.

To obtain a polynomial commitment scheme, we impose structure on the vectors $\boldsymbol{w}$ and $\boldsymbol{y}$ such that if p is the polynomial in question, then $p(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{y} \rangle$. We first show how to instantiate Algorithm 5.2 to obtain a multilinear polynomial commitment scheme. Let $p \colon \{0,1\}^n \to \mathbb{F}$ be a function and $\widetilde{f}$ be its multilinear extension. Then, set $\boldsymbol{w} = (p(0), p(1), \ldots, p(2^n - 1))$, where we interpret $i \in \{0, 1, \ldots, 2^n - 1\}$ as an n-bit string in the natural way. Simply put, $\boldsymbol{w}$ is the list of evaluations of p (and thus $\widetilde{p}$) on the Boolean hypercube.

Now, if we wish to evaluate $\widetilde{p}$ at some point $\boldsymbol{x} \in \mathbb{F}^n$, we can express this as an inner product between the above $\boldsymbol{w}$ and $\boldsymbol{y} \in \mathbb{F}^N$ defined as $\boldsymbol{y}_i := \mathsf{eq}(\boldsymbol{x}, i - 1)$, where eq is defined in Eq. (2), and we again interpret $i - 1$ as an n-bit string in the natural way. Clearly, by Eq. (1), we have that $\widetilde{p}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{y} \rangle$ for our defined $\boldsymbol{w}$ and $\boldsymbol{y}$. Therefore, Algorithm 5.2 for $\boldsymbol{w}$ and $\boldsymbol{y}$ defined this way for the prover's input p and the verifier's challenge $\boldsymbol{x}$ yields a polynomial commitment scheme for multilinear polynomials.

Now, if p is a univariate polynomial of degree at most $N - 1$, we write $\boldsymbol{w} = (p_0, p_1, \ldots, p_{N-1})$, where p_i is the i^{th}coefficient of p. For any evaluation point x, we write $\boldsymbol{y} = (1, x, x^2, \ldots, x^{N-1})$. Now clearly, we have $p(x) = \langle \boldsymbol{w}, \boldsymbol{y} \rangle$. Doing this transformation in Algorithm 5.2 readily yields a polynomial commitment scheme for univariate polynomials. □

5.4 Why the OMA Model does not Suffice for Bulletproofs

While it is possible to frame Bulletproofs as a **OMA** protocol (e.g., one can apply Lemma 3.9 to Figure 2), we give our reasoning for why we do not believe this is the right model for capturing Bulletproofs. As with the **SMH** protocol for Bulletproofs, we can reimagine Bulletproofs as a $\mathbf{OMA}[n]$ protocol with parties Alice, Bob, and Merlin such that Merlin and Bob will interact, followed by Alice sending a single message to Bob. In this **OMA** protocol, Bob and Merlin execute the same n-round sum-check as in Figure 2, and Bob will perform all the same verification checks as C.

The main question with framing Bulletproofs as an **OMA** protocol is the following: which party gets which input? First, suppose that Alice is given the input $\boldsymbol{w}$ and Bob is given the input $\boldsymbol{r}$. If Alice is deterministic and does not share randomness with Bob, then we obtain our same lower bound as in Figure 2

Algorithm 5.2: BP-Sum-Check-2

Public Params: $N = 2^n$, $\mathbb{G}$ with scalar field $\mathbb{F}$, $\boldsymbol{g} \in \mathbb{G}^N$.
Private Input: $\boldsymbol{w} \in \mathbb{F}^N$.

Commit Phase: $(\boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{w} \in \mathbb{F}^N)$
- **Output:** $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{G}$

Proof of Knowledge: $(C_1 \in \mathbb{G}, \boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{y} \in \mathbb{F}^N, v_1 \in \mathbb{F}; \boldsymbol{w} \in \mathbb{F}^N)$
- **Output:** Accept or reject.
- The prover defines n-variate polynomials $F(\boldsymbol{X}) := \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{G}[\boldsymbol{X}]$ and $H(\boldsymbol{X}) := \widetilde{\boldsymbol{y}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{F}[\boldsymbol{X}]$.
- **for** $i \in [n]$ **do**
  - The prover defines and sends polynomials $q_i(X_i) \colon \mathbb{F} \to \mathbb{G}$ and $p_i(X_i) \colon \mathbb{F} \to \mathbb{F}$ as

$$q_i(X_i) := \sum_{\boldsymbol{c} \in \{0,1\}^{n-i}} F(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}); \quad p_i(X_i) := \sum_{\boldsymbol{c} \in \{0,1\}^{n-i}} H(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}).$$

  - The verifier checks $C_i \stackrel{?}{=} q_i(0) + q_i(1)$, $v_i \stackrel{?}{=} p_i(0) + p_i(1)$, $\deg(q_i) \leqslant 2$, and $\deg(p_i) \leqslant 2$; reject if not.
  - The verifier samples and sends $\alpha_i \xleftarrow{\$} \mathbb{F}$, and defines $C_{i+1} := q_i(\alpha_i)$ and $v_{i+1} = q_{i+1}(\alpha_i)$.
- The prover sends $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha} \in \mathbb{F}^n$.
- The verifier computes $g = \widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ and $y = \widetilde{\boldsymbol{y}}(\boldsymbol{\alpha})$. The verifier checks $C_{n+1} \stackrel{?}{=} g \cdot w$ and $v_{n+1} \stackrel{?}{=} y \cdot w$ and rejects if not equal; otherwise the verifier outputs accept.

due to Corollary 3.11: Alice must send the entire vector $\boldsymbol{w}$ to Bob. In this case, if Alice needs to do this, then Bob does not need to interact with Merlin at all, as Bob can simply compute $\langle \boldsymbol{w}, \boldsymbol{r} \rangle$ from Alice's message. Moreover, this does not make sense if we try to map this behavior back to the cryptographic protocol: it essentially says that if the prover sends their input $\boldsymbol{w}$ to the verifier, then the verifier can compute the inner product. This is just the trivial proof system for computing inner product, and is not what Bulletproofs is doing as a cryptographic protocol.

Now, if Alice is randomized with input $\boldsymbol{w}$ and shares randomness $\boldsymbol{\alpha} \in \mathbb{F}^n$ with Bob, then Bob and Merlin can execute the sum-check protocol and Alice's message is $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. Then, Bob computes $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ and performs the sum-check verification. In this case, this is identical to what the verifier of Bulletproofs does, where the verifier computes $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ to implicitly compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$. Notice that this **OMA** protocol communicates $O(\log(N))$ field elements, but clearly the cryptographic verifier in this case performs $O(N)$ group operations! So we have lost our connection between the communication complexity and the verifier time in the cryptographic protocol. In particular, the work done by Bob to compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ is not captured in the **OMA** model. However, we believe the **OMA** model may be a viable path to obtain lower bounds on *proof sizes* of interactive arguments; we leave this as future work for exploration.

Next, consider switching the inputs of Alice and Bob: let Alice's input be $\boldsymbol{r}$ and Bob's input be $\boldsymbol{w}$. In this case, again if Alice is deterministic, by Corollary 3.11, she must send $\Omega(N)$ bits, and in the protocol, she would send her whole vector $\boldsymbol{r}$ as her message, and Bob can compute $\langle \boldsymbol{r}, \boldsymbol{w} \rangle$ without interacting with Merlin. If Alice shares private randomness $\boldsymbol{\alpha}$ with Bob, then Bob and Merlin can execute the sum-check protocol and have Alice send $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ as her message, allowing Bob to compute the sum-check verification.

However, both of these scenarios does not make sense in the context of the cryptographic protocol. If Bob is playing the role of the cryptographic verifier, then his input being $\boldsymbol{w}$ represents the cryptographic

verifier having the prover's private input $\boldsymbol{w}$ already. So the verifier does not need to check anything or talk to any other parties! Thus, we do not believe that the **OMA** model is sufficient to capture the verifier time in Bulletproofs (or in Dory).

5.5 Rounds vs. Communication Tradeoffs for Bulletproofs

Bulletproofs as originally given in [BCC+16, BBB+18] has $O(\log(N))$ rounds, where $N = 2^n$ is the length of the prover's input, allowing the proof size to be succinct and also of size $O(\log(N))$. As shown by [BCS21] and restated by us in this section, Bulletproofs is an instantiation of the sum-check protocol. This allows us to trade off rounds versus proof size. In this section, we generalize the Bulletproofs sum-check to allow for this trade off. Specifically, for constant r, we show how to construct an r-round Bulletproofs sum-check with proof size $O(r \cdot N^{1/r})$.

An important observation is that the number of rounds in a sum-check protocol corresponds to the number of variables in the polynomial over which the sum-check operates. To construct an r-round sum-check protocol, the underlying polynomial must be defined over r variables. Given that we have N points to interpolate, we require a multivariate polynomial of degree d such that it satisfies the condition $(d+1)^r \geqslant N$ to ensure all the points are interpolated. Hence, we can derive $d = N^{1/r} - 1$. Notably, in the standard Bulletproof (and sum-check) protocol for proving the inner product between two vectors, $r = \log N$ and $d = 1$ which satisfies the condition $(d+1)^r \geqslant N$.

To generalize to any r-round protocol, we modify the extension of vectors and polynomials from multilinear to r-variate, d-degree multivariate polynomials, for degree bound $d = N^{1/r} - 1$. We then apply the sum-check protocol over this multivariate polynomial. This generalization results in an r-round Bulletproofs/sum-check protocol, presented in Algorithm 5.3, where all modifications are highlighted with a gray box. The key change lies in representing the vectors $\boldsymbol{w}, \boldsymbol{g}, \boldsymbol{y}$ using a multivariate polynomial extension. This representation can be constructed in two primary ways. The first method involves direct interpolation: we construct the polynomial $\widetilde{\boldsymbol{w}}(X_1, \ldots, X_r)$ by interpolating over the evaluation points corresponding to the vector entries:

$$\widetilde{\boldsymbol{w}}(i_1, \ldots, i_r) = \boldsymbol{w}[i_1 \cdot (d+1)^{r-1} + i_2 \cdot (d+1)^{r-2} \ldots + i_{r-1}].$$

For all $i_1, \ldots, i_r \in [0, d]$, the index mapping is defined as $\ell = i_1 \cdot (d+1)^{r-1} + i_2 \cdot (d+1)^{r-2} \ldots + i_{r-1}$, which converts the multi-index $(i_1, \ldots, i_r)$ in a $(d+1)$-ary number system into a unique index in the standard decimal system (i.e., $\ell \in [N]$). This mapping ensures a one-to-one correspondence between the vector entries and the multivariate evaluation domain. Using the same approach, we construct the multivariate interpolating polynomials $\widetilde{\boldsymbol{g}}(X_1, \ldots, X_r)$ and $\widetilde{\boldsymbol{y}}(X_1, \ldots, X_r)$ corresponding to the vectors $\boldsymbol{g}$ and $\boldsymbol{y}$, respectively.

The second method for constructing the multivariate polynomial is to define it as a multivariate extension, analogous to the multilinear extension described in Eq. (1) and Eq. (2). In this approach, the polynomial $\widetilde{\boldsymbol{w}}(\cdot)$ can be represented as:

$$\widetilde{w}(X_1, \ldots, X_r) := \sum_{\boldsymbol{c} \in [0,d]^r} \boldsymbol{w}_{\ell(\boldsymbol{c})} \cdot \mathsf{eq}(\boldsymbol{X}, \boldsymbol{c}), \text{ where } \ell(\boldsymbol{c}) = \boldsymbol{c}_1(d+1)^{r-1} + \cdots + \boldsymbol{c}_d \text{ and}$$

$$\mathsf{eq}(\boldsymbol{X}, \boldsymbol{c}) := \prod_{i=1}^{r} \prod_{j=0, j \neq \boldsymbol{c}_i}^{d} \frac{X_i - j}{\boldsymbol{c}_i - j}$$

Other minor modifications include requiring the verifier to compute d evaluations of the polynomials $q_i(\cdot)$ and $p_i(\cdot)$, and to verify that their degrees do not exceed $2d$.

Translating Algorithm 5.3 into a **SMH** protocol follows the same methodology used to convert Algorithm 5.2 into its **SMH** counterpart. The primary differences are how the multivariate extensions $\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{y}}$ are

Algorithm 5.3: BP-Sum-Check-Round-vs-Comm

Public Params: r, $d = N^{1/r} - 1$, $\mathbb{G}$ with scalar field $\mathbb{F}$, $\boldsymbol{g} \in \mathbb{G}^N$.
Private Input: $\boldsymbol{w} \in \mathbb{F}^N$.

Commit Phase: $(\boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{w} \in \mathbb{F}^N)$
- **Output:** $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{G}$

Proof of Knowledge: $(C_1 \in \mathbb{G}, \boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{y} \in \mathbb{F}^N, v_1 \in \mathbb{F}; \boldsymbol{w} \in \mathbb{F}^N)$
- **Output:** Accept or reject.
- The prover defines r-variate degree-d polynomials $\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{y}} \in \mathbb{F}[\boldsymbol{X}]$ and $\widetilde{\boldsymbol{g}} \in \mathbb{G}[\boldsymbol{X}]$ and compute $F(\boldsymbol{X}) := \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{G}[\boldsymbol{X}]$ and $H(\boldsymbol{X}) := \widetilde{\boldsymbol{y}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{F}[\boldsymbol{X}]$.
- **for** $i \in [r]$ **do**
  - The prover defines and sends polynomials $q_i(X_i) \colon \mathbb{F} \to \mathbb{G}$ and $p_i(X_i) \colon \mathbb{F} \to \mathbb{F}$ as

$$q_i(X_i) := \sum_{\boldsymbol{c} \in [0,d]^{r-i}} F(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}) \quad p_i(X_i) := \sum_{\boldsymbol{c} \in [0,d]^{r-i}} H(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}).$$

  - The verifier checks $C_i \stackrel{?}{=} \sum_{a \in [0,d]} q_i(a)$, $v_i \stackrel{?}{=} \sum_{a \in [0,d]} p_i(a)$, $\deg(q_i) \leqslant 2d$, and $\deg(p_i) \leqslant 2d$; reject if not.
  - The verifier samples and sends $\alpha_i \xleftarrow{\$} \mathbb{F}$, and defines $C_{i+1} := q_i(\alpha_i)$ and $v_{i+1} = q_{i+1}(\alpha_i)$.
- The prover sends $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ for $\boldsymbol{\alpha} \in \mathbb{F}^n$.
- The verifier computes $g = \widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ and $y = \widetilde{\boldsymbol{y}}(\boldsymbol{\alpha})$. The verifier checks $C_{n+1} \stackrel{?}{=} g \cdot w$ and $v_{n+1} \stackrel{?}{=} y \cdot w$ and rejects if not equal; otherwise the verifier outputs accept.

defined, as well as the checks performed by party C (similar to the changes made in the protocol). Additionally, the number of messages sent by party M and the number of interaction rounds between party M and party C are modified. However, these changes do not impact the overall complexity analysis of the verifier or Party C. Overall, these changes give a $\mathbf{SMH}[r]$ for computing the inner product function, where M sends $O(r \cdot N^{1/r})$ messages to C.

6 Dory: Information-Theoretic Core and Evidence of Optimal Verifier Time

We now turn our attention to Dory [Lee21]. Essentially, Dory is Bulletproofs in the setting of pairing-friendly groups, but with *logarithmic verification*. This succinct verification crucially relies on the pairing setting, where parties can commit to scalars in $\mathbb{F}$ *as well as* elements of the underlying group(s) $\mathbb{G}$. The seemingly simple addition of this fact allows for succinct verification. We seek to extract the information-theoretic core out of Dory to understand what is happening at a deeper level. We prove the following theorem in Section 6.2.1.

Theorem 6.1. *Let $\mathbb{G}, \mathbb{G}_T$ be prime-order, pairing friendly groups with pairing function $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, with scalar field $\mathbb{F}$, and let $N = 2^n$. Then, for private prover input $\boldsymbol{w} \in \mathbb{G}^N$ and public parameters $\boldsymbol{g} \in \mathbb{G}^N$ for the Dory proof of knowledge for opening a AFGHO vector commitment over $\mathbb{G}$, there exists a $\mathbf{SMH}[n]$ protocol Π which verifies the same function as Dory with inputs $\boldsymbol{w}, \boldsymbol{g}$ such that $C(\Pi) = \Omega(\log(N))$.*

As a direct corollary, we obtain strong evidence that Dory-PCS has (nearly) optimal verifier time.

Corollary 6.2. *Let $\mathbb{G}, \mathbb{G}_T$ be prime-order, pairing friendly groups with pairing function $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, with scalar field $\mathbb{F}$, and let $N = 2^n$.*

- *For inputs $f\colon \{0,1\}^n \to \mathbb{F}$, $\boldsymbol{u} \in \mathbb{F}^n$, and $\boldsymbol{g} \in \mathbb{G}^N$ to Dory-PCS for multilinear polynomials, there exists a* **SMH**$[n]$ *protocol Π which verifies the same function as Dory-PCS with inputs $f, \boldsymbol{u}, \boldsymbol{g}$ such that $C(\Pi) = \Omega(\log(N))$.*
- *For inputs $f \in \mathbb{F}^{<N}[X]$, $u \in \mathbb{F}$, and $\boldsymbol{g} \in \mathbb{G}^N$ to Dory-PCS for univariate polynomials, there exists a* **SMH**$[n]$ *protocol Π which verifies the same function as Dory-PCS for inputs $f, u, \boldsymbol{g}$ such that $C(\Pi) = \Omega(\log(N))$.*

Moreover, the prescribed Dory-PCS verifier (for both multilinear and univariate polynomials) performs at most $O(\log(N)\,\text{polylog}\,|\mathbb{G}|)$ bit-operations, assuming a single $\mathbb{G}/\mathbb{F}$ operation (including pairings) requires $\text{polylog}\,|\mathbb{G}|$ *bit-operations.*

We proceed as follows. First, we examine a simpler version of Dory we call Proto-Dory with $O(\log^2(N))$ verification time (when operating over vectors of length N).[6] We then frame Proto-Dory as another sum-check protocol, as we did with Bulletproofs. From this sum-check, we extract the information theoretic core as a communication protocol in our **SMH** model. Finally, we frame the full Dory protocol (with $O(\log(N))$ verification) as a sum-check, and give its communication analogue in the **SMH** model. To the best of our knowledge, we are the first to formalize the connection between Dory and the sum-check protocol.

Remark 6.3. For simplicity, we state (Proto-)Dory in the context of *symmetric* pairings (i.e., $\mathbb{G}_1 = \mathbb{G}_2$), even though the symmetric external Diffie-Hellman assumption does not hold in this case. We do this for two reasons: (1) it greatly simplifies the presentation; and (2) the information-theoretic core is agnostic to the use of (a)symmetric pairings (since there will be no pairings in this model anyway).

Similar to Bulletproofs, (Proto-)Dory is a proof of knowledge for the statement $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{G}_T$, where $\boldsymbol{g}, \boldsymbol{w} \in \mathbb{G}^N$ for pairing friendly group $\mathbb{G}$, and where $\langle \boldsymbol{g}, \boldsymbol{w} \rangle := \langle \boldsymbol{g}, \boldsymbol{w} \rangle_e$ for pairing function e defined in Section 2. In other words, it is (essentially) the same proof of knowledge statement being proven in Bulletproofs: given generator $h \in \mathbb{G}$ and vector $\boldsymbol{u} \in \mathbb{F}^N$, we can commit to $\boldsymbol{u}$ by setting $\boldsymbol{w}_i = h \cdot \boldsymbol{u}_i$ for all i and computing $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle_e$ At first, Dory proceeds identically as Bulletproofs: it proves the statement $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$ via a sum-check over the polynomial $F(\boldsymbol{X}) = \widetilde{\boldsymbol{g}}(\boldsymbol{X})\widetilde{\boldsymbol{w}}(\boldsymbol{X})$ for appropriately defined multilinear extensions (see Section 2). However, recall in Bulletproofs that the verifier at the end of the protocol must compute the value $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$, which lead to our linear lower bound. In Dory, we take explicit advantage of the pairing setting: Dory instead *forces* the prover to commit to $\widetilde{\boldsymbol{g}}(\boldsymbol{\alpha})$ and prove that the commitment is valid. This is done via another set of random generators $\boldsymbol{\Gamma}$, which will be commitment keys used to commit to $\boldsymbol{g}$ (and its correct foldings) during every round of the sum-check. For a more thorough treatment on the insights of Dory, see [Tha22, Chapter 15].

6.1 Proto-Dory is (a variant of) Sum-Check

We first present a simpler version of Dory, which we call Proto-Dory, with $O(\log^2(N))$ verification for vectors of length $N = 2^n$. The public parameters of Proto-Dory are group generators $\boldsymbol{g} \in \mathbb{G}^N$ and $\boldsymbol{\Gamma} \in \mathbb{G}^{N/2}$. To aid in the sum-check protocol, Proto-Dory uses pairings to precompute some preprocessing polynomials that the verifier will use throughout the sum-check protocol.

Consider the first round of the Proto-Dory sum-check, which is identical to the first round of the Bulletproofs sum-check. The prover is trying to convince the verifier it knows $\boldsymbol{w}$ such that $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$. So

[6] Proto-Dory is presented in [Lee21] as a warm-up protocol to the $O(\log(N))$ verification time Dory.

the prover and verifier engage in the sum-check protocol over the polynomial $F(\boldsymbol{X}) = \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X})$. Let $S_1(X_1)$ be the sum-check polynomial sent by the prover this round. After the first round of the sum-check, the verifier has the claim $C_2 = S_1(\alpha_1)$, which is equivalent to the claim (if the prover is honest) $C_2 = \langle \boldsymbol{g}', \boldsymbol{w}' \rangle$ for $\boldsymbol{g}' = (1-\alpha_1)\boldsymbol{g}_L + \alpha_1 \boldsymbol{g}_R$ and $\boldsymbol{w}' = (1-\alpha_1)\boldsymbol{w}_L + \alpha_1 \boldsymbol{w}_R$.

Intuitively, to overcome the limitations of the discrete-log setting of Bulletproofs, Dory takes advantage of pairings and gives the verifier the ability to compute the commitment $D_1 = \langle \boldsymbol{g}', \boldsymbol{\Gamma} \rangle$ *without explicitly computing* $\boldsymbol{g}'$. Once the verifier has this commitment, then during the next round of the protocol, the prover and verifier engage in *two parallel sum-check protocols*, one for the claim $C_2 = \sum_{\boldsymbol{c}} F(\alpha_1, \boldsymbol{c})$ and another for the claim $D = \langle \boldsymbol{g}', \boldsymbol{\Gamma} \rangle = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{g}}(\alpha_1, \boldsymbol{c}) \cdot \widetilde{\boldsymbol{\Gamma}}(\boldsymbol{c})$.

How does Dory ensure the verifier can compute the claim D_1 above without reading the entire $\boldsymbol{g}$ or $\boldsymbol{\Gamma}$? This is handled during a linear-time preprocessing phase. Notice that the verifier can compute D_1 for any α_1 sampled if given the polynomial $Q(X_1) = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{g}}(X_1, \boldsymbol{c}) \cdot \widetilde{\boldsymbol{\Gamma}}(\boldsymbol{c})$, where $\widetilde{\boldsymbol{g}} \colon \mathbb{G}^n \to \mathbb{G}_T$ and $\widetilde{\boldsymbol{\Gamma}} \colon \mathbb{G}^{n-1} \to \mathbb{G}_T$. Notice further that this polynomial Q has degree at most 1, so the verifier can compute D by reading two group elements (i.e., the description of Q) from the preprocessing.

Proto-Dory's preprocessing enables the verifier to repeat this process each round of the sum-check. In particular, in round i of the sum-check, the prover is trying to prove i parallel sum-check claims. The first claim is still for proving the original inner product claim $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$. The second claim is for proving the first folding $D_1 = \langle \boldsymbol{g}', \boldsymbol{\Gamma} \rangle$. The third claim now makes sure that the prover folds $\boldsymbol{\Gamma}$ to $\boldsymbol{\Gamma}'$ under challenge α_2 correctly, using some other commitment key $\boldsymbol{\Delta} \in \mathbb{G}^{N/4}$; that is, proving the claim $D_2 = \langle \boldsymbol{\Gamma}', \boldsymbol{\Delta} \rangle$. This process is repeated during every round of the sum-check, leading to the $O(\log^2(N))$ proof size.

The formal protocol for Proto-Dory as a sum-check is presented in Algorithm 6.1. First, given public $\boldsymbol{g} \in \mathbb{G}^N$ and $\boldsymbol{\Gamma} \in \mathbb{G}^{N/2}$, the linear-time preprocessing phase constructs n degree-1 polynomials $Q_{j,j}(X_j)$ as follows. Set $\boldsymbol{\Gamma}^{(1)} = \boldsymbol{g}$, $\boldsymbol{\Gamma}^{(2)} = \boldsymbol{\Gamma}$, and for $i \in \{3, \ldots, n\}$, define $\boldsymbol{\Gamma}^{(i)} = \boldsymbol{\Gamma}_L^{(i-1)} \in \mathbb{G}^{N/2^{i-i}}$. Finally, set $\Gamma^{(n+1)} = \boldsymbol{\Gamma}_L^{(n)} \in \mathbb{G}$. Here, the subscript L represents the left half of the vector. Then we define $Q_{j,j}(X_j) = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{\Gamma}}^{(j)}(X_j, \boldsymbol{c}) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(\boldsymbol{c})$ for all $j \in [n]$. The preprocessing outputs all $Q_{j,j}$ along with $\Gamma^{(n+1)}$.[7]

At the start of round i of the sum-check, the prover (in parallel) proves i sum-check claims. First, it computes $S_i(X_i) = \sum_{\boldsymbol{c}} F(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c})$ for verifier challenges $\alpha_j \xleftarrow{\$} \mathbb{F}$ for $j \in [i-1]$; this corresponds to the sum-check for the original claim $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle$. Then, for each $j \in [i-1]$, the prover additionally computes polynomials $Q_{j,i}(X_i) = \sum_{\boldsymbol{c}} P_j(\alpha_j, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c})$, where $P_j(\boldsymbol{X}_{[j:n]}) = \widetilde{\boldsymbol{\Gamma}}^{(j)}(\boldsymbol{X}_{[j:n]}) \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(\boldsymbol{X}_{[j+1:n]})$. Here, $\boldsymbol{X}_{[j:n]}$ denotes the variables $X_j, \ldots, X_n$. Intuitively, each polynomial $Q_{j,i}$ corresponds to the sum-check claim $D_j = \langle (1-\alpha_j)\boldsymbol{\Gamma}^{(j)} + \alpha_j \boldsymbol{\Gamma}^{(j)}, \boldsymbol{\Gamma}^{(j+1)} \rangle = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{\Gamma}}^{(j)}(\alpha_j, \boldsymbol{c}) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(\boldsymbol{c})$. At the end of round i, the verifier updates its i sum-check claims as $C_{i+1} = S_i(X_i)$ and $D_{j,i+1} = Q_{j,i}(\alpha_i)$ for $j \in [i-1]$. Additionally, the verifier now adds a new sum-check claim $D_{i,i+1} = Q_{i,i}(\alpha_i)$ by using the preprocessing polynomial $Q_{i,i}$, increasing the number of sum-check claims from i to $i+1$ for the next round.

In the final round of the protocol, the prover sends the claimed evaluations $w = \widetilde{w}(\boldsymbol{\alpha})$ and $\Gamma^{(j)} = \widetilde{\boldsymbol{\Gamma}}^{(j)}(\alpha_j, \ldots, \alpha_n)$ for $j \in [n]$. Using these values and the preprocessing value $\Gamma^{(n+1)}$, the verifier checks consistency of the values sent by the prover with the set of claims it has been tracking C_{n+1} and $D_{j,n+1}$ for $j \in [n]$. Intuitively, because Dory is a proof of knowledge, the prover is forced to send the correct values unless it can break the discrete-log assumption, otherwise (1) the verifier will reject (with high probability), or (2) we can extract the prover's witness from the protocol transcript.

[7] Notice that $Q_{j,j}(0) = \langle \boldsymbol{\Gamma}_L^{(j)}, \boldsymbol{\Gamma}^{(j+1)} \rangle$ and $Q_{j,j}(1) = \langle \boldsymbol{\Gamma}_R^{(j)}, \boldsymbol{\Gamma}^{(j+1)} \rangle$. That is, $Q_{j,j}$ contains the commitments to the left and right halves of $\boldsymbol{\Gamma}^{(j)}$ under the commitment key $\boldsymbol{\Gamma}^{(j+1)}$, which is exactly the output of the Dory preprocessing [Lee21].

Algorithm 6.1: Proto-Dory-Sum-Check

Public Parameters: $N = 2^n$, pairing-friendly $(\mathbb{G}, \mathbb{G}_T, \mathbb{F})$, $\boldsymbol{g} \in \mathbb{G}^N$, $\boldsymbol{\Gamma} \in \mathbb{G}^{N/2}$.
Private Input: $\boldsymbol{w} \in \mathbb{G}^N$.

Preprocessing $(\boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{\Gamma} \in \mathbb{G}^{N/2})$
- **Output:** $\Gamma^{(n+1)} \in \mathbb{G}$ and $\{Q_{j,j}(X_j)\}_{j\in[n]}$.
- Define $\boldsymbol{\Gamma}^{(1)} := \boldsymbol{g}$, $\boldsymbol{\Gamma}^{(2)} := \boldsymbol{\Gamma}$ and set $\boldsymbol{\Gamma}^{(i)} := \boldsymbol{\Gamma}_L^{(i-1)} \in \mathbb{G}^{N/2^{i-1}}$ for $i \in \{3, \ldots, n\}$. Set $\Gamma^{(n+1)} := \boldsymbol{\Gamma}_L^{(n)} \in \mathbb{G}$.
- For $j \in [n]$, define polynomials

$$Q_{j,j}(X_j) = \sum_{\boldsymbol{c}\in\{0,1\}^{n-j}} \widetilde{\boldsymbol{\Gamma}}^{(j)}(X_j, \boldsymbol{c}) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(\boldsymbol{c}) \in \mathbb{G}_T[X_j].$$

- Output $\Gamma^{(n+1)}$ and $\{Q_{j,j}(X_j)\}_{j\in[n]}$.

Commit Phase $(\boldsymbol{w}, \boldsymbol{g} \in \mathbb{G}^N)$
- **Output:** $C = \langle \boldsymbol{g}, \boldsymbol{w} \rangle \in \mathbb{G}_T$

Proof of Knowledge $(C \in \mathbb{G}_T, \boldsymbol{g} \in \mathbb{G}^N, \boldsymbol{\Gamma} \in \mathbb{G}^{N/2}, \{Q_{j,j}(X_j) \in \mathbb{G}_T[X_j]\}_{j\in[n]}\ \Gamma^{(n+1)} \in \mathbb{G}; \boldsymbol{w} \in \mathbb{G}^N)$
- **Output:** Accept or reject.
- The prover defines $P_j(\boldsymbol{X}_{[j:n]}) = \widetilde{\boldsymbol{\Gamma}}^{(j)}(\boldsymbol{X}_{[j:n]}) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(\boldsymbol{X}_{[j+1:n]}) \in \mathbb{G}_T[\boldsymbol{X}_{[j:n]}]$ for $j \in [i-1]$ and $S(\boldsymbol{X}) = \widetilde{\boldsymbol{g}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{w}}(\boldsymbol{X}) \in \mathbb{G}_T[\boldsymbol{X}]$.
- The verifier defines $C_1 := C$.
- **for** $i \in [n]$ **do**
  - The prover defines polynomials

$$S_i(X_i) := \sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} S(\alpha_1, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c})$$
$$Q_{j,i}(X_i) := \sum_{\boldsymbol{c}\in\{0,1\}^{n-j-i-1}} P_j(\alpha_j, \ldots, \alpha_{i-1}, X_i, \boldsymbol{c}) \quad \forall j \in [i-1].$$

  - The prover sends $S_i(X_i)$ and $\{Q_{j,i}(X_i)\}_{j\in[i-1]}$.
  - The verifier checks $C_i \stackrel{?}{=} S_i(0) + S_i(1)$, $\{Q_{j,i}(0) + Q_{j,i}(1) \stackrel{?}{=} D_{j,i}\}_{j\in[i-1]}$, and that all polynomials have degree at most 2; reject if not.
  - The verifier samples and sends $\alpha_i \xleftarrow{\$} \mathbb{F}$.
  - The verifier defines $C_{i+1} = S_i(\alpha_i)$ and $\{D_{j,i+1} = Q_{j,i}(\alpha_i)\}_{j\in[i]}$.
- The prover sends $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ and $\{\Gamma^{(j)} = \widetilde{\boldsymbol{\Gamma}}^{(j)}(\alpha_j, \ldots, \alpha_n)\}_{j\in[n]}$.
- The verifier checks $C_{n+1} \stackrel{?}{=} w \cdot \Gamma^{(1)}$ and $\{D_{j,n+1} \stackrel{?}{=} \Gamma^{(j)} \cdot \Gamma^{(j+1)}\}_{j\in[n]}$, rejecting if not all are true.

6.1.1 Proto-Dory as a SMH Communication Protocol

As with Bulletproofs, the benefit of framing Proto-Dory as a sum-check protocol is that we can cleanly capture the information-theoretic core using our new SMH communication model. We present the formal $\mathbf{SMH}[n]$ protocol for Proto-Dory in Figure 3. This protocol is nearly identical to Figure 2, with two notable exceptions. First, we have M send an additional message $\widehat{R}(X_n)$ in round $i = n$, and second is that A is now *randomized*.

In more detail, party A of Figure 3 has the same input as in our Bulletproofs protocol (Figure 2): $\boldsymbol{r} \in \mathbb{F}^N$ representing the scalar vector "hidden" within $\boldsymbol{g} \in \mathbb{G}^N$. Now, we allow A, B, and M to sample shared randomness $\boldsymbol{\gamma}^{(i+1)} \xleftarrow{\$} \mathbb{F}^{N/2^{n-i}}$ for $i \in [n]$, intuitively representing the "hidden" scalars within vectors $\boldsymbol{\Gamma}^{(i+1)} \in \mathbb{G}^{N/2^{n-i}}$. There is one major difference when translating Algorithm 6.1 to Figure 3 regarding how the commitment keys $\boldsymbol{\Gamma}^{(i)}$ are sampled and how the private shared randomness $\boldsymbol{\gamma}^{(i)}$ are sampled. In Figure 3, the vectors $\boldsymbol{\gamma}^{(i)}$ for $i \in [2, n+1]$ are sampled uniformly at random, whereas in Algorithm 6.1 only the first vector $\boldsymbol{\Gamma} \in \mathbb{G}^{N/2}$ was random (and the remaining were the left halves). In the actual specification of Dory [Lee21], this vector is chosen as $\boldsymbol{\Gamma}^{(2)} = \boldsymbol{g}_L$, with each subsequent commitment key being set as $\boldsymbol{\Gamma}^{(j+1)} = \boldsymbol{\Gamma}_L^{(j)}$, with the subscript L denoting the left half of the vector. Note that in the actual definitions of our models (Definitions 3.1 and 3.6), parties must sample shared randomness *before* receiving any inputs to the protocol. However, suppose we do allow parties to sample shared randomness after receiving their inputs.

When selecting vectors $\boldsymbol{\Gamma}^{(i)}$ in Algorithm 6.1, they do not need to be sampled randomly, as the hardness

of discrete-log, intuitively, lets us treat each vector $\boldsymbol{\Gamma}^{(i)}$ as random, even though they are structured. This, however, *is not possible* in the information theoretic setting. In particular, one can demonstrate an attack in the information theoretic setting when using such structured vectors; see Remark 6.5 for more discussion. Notably, if we mimic how Dory chooses the intermediate keys, where they are simply a function of the initial input $\boldsymbol{g}$, then this would translate to a SMH protocol with a *deterministic* party A, and thus we would run into the lower bound of Theorem 3.10. The aforementioned attack lines up with the proof of Theorem 3.10: party M can run the protocol with respect to some $\boldsymbol{r}' \neq \boldsymbol{r}$ but party A's messages would send the same messages for both $\boldsymbol{r}$ and $\boldsymbol{r}'$ as their input. However, if we select the commitment key $\boldsymbol{\gamma}^{(2)}$ uniformly and independently at random from $\boldsymbol{\gamma}^{(1)} = \boldsymbol{r}$ in the communication protocol, then we can set $\boldsymbol{\gamma}^{(i+1)} = \boldsymbol{\gamma}_L^{(i)}$ for $i \geqslant 2$ and retain the soundness of the protocol; for ease of presentation, we choose all $\boldsymbol{\gamma}^{(i+1)}$ uniformly at random.

Now, why do we allow A to share this same randomness with B? Allowing A and B to share this same randomness enables B to send the correct values $\gamma^{(i+1)} = \widetilde{\boldsymbol{\gamma}}^{(i+1)}(\alpha_{i+1}, \ldots, \alpha_n)$ for $i \in [n-1]$ at the end of the protocol, along with the correct $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. Intuitively, this is capturing the proof of knowledge property of Dory, which forces the cryptographic prover to send these correct values, otherwise we can extract the prover's witness $\boldsymbol{w}$ from the transcript of the interaction, or the prover has broken the discrete-log assumption.

The final subtle difference between Algorithm 6.1 and Figure 3 is the extra message $\widetilde{R}(X_n)$ sent by M in round n of the sum-check. We enforce this to again, intuitively, try to capture the cryptography in Proto-Dory. In Algorithm 6.1, the discrete-log assumption and proof of knowledge property force the cryptographic prover to correctly send $\gamma^{(i)} = \widetilde{\boldsymbol{\gamma}}^{(i+1)}(\alpha_i, \ldots, \alpha_n)$ for $i \in [n]$. By giving party B the random vectors $\boldsymbol{\gamma}^{(j)}$, we allow party B to emulate this behavior for all but the value $\gamma^{(1)}$. Under the specification of Figure 3, $\boldsymbol{\gamma}^{(1)} = \boldsymbol{r}$, which is A's input. Notice that if A could share their input with B, then there is no need to interact with M and no need to share any random strings with any party: B can simply send $\langle \boldsymbol{w}, \boldsymbol{r} \rangle$ directly. The randomness allows B to aid C in the computation of $\langle \boldsymbol{w}, \boldsymbol{r} \rangle$ even though B does not have access to A's input $\boldsymbol{r}$.

However, to compute all the necessary checks for the sum-check protocol, C is missing one final piece of information: the value $\gamma^{(1)} = \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$. We give C access to this value through M sending $\widehat{R}(X_n)$, which is claimed to be the polynomial $\widetilde{\boldsymbol{r}}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$. This is to simplify the analysis; we could also add one more round of interaction where M receives α_n and sends $\widehat{\gamma}^{(n+1)}$ which is claimed to be $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$. In this case, M could try to cheat by computing $\widehat{\gamma}^{(n+1)} = v^{(n+1)}/w$ (M can compute w now that it has received α_n), but we explicitly disallow this because in the cryptographic setting, this would translate to the cryptographic prover finding a non-trivial discrete-log relation. So, intuitively, forcing M to send $\widehat{R}(X_n)$ is trying to also enforce that the cryptographic prover would be forced to send $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ honestly.

Note also that C actually doesn't need to receive this value from M. Instead, C could recover $\gamma^{(1)}$ through computing $\gamma^{(1)} := v^{(n+1)}/w$, then running all the remaining checks with respect to the honestly received $\gamma^{(j)}$ for $j \geqslant 2$. But then C is doing something impossible in the cryptographic setting, since computing this $\gamma^{(1)}$ is actually happening in the exponents of the cryptographic scheme, and this would require C to find a non-trivial discrete-log relation. So we explicitly disallow this and have C compute this value from M's extra message in the last round of the sum-check protocol.

With the discussion on the discrepancies between Algorithm 6.1 and Figure 3 complete, we now show that the protocol of Figure 3 is a valid **SMH** protocol.

Claim 6.4. *Let $N = 2^n$ and let $f(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^N$ and any finite field $\mathbb{F}$. Then, Figure 3 is a $(0, O(n^2)/|\mathbb{F}|))$-secure $\mathbf{SMH}[n]$ protocol for f.*

Proof. Perfect completeness follows from the sum-check protocol. Similarly, soundness follows almost directly from the sum-check protocol. During round i of the sum-check, C is verifying i claims, and constructing claim $i+1$ using random challenge α_i. Each of these claims is about a degree at most 2

polynomial, so the total soundness error contribution is at most $2i/|\mathbb{F}|$. Taking a union bound over all rounds $i \in [n]$ gives us a $2n^2/|\mathbb{F}|$ upper bound.

However, in the last round of the sum-check, M sends $\widehat{R}(X_n)$, which is claimed to be $\widetilde{\boldsymbol{r}}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$. If M has been cheating so far and has lucked out with all the sum-check verification checks for $i \in [n]$, the final checks that need to pass are $\widetilde{v}^{(n+1)} \stackrel{?}{=} w \cdot \widehat{R}(\alpha_n)$ and $\widehat{D}_{j,n+1} \stackrel{?}{=} \gamma^{(j)} \cdot \gamma^{(j+1)}$ for $j \in [n]$. Here, we use "$\widehat{\cdot}$" to denote potentially tampered values sent by M. Notice that since $\widehat{R}$ is sent without knowing α_n, M never learns the value of $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$. So the probability the first check passes if $\widehat{R}$ is not honestly sent is $1/|\mathbb{F}|$. Notice also that the values $\widehat{D}_{j,n+1}$ are defined by polynomials sent by M without knowing the value α_n, while $\gamma^{(j+1)}$ is honestly sent by B for $j \in [n]$. So the probability each of these checks pass is also $1/|\mathbb{F}|$. This gives a final soundness upper bound of $O(n^2/|\mathbb{F}|)$. □

Remark 6.5. We demonstrate how to break the soundness of the above protocol if we choose $\boldsymbol{\gamma}^{(1)} = \boldsymbol{r}$ and $\boldsymbol{\gamma}^{(i+1)} = \boldsymbol{\gamma}_L^{(i)}$ for all $i \in [n]$. Consider $N = 4$ and $n = 2$. Let $\boldsymbol{r} = \boldsymbol{\gamma}^{(1)} = (r_{00}, r_{01}, r_{10}, r_{11})$. Then, we have $\boldsymbol{\gamma}^{(2)} = (r_{00}, r_{01})$ and $\boldsymbol{\gamma}^{(3)} = r_{00}$.

We construct another vector $\boldsymbol{s} \neq \boldsymbol{r}$ such that party A's messages to C are identical when A's input is $\boldsymbol{s}$ and $\boldsymbol{r}$. Let $\boldsymbol{s} = (s_{00}, s_{01}, s_{10}, s_{11})$ First, note that A sends two polynomials $Q_{1,1}, Q_{2,2}$, and the value $\gamma^{(3)}$. Since we want A to send the same messages for both input $\boldsymbol{s}, \boldsymbol{r}$, we set $s_{00} = r_{00} = \gamma^{(3)}$. Now, the polynomial $Q_{2,2}(X_2)$ can be expressed as $Q_{2,2}(X_2) = (1 - X_2)\langle \boldsymbol{\gamma}_L^{(2)}, \boldsymbol{\gamma}^{(3)} \rangle + X_2 \langle \boldsymbol{\gamma}_R^{(2)}, \boldsymbol{\gamma}^{(3)} \rangle = (1 - X_2) r_{00}^2 + X_2 r_{01} r_{00}$. Since we have set $s_{00} = r_{00}$, in order for this message to be identical for input $\boldsymbol{s}$, we must set $s_{01} = r_{01}$.

Finally, we demonstrate how to set $s_{10} \neq r_{10}$ or $s_{11} \neq r_{11}$ such that the message $Q_{1,1}(X_1)$ remains the same. From the definition of $Q_{1,1}(X_1)$, it can be shown that for A's input $\boldsymbol{r}$:

$$\begin{aligned} Q_{1,1}(X_1) &= (1 - X_1)\langle \boldsymbol{\gamma}_L^{(1)}, \boldsymbol{\gamma}^{(2)} \rangle + X_1 \langle \boldsymbol{\gamma}_R^{(1)}, \boldsymbol{\gamma}^{(2)} \rangle \\ &= (1 - X_1)\langle \boldsymbol{\gamma}^{(2)}, \boldsymbol{\gamma}^{(2)} \rangle + X_1 \langle \boldsymbol{\gamma}_R^{(1)}, \boldsymbol{\gamma}^{(2)} \rangle \\ &= (1 - X_1)\langle (r_{00}, r_{01}), (r_{00}, r_{01}) \rangle + X_1 \langle (r_{10}, r_{11}), (r_{00}, r_{01}) \rangle \\ &= (1 - X_1)(r_{00}^2 + r_{01}^2) + X_1 (r_{10} r_{00} + r_{11} r_{01}) \end{aligned}$$

Now, if A's input is $\boldsymbol{s}$ such that $s_{00} = r_{00}$ and $s_{01} = r_{01}$, we can write $Q_{1,1}(X_1)$ as

$$Q_{1,1}(X_1) = (1 - X_1)(r_{00}^2 + r_{01}^2) + X_1 (s_{10} r_{00} + s_{11} r_{01}).$$

In order for A to send the same message for both $\boldsymbol{r}$ and $\boldsymbol{s}$ as input, the above equations tell us that the following must hold:

$$r_{10} r_{00} + r_{11} r_{01} = s_{10} r_{00} + s_{11} r_{01}.$$

Letting s_{10} be any value such that $s_{10} \neq r_{10}$, we can rewrite the above (assuming $r_{01} \neq 0$) as

$$s_{11} = \frac{r_{00}(r_{10} - s_{10}) + r_{11} r_{01}}{r_{01}}.$$

So party A sends the same $Q_{1,1}, Q_{2,2}$, and $\gamma^{(3)}$ to party C with inputs $\boldsymbol{r} \neq \boldsymbol{s}$, breaking soundness of the protocol (in fact, demonstrating the exact attack derived in the proof of Theorem 3.10).

6.2 Full Dory as Sum-Check

Building upon Proto-Dory, the full Dory protocol compresses the proofs from $O(\log^2(N))$ size to $O(\log(N))$ size. While the full details of this compression are (somewhat) messy, the intuition behind the compression is quite simple. Recall that in round i of the Proto-Dory sum-check, the prover and verifier were running i parallel instances of the sum-check protocol in order to prove the claims $\langle \boldsymbol{g}, \boldsymbol{w} \rangle$, $\langle (1-\alpha_1)\boldsymbol{g}_L + \alpha_1 \boldsymbol{g}_R, \boldsymbol{\Gamma}^{(2)} \rangle$, $\langle (1-\alpha_2)\boldsymbol{\Gamma}_L^{(2)} + \alpha_2 \boldsymbol{\Gamma}_R^{(2)}, \boldsymbol{\Gamma}^{(3)} \rangle, \ldots, \langle (1-\alpha_{i-1})\boldsymbol{\Gamma}_L^{(i-1)} + \alpha_{i-1}\boldsymbol{\Gamma}_R^{(i-1)}, \boldsymbol{\Gamma}^{(i)} \rangle$. This essentially lead to the $O(\log^2(N))$ sized proofs. The key insight to the full Dory protocol is that we can still force the prover to track *all* of these claims, but only using a *constant number* of claims (specifically 3) per round. We outline the intuition behind this insight.

After round 1 of Proto-Dory, we went from one claim to two claims. Rather than have the verifier track these claims, Dory actually has the verifier specify an additional random value β_1 and force the prover to "fold" together all the claims via a random linear combination (through β_1). Formally, the full Dory sum-check is simultaneously proving 3 claims. Let $\boldsymbol{w}, \boldsymbol{g}, \boldsymbol{\Gamma}^{(1)} \in \mathbb{G}^N$. Then Dory is a sum-check protocol for simultaneously proving the claims $C_1 = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{w}}(\boldsymbol{c})\widetilde{\boldsymbol{g}}(\boldsymbol{c})$, $C_2 = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{w}}(\boldsymbol{c})\widetilde{\boldsymbol{\Gamma}}^{(1)}(\boldsymbol{c})$, and $C_3 = \widetilde{\boldsymbol{g}}(\boldsymbol{c})\widetilde{\boldsymbol{\Gamma}}^{(1)}(\boldsymbol{c})$. Let $\boldsymbol{\Gamma}^{(2)} \in \mathbb{G}^{N/2}$.

The Dory sum-check will reduce the above claims to three new claims of the form $C_1' = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{w}'}(\boldsymbol{c})\widetilde{\boldsymbol{g}'}(\boldsymbol{c})$, $C_2' = \sum_{\boldsymbol{c}} \widetilde{\boldsymbol{w}'}(\boldsymbol{c})\widetilde{\boldsymbol{\Gamma}}^{(2)}(\boldsymbol{c})$, and $C_3' = \widetilde{\boldsymbol{g}'}(\boldsymbol{c})\widetilde{\boldsymbol{\Gamma}}^{(2)}(\boldsymbol{c})$, where for verifier challenges α_1, β_1, we have $\boldsymbol{w}' = (1-\alpha_1)(\boldsymbol{w} + \beta_1\boldsymbol{\Gamma}^{(1)})_L + \alpha_1(\boldsymbol{w} + \beta_1\boldsymbol{\Gamma}^{(1)})_R$ and $\boldsymbol{g}' = (1-\alpha_1)(\boldsymbol{g} + \beta_1\boldsymbol{\Gamma}^{(1)})_L + \alpha_1(\boldsymbol{g} + \beta_1\boldsymbol{\Gamma}^{(1)})_R$. As polynomials, notice that

$$\widetilde{\boldsymbol{w}'}(\boldsymbol{X}_{[2:n]}) = \widetilde{\boldsymbol{w}}(\alpha_1, \boldsymbol{X}_{[2:n]}) + \beta_1\widetilde{\boldsymbol{\Gamma}}^{(1)}(\alpha_1, \boldsymbol{X}_{[2:n]}) \quad \widetilde{\boldsymbol{g}'}(\boldsymbol{X}_{[2:n]}) = \widetilde{\boldsymbol{g}}(\alpha_1, \boldsymbol{X}_{[2:n]}) + \beta_1\widetilde{\boldsymbol{\Gamma}}^{(1)}(\alpha_1, \boldsymbol{X}_{[2:n]}).$$

So Dory has forced the prover to track the original polynomials $\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{g}}$, and the preprocessing polynomial $\widetilde{\boldsymbol{\Gamma}}^{(1)}$. This continues recursively each round: after round 2, the prover is forced to track $\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{g}}, \widetilde{\boldsymbol{\Gamma}}^{(1)}$, and $\widetilde{\boldsymbol{\Gamma}}^{(2)}$. Meanwhile, the verifier only needs to track a constant number of claims, and to update these claims, the verifier uses the preprocessing polynomials, along with two additional auxiliary polynomials given by the prover each round of the protocol. We give the full Dory protocol as a sum-check protocol in Algorithm 6.2.

6.2.1 Full Dory as a SMH Protocol

With the specification of full Dory as a sum-check protocol, we now move extract the information-theoretic core of Dory as a **SMH** protocol, which will be similar to Figure 3. The full protocol is presented in Figure 7, and we give an overview of the protocol here. Intuitively, A is capturing the public setup of Dory, B is capturing the (most) of the information that the cryptographic prover sends at the end of the protocol, and M is capturing a potentially dishonest prover interacting with the cryptographic verifier.

Before the start of the protocol, B and C share private randomness $\boldsymbol{\alpha}, \boldsymbol{\beta} \xleftarrow{\$} \mathbb{F}^n$, and A, B, and M share private randomness $\boldsymbol{\gamma}^{(i)} \xleftarrow{\$} \mathbb{F}^{N/2^{n+1-i}}$ for $i \in [n+1]$. Now, C and M engage in a sum-check protocol that is identical to the proof of knowledge of Algorithm 6.2, except that in round $i = n$, M additionally sends a degree at most 1 polynomial $\widehat{R}(X_n)$, claimed to be the polynomial $\widetilde{\boldsymbol{r}}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$. Next, A computes preprocessing polynomials $Q_j, \widehat{Q}_j$ identically to the preprocessing in Algorithm 6.2, except using the sampled vectors $\boldsymbol{\gamma}^{(j)}$. A sends these polynomials and the value $\gamma^{(n+1)}$ to C. B computes and sends $w = \widetilde{\boldsymbol{w}}(\boldsymbol{\alpha})$ and $\gamma^{(j)} = \widetilde{\boldsymbol{\gamma}}^{(j)}(\alpha_j, \ldots, \alpha_n)$ for $j \in [2, n]$. Finally, C performs all the verification checks of Algorithm 6.2,, plus one more check with the polynomial $\widehat{R}(X_n)$: C checks that this polynomial has degree at most 1. Moreover, C sets $W^{(n+1)} = w + \sum_{j\in[n]} \beta_j \gamma^{(j)}$ and $R^{(n+1)} = \widehat{R}(\alpha_n) + \sum_{j\in[n]} \beta_j \gamma^{(j)}$. Note that with standard algebraic manipulation, if M is honest, it can be seen that $W^{(n+1)}$ and $R^{(n+1)}$ computed in this way are identical to the honest prover messages $W^{(n+1)}, G^{(n+1)}$ from Algorithm 6.2.

Algorithm 6.2: Full-Dory-Sum-Check (Preprocessing and Commit)

Public Parameters: Instance size $N = 2^n$, prime-order pairing-friendly group $\mathbb{G}$ with scalar field $\mathbb{F}$, and random $\boldsymbol{g}, \boldsymbol{\Gamma} \in \mathbb{G}^N$.
Private Input: Prover has private input $\boldsymbol{w} \in \mathbb{G}^N$.

Preprocessing
- **Input:** Group elements $\boldsymbol{g}, \boldsymbol{\Gamma}$ from public parameters.
- **Output:** $\{(Q_j(X_j), \widehat{Q}_j(X_j))\}_{j\in[n]}$ of degree at most 1 and $\Gamma^{(n+1)} \in \mathbb{G}$.
- Define $\boldsymbol{\Gamma}^{(1)} := \boldsymbol{\Gamma}$ and set $\boldsymbol{\Gamma}^{(i)} := \boldsymbol{\Gamma}_L^{(i-1)} \in \mathbb{G}^{N/2^{i-1}}$ for $i \in \{2, \ldots, n\}$. Set $\Gamma^{(n+1)} := \boldsymbol{\Gamma}_L^{(n)}$.
- For $j \in [n]$, define polynomials $P_j(X_j, \ldots, X_n) := \widetilde{\boldsymbol{\Gamma}}^{(j)}(X_j, \ldots, X_n) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j)}(X_j, \ldots, X_n)$, $\widehat{P}_j(X_j, \ldots, X_n) := \widetilde{\boldsymbol{\Gamma}}^{(j)}(X_j, \ldots, X_n) \cdot \widetilde{\boldsymbol{\Gamma}}^{(j+1)}(X_{j+1}, \ldots, X_n)$, and

$$Q_j(X_j) = \sum_{\boldsymbol{c}\in\{0,1\}^{n-j}} P_j(X_j, \boldsymbol{c}) \qquad \widehat{Q}_j(X_j) = \sum_{\boldsymbol{c}\in\{0,1\}^{n-j}} \widehat{P}_j(X_j, \boldsymbol{c}).$$

- Output $\Gamma^{(n+1)}$ and $\{Q_j(X_j), \widehat{Q}_j(X_j)\}_{j\in[n]}$.

Commit Phase $(\boldsymbol{w}, \boldsymbol{g}, \boldsymbol{\Gamma} \in \mathbb{G}^N)$
- **Output:** Commitments $C_1 = \langle \boldsymbol{g}, \boldsymbol{w} \rangle, C_2 = \langle \boldsymbol{w}, \boldsymbol{\Gamma} \rangle, C_3 = \langle \boldsymbol{g}, \boldsymbol{\Gamma} \rangle \in \mathbb{G}_T$

Proof of Knowledge
- **Input:** Commitments $C_1, C_2, C_3 \in \mathbb{G}_T$, public parameters $\boldsymbol{g}, \boldsymbol{\Gamma} \in \mathbb{G}^N$, public preprocessing $(\Gamma^{(n+1)}, \{(Q_j(X_j), \widehat{Q}_j(X_j))\})$; private input $\boldsymbol{w} \in \mathbb{G}$.
- **Output:** Verifier accepts or rejects.
- The verifier defines $C_j^{(1)} := C_j$ for $j \in [3]$. The prover defines $J^{(n+1)} = \Gamma^{(n+1)}$ and

$$W^{(1)}(\boldsymbol{X}) = \widetilde{\boldsymbol{w}}(\boldsymbol{X}), \qquad G^{(1)}(\boldsymbol{X}_{[1,n]}) = \widetilde{\boldsymbol{g}}(\boldsymbol{X}), \qquad \{J^{(k)}(\boldsymbol{X}_{[k:n]}) = \widetilde{\boldsymbol{\Gamma}}^{(k)}(\boldsymbol{X}_{[k:n]})\}_{k\in[n]}.$$

- **for** $i \in [n]$ **do**
  - The prover defines and sends polynomials

$$S_1^{(i)}(X_i) = \textstyle\sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} W^{(i)}(X_i, \boldsymbol{c}) \cdot G^{(i)}(X_i, \boldsymbol{c}) \qquad S_2^{(i)}(X_i) = \sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} W^{(i)}(X_i, \boldsymbol{c}) \cdot J^{(i)}(X_i, \boldsymbol{c})$$
$$S_3^{(i)}(X_i) = \textstyle\sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} G^{(i)}(X_i, \boldsymbol{c}) \cdot J^{(i)}(X_i, \boldsymbol{c}) \qquad S_4^{(i)}(X_i) = \sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} W^{(i)}(X_i, \boldsymbol{c}) \cdot J^{(i+1)}(\boldsymbol{c})$$
$$S_5^{(i)}(X_i) = \textstyle\sum_{\boldsymbol{c}\in\{0,1\}^{n-i}} G^{(i)}(X_i, \boldsymbol{c}) \cdot J^{(i+1)}(\boldsymbol{c}).$$

  - The verifier checks $C_j^{(i)} \stackrel{?}{=} S_j^{(i)}(0) + S_j^{(i)}(1)$ for $j \in [3]$ and $\deg(S_j^{(i)}) \leqslant 2$ for $j \in [5]$.
  - The verifier samples and sends $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{F}$.
  - The verifier defines

$$C_1^{(i+1)} = S_1^{(i)}(\alpha_i) + \beta_i \left(S_2^{(i)}(\alpha_i) + S_3^{(i)}(\alpha_i)\right) + \beta_i^2 Q_i(\alpha_i) \qquad C_2^{(i+1)} = S_4^{(i)}(\alpha_i) + \beta_i \widehat{Q}_i(\alpha_i)$$
$$C_3^{(i+1)} = S_5^{(i)}(\alpha_i) + \beta_i \widehat{Q}_i(\alpha_i)$$

  - The prover defines

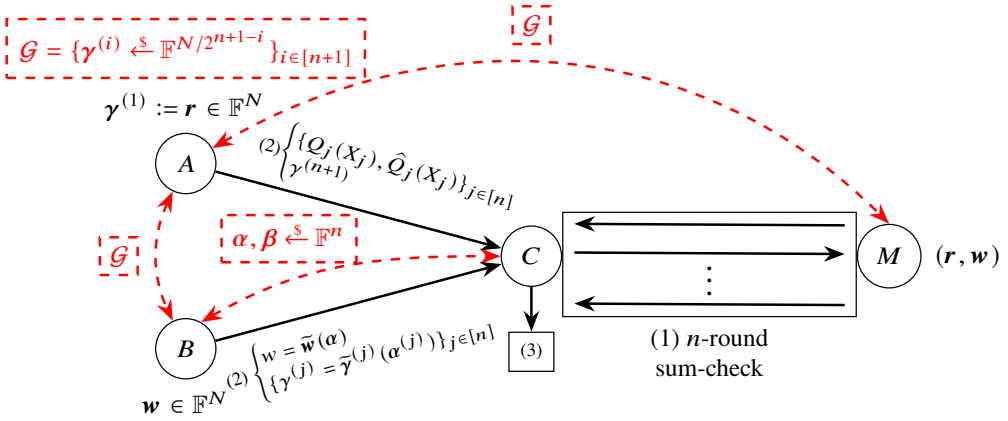
$$W^{(i+1)}(\boldsymbol{X}_{[i+1,n]}) = W^{(i)}(\alpha_i, \boldsymbol{X}_{[i+1,n]}) + \beta_i J^{(i)}(\alpha_i, \boldsymbol{X}_{[i+1,n]})$$
$$G^{(i+1)}(\boldsymbol{X}_{[i+1,n]}) = G^{(i)}(\alpha_i, \boldsymbol{X}_{[i+1,n]}) + \beta_i J^{(i)}(\alpha_i, \boldsymbol{X}_{[i+1,n]}).$$

- The prover sends $w^{(n+1)} = W^{(n+1)}$ and $g^{(n+1)} = G^{(n+1)}$ to the verifier.
- The verifier checks $C_1^{(n+1)} \stackrel{?}{=} w^{(n+1)} \cdot g^{(n+1)}$, $C_2^{(n+1)} \stackrel{?}{=} w^{(n+1)} \cdot \Gamma^{(n+1)}$, and $C_3^{(n+1)} \stackrel{?}{=} g^{(n+1)} \cdot \Gamma^{(n+1)}$; reject if not all hold.

Given the above high-level overview of Figure 7, we now prove Theorem 6.1.

Proof of Theorem 6.1. We begin by showing that Figure 7 is a valid **SMH** protocol.

Claim 6.6. *Let $N = 2^n$ and let $f(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^N$ and any finite field. Then, Figure 7 is a*



(0) B and C sample shared private randomness $\alpha, \beta \in \mathbb{F}^n$. A, B, and M sample shared private randomness $\gamma^{(i)} \in \mathbb{F}^{N/2^{n+1-i}}$ for $i \in [n+1]$.
(1) M and C engage in an n-round sum-check protocol. M defines $W^{(1)}(X) = \widetilde{w}(X)$, $R^{(1)}(X) = \widetilde{r}(X)$, and $J^{(1)}(X) = \widetilde{\gamma}^{(1)}(X)$. For $i \in [n]$:

- M defines $J^{(i+1)}(X_{[i+1:n]}) = \widetilde{\gamma}^{(i+1)}(X_{[i+1:n]})$ and the following 5 polynomials.

$$S_1^{(i)}(X_i) = \sum_{c} W^{(i)}(X_i, c) R^{(i)}(X_i, c) \quad S_2^{(i)}(X_i) = \sum_{c} W^{(i)}(X_i, c) J^{(i)}(X_i, c) \quad S_3^{(i)}(X_i) = \sum_{c} R^{(i)}(X_i, c) J^{(i)}(X_i, c)$$
$$S_4^{(i)}(X_i) = \sum_{c} W^{(i)}(X_i, c) J^{(i+1)}(c) \quad S_5^{(i)}(X_i) = \sum_{c} R^{(i)}(X_i, c) J^{(i+1)}(c).$$

- M sends $\{S_i^{(j)}(X_i)\}_{j\in[5]}$. If $i = 1$, then also send $v_1 = \langle r, w\rangle$, $v_2 = \langle r, \gamma^{(1)}\rangle$, $v_3 = \langle w, \gamma^{(1)}\rangle$. Else if $i = n$, then also send $\widehat{R}(X_n) = \widetilde{r}(\alpha_1, \ldots, \alpha_{n-1}, X_n)$
- C sends α_i, β_i if $i < n$.
- If $i < n$, then M defines new polynomials

$$W^{(i+1)}(X_{[i+1:n]}) = W^{(i)}(\alpha_i, X_{[i+1:n]}) + \beta_i J^{(i)}(\alpha_i, X_{[i+1:n]}) \quad R^{(i+1)}(X_{[i+1:n]}) = R^{(i)}(\alpha_i, X_{[i+1:n]}) + \beta_i J^{(i)}(\alpha_i, X_{[i+1:n]}).$$

(2) A computes $Q_j(X_j) = \sum_c \widetilde{\gamma}^{(j)}(X_j, c)^2$ and $\widehat{Q}_j(X_j) = \sum_c \widetilde{\gamma}^{(j)}(X_j, c) \cdot \widetilde{\gamma}^{(j+1)}(c)$ for $j \in [n]$ and sends $(\{Q_j, \widehat{Q}_j\}_{j\in[n]}, \gamma^{(n+1)})$. B computes and sends $w = \widetilde{w}(\alpha)$ and $\gamma^{(i)} = \gamma^{(j+1)}(\alpha^{(j+1)})$ for $j \in [n-1]$ and $\alpha^{(j+1)} = (\alpha_{j+1}, \ldots, \alpha_n)$.
(3) C does the following.

- Set $v_1^{(1)} = v_1$, $v_2^{(1)} = v_2$, and $v_3^{(3)} = v_3$. For each $i \in [n]$:
  - Check if $v_j^{(i)} \stackrel{?}{=} S_j^{(i)}(0) + S_j^{(i)}(1)$ for $j \in [3]$ and that $\deg(S_j^{(i)}) \leqslant 2$ for $j \in [5]$.
  - Set $v_1^{(i+1)} = S_1^{(i)}(\alpha_i) + \beta_i(S_2^{(i)}(\alpha_i) + S_3^{(i)}(\alpha_i)) + \beta_i^2 Q_i(\alpha_i)$, $v_2^{(i+1)} = S_4^{(i)}(\alpha_i) + \beta_i \widehat{Q}_i(\alpha_i)$, and $v_3^{(i+1)} = S_5^{(i)}(\alpha_i) + \beta_i \widehat{Q}_i(\alpha_i)$.
- Check that $\deg(\widehat{R}) \leqslant 1$.
- Compute $W^{(n+1)} = w + \sum_{j\in[n]} \beta_j \gamma^{(j)}$ and $R^{(n+1)} = \widehat{R}(\alpha_n) + \sum_{j\in[n]} \beta_j \gamma^{(j)}$.
- Check if $v_1^{(n+1)} \stackrel{?}{=} W^{(n+1)} \cdot R^{(n+1)}$, $v_2^{(n+1)} \stackrel{?}{=} W^{(n+1)} J^{(n+1)}$, and $v_3^{(n+1)} \stackrel{?}{=} R^{(n+1)} J^{(n+1)}$.
- Accept if and only if all checks pass.

Figure 7: Dory-Sum-Check as a $\mathbf{SMH}[n]$ Protocol for $N = 2^n$. The red dashed line indicates private randomness shared between the respective parties. The protocol executes (1), followed by (2), followed by (3).

$(0, O(n/|\mathbb{F}|))$-*secure* $\mathbf{SMH}[n]$ *protocol for* f.

Completeness immediately follows from completeness of the sum-check protocol, along with the observation that taking a random linear combination of two multilinear extensions (i.e., how M computes $W^{(i+1)}$ and $R^{(i+1)}$) is itself a multilinear extension (so all polynomials in the protocol have degree at most 2).

Soundness also immediately follows from the sum-check protocol. During round i of the sum-check, since C is taking a random β_i combination of polynomials, the soundness error incurred is proportional to the largest power of β_i taken in the combination. Notably, C updates $v_1^{(i+1)}$ using β_i^2, and the other two values

with only β_i. Thus, the total contribution to the soundness error of this round is $4/|\mathbb{F}|$. Additionally, C could sample a bad α_i as a root of one of the polynomials it is checking consistency with. Since all the polynomials have degree at most 2, this again contributes at most $2/|\mathbb{F}|$ to the soundness error per polynomial. So we have an upper bound of $14/|\mathbb{F}|$ for the soundness error in round i. Now, in the final round of the protocol, when M sends the additional message $\widehat{R}(X_n)$, M sends this without knowing the final challenge α_n, so it only passes C's check using $\widehat{R}(\alpha_n)$ with probability at most $1/|\mathbb{F}|$. A union bound over all rounds i plus the soundness error of this final check gives us the final bound of $O(n/|\mathbb{F}|)$, establishing the claim. With the claim established, the rest of the proof is trivial. The specified protocol has n rounds, and by definition, the amount of information exchanged between parties is $\Omega(n) = \Omega(\log(N))$ bits. □

6.3 Evidence that Dory-PCS has Optimal Verifier Time

As with Bulletproofs, (Proto-)Dory is readily adaptable into a polynomial commitment scheme for any polynomials which can express evaluation as an inner product (again, multilinear and univariate polynomials satisfy this property). We present this modification below as we prove Corollary 6.2.

Proof of Corollary 6.2. We modify Algorithm 6.2 to obtain a new scheme satisfying the above property, but for brevity specify it here. The modification we present is first for univariate polynomials; the case of multilinear polynomials is similar.

First, suppose that the prover has private input a univariate polynomial $p \in \mathbb{F}^{<N}[X]$. To commit to p, let $\boldsymbol{p} = (p_0, \ldots, p_{N-1}) \in \mathbb{F}^N$ denote the coefficients of p. We additionally add a new generator $h \in \mathbb{G}$ to the public parameters. Then, the prover first computes a vector $\boldsymbol{w} = (h \cdot p_0, \ldots, h \cdot p_{N-1}) \in \mathbb{G}$ and commits to p via $C_1 = \langle \boldsymbol{w}, \boldsymbol{g} \rangle$ for public parameters $\boldsymbol{g}$, identical to Algorithm 6.2 (note it also computes $C_2 = \langle \boldsymbol{w}, \boldsymbol{\Gamma} \rangle$ as well; C_3 remains the same). Now, at the start of the proof of knowledge phase of Algorithm 6.2, the verifier additionally specifies an evaluation point $r \in \mathbb{F}$. The prover and verifier then run the proof of knowledge of Algorithm 6.2 with the prover using $\boldsymbol{w}$ defined above, and additionally adding another sum-check polynomial $F(\boldsymbol{X}) := \widetilde{\boldsymbol{p}}(\boldsymbol{X}) \cdot \widetilde{\boldsymbol{r}}(\boldsymbol{X})$, where $\boldsymbol{r} := (1, r, r^2, r^3, \ldots, r^{N-1})$, and with the prover additionally sending $y \in \mathbb{F}$ in the first round, claimed to be the value $\langle \boldsymbol{p}, \boldsymbol{r} \rangle = p(r)$.

In the final round of the proof of knowledge (round $n+1$), the prover additionally sends a value $q \in \mathbb{F}$, claimed to be $\widetilde{\boldsymbol{p}}(\boldsymbol{\alpha})$, and a value $\sigma^{(n+1)} \in \mathbb{G}$, claimed to be $\sum_i \beta_i J^{(i)}(\alpha_i, \ldots, \alpha_n)$. The verifier additionally checks that $(h \cdot q) + \sigma^{(n+1)} = w^{(n+1)}$, where $w^{(n+1)} \in \mathbb{G}$ is claimed to be $\widetilde{\boldsymbol{w}}(\boldsymbol{\alpha}) + \sum_i \beta_i \widetilde{\boldsymbol{\Gamma}}^{(i)}(\alpha_i, \ldots, \alpha_n)$ sent by the prover (i.e., the final folded sum-check polynomial $W^{(n+1)}$). Now, let $y^{(n+1)}$ be the final value obtained from the prover and verifier running the sum-check on the polynomial $F(\boldsymbol{X})$, claimed to be $F(\boldsymbol{\alpha})$. In order to verify this check as well, the verifier needs to compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ and check $q \cdot \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha}) = y^{(n+1)}$.

The verifier takes advantage of the tensor structure of $\boldsymbol{r}$ to efficiently compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ in only logarithmic time. To see this, first consider the value $\sum_{\boldsymbol{c}} \widetilde{\boldsymbol{r}}(\alpha_1, \boldsymbol{c})$. Using the definition of multilinear extensions, we see that this value is computed as

$$\sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \widetilde{\boldsymbol{r}}(\alpha_1, \boldsymbol{c}) = \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \sum_{\boldsymbol{b} \in \{0,1\}^n} \boldsymbol{r}_{\boldsymbol{b}} \cdot \mathsf{eq}((\alpha_1, \boldsymbol{c}), \boldsymbol{b}) = \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}(1 - \alpha_1) + \boldsymbol{r}_{1\boldsymbol{c}} \alpha_1.$$

By definition of $\boldsymbol{r}$, $0\boldsymbol{c}$ indexes the left half of $\boldsymbol{r}$ and $1\boldsymbol{c}$ indexes the right half of $\boldsymbol{r}$. The left half is given by $\boldsymbol{r}_L = (1, r, r^2, \ldots, r^{N/2-1})$, and the right half id given by $\boldsymbol{r}_R = (r^{N/2}, r^{N/2+1}, \ldots, r^{N-1})$. Notice that

$\boldsymbol{r}_R = r^{N/2} \cdot \boldsymbol{r}_L$. So we can rewrite the summation as

$$\sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}(1-\alpha_1) + \boldsymbol{r}_{1\boldsymbol{c}}\alpha_1 = \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}(1-\alpha_1) + r^{N/2} \cdot \boldsymbol{r}_{0\boldsymbol{c}} \cdot \alpha_1$$
$$= [(1-\alpha_1) + \alpha_1 r^{N/2}] \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}.$$

We can continue this expansion and manipulation for each round i with respect to $\sum_{\boldsymbol{c}} \widetilde{\boldsymbol{r}}(\alpha_1, \ldots, \alpha_i, \boldsymbol{c})$, which yields the following equality:

$$\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha}) = \prod_{i=1}^{n} (1-\alpha_i) + \alpha_i \cdot r^{N/2^i}.$$

This equation tells us that the verifier only needs the powers $r, r^2, r^4, \ldots, r^{N/2}$ for $N = 2^n$ in order to compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$, which by repeated squaring are computable in $O(\log(N))$ field operations, so $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ is computable in $O(\log(N))$ field operations, culminating in the final verification check by the verifier $y^{(n+1)} \stackrel{?}{=} q \cdot \widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$.

Translating this modification of Algorithm 6.2 to a $\mathbf{SMH}[n]$ protocol, we additionally give party A the input $r \in \mathbb{F}$. Then, C and M execute the same protocol as Figure 7, adding in the minor modifications above. The only additional change is that A now additionally sends C the vector $(r, r^2, r^4, \ldots, r^{N/2}) \in \mathbb{F}^{\log(N)}$ so C can perform the additional checks above. The protocol, being $\log(N)$ rounds, still has a communication lower-bound of $\Omega(\log(N))$, which is (nearly) matched by the communication sent in the $\mathbf{SMH}[n]$ protocol and the work of the verifier in Algorithm 6.2 (up to polylog $|\mathbb{G}|$ factors).

For multilinear polynomials, the modifications to Algorithm 6.2 are nearly identical to the above for univariate polynomials. Now, given $p\colon \{0,1\}^n \to \mathbb{F}$ as a multilinear polynomial, the prover commits to the vector $\boldsymbol{p} = (p(0), p(1), \ldots, p(N-1))$ (the vector of evaluations) by computing $\boldsymbol{w} = (h \cdot p(0), h \cdot p(1), \ldots, h \cdot p(N-1))$ as above. The only other modification we need to address is how to efficiently compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ for a vector $\boldsymbol{r} \in \mathbb{F}^N$ such that $\langle \boldsymbol{p}, \boldsymbol{r} \rangle = p(\boldsymbol{u})$ for $\boldsymbol{u} \in \mathbb{F}^n$. First, using Eq. (1) (and work we already did in Section 5.3), the vector $\boldsymbol{r}$ is defined as $\boldsymbol{r}_i = \mathsf{eq}(\boldsymbol{u}, i-1)$ for $i \in [N]$.

We just need to argue that $\boldsymbol{r}$ also has sufficient tensor structure to allow the verifier to compute $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ in logarithmic time. As before, first consider $\sum_{\boldsymbol{c}} \widetilde{\boldsymbol{r}}(\alpha_1, \boldsymbol{c})$, which can be written as

$$\sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \widetilde{\boldsymbol{r}}(\alpha_1, \boldsymbol{c}) = \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}(1-\alpha_1) + \boldsymbol{r}_{1\boldsymbol{c}}\alpha_1.$$

Let $\boldsymbol{u}^{(1)} := \boldsymbol{u}$ and let $\boldsymbol{u}^{(2)} := (\boldsymbol{u}_2, \ldots, \boldsymbol{u}_n)$. By definition of $\boldsymbol{r}$, we see that

$$\boldsymbol{r}_{0\boldsymbol{c}} = \mathsf{eq}(\boldsymbol{u}^{(1)}, 0\boldsymbol{c}) = (1-\boldsymbol{u}_1) \cdot \prod_{j=1}^{n} (1-\boldsymbol{u}_j)(1-\boldsymbol{c}_j) + \boldsymbol{u}_j\boldsymbol{c}_j = (1-\boldsymbol{u}_1) \cdot \mathsf{eq}(\boldsymbol{u}^{(2)}, \boldsymbol{c})$$
$$\boldsymbol{r}_{1\boldsymbol{c}} = \mathsf{eq}(\boldsymbol{u}^{(1)}, 1\boldsymbol{c}) = \boldsymbol{u}_1 \cdot \prod_{j=1}^{n} (1-\boldsymbol{u}_j)(1-\boldsymbol{c}_j) + \boldsymbol{u}_j\boldsymbol{c}_j = \boldsymbol{u}_1 \cdot \mathsf{eq}(\boldsymbol{u}^{(2)}, \boldsymbol{c}).$$

Thus, we can simplify the summation as

$$\sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \boldsymbol{r}_{0\boldsymbol{c}}(1-\alpha_1) + \boldsymbol{r}_{1\boldsymbol{c}}\alpha_1 = \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} (1-\boldsymbol{u}_1)\mathsf{eq}(\boldsymbol{u}^{(2)}, \boldsymbol{c})(1-\alpha_1) + \boldsymbol{u}_1\mathsf{eq}(\boldsymbol{u}^{(2)}, \boldsymbol{c})\alpha_1$$
$$= [(1-\alpha_1)(1-\boldsymbol{u}_1) + \alpha_1\boldsymbol{u}_1] \sum_{\boldsymbol{c} \in \{0,1\}^{n-1}} \mathsf{eq}(\boldsymbol{u}^{(2)}, \boldsymbol{c}).$$

As before, continuing to expand the summation for $\widetilde{\boldsymbol{r}}(\alpha_1, \ldots, \alpha_i, \boldsymbol{c})$, we can write $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ as:

$$\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha}) = \prod_{i=1}^{n} (1 - \alpha_i)(1 - \boldsymbol{u}_i) + \alpha_i \boldsymbol{u}_i.$$

Clearly, this computation can be done in $O(\log(N)) = O(n)$ field operations. Finally, in the $\mathbf{SMH}[n]$ protocol, party A receives the additional input $\boldsymbol{u} \in \mathbb{F}^n$ and simply sends $\boldsymbol{u}$ to C at the end of the protocol, and C computes $\widetilde{\boldsymbol{r}}(\boldsymbol{\alpha})$ as specified above, yielding the same upper and lower bounds as in the univariate case. □

7 Conclusions and Open Problems

In this work, we initiate the study of the connection between SNARK verification times and the communication complexity of the underlying information-theoretic cores of the SNARKs. Our work examines the polynomial commitment schemes of Hyrax, Bulletproofs, and Dory, extracts their information-theoretic cores as communication protocols, and lower bounds the number of bits that must be exchanged between parties in these protocols. This connection gives us strong evidence that these polynomial commitment schemes all have optimal verifier time. Moreover, our results unfortunately support evidence that the Bulletproofs verifier is inherently linear-time, possibly settling a long-standing open question.

Natural future directions in this line of work are extending the results in this paper to other polynomial commitment schemes. For example, based on groups of unknown order [BFS20, BHR+21], based on the FRI protocol [BBH+18, ZCF24, ACF+24], based on Ligero/Brakedown [AHI+23, BFH+20, BBH+22, GLS+23, BFK+24], and the KZG scheme [KZG10]. There is also potential to extend our results to the verification costs of *lookup arguments* [PT12, PT13, GW20, STW24, AST24]. Finally, it is natural to question whether the connections we made in this paper can extend to the Groth16 SNARK [Gro16]. We believe these are all exciting and promising future directions to explore.

Another direction is towards understanding the connections we began to make in this paper. Specifically, can we strengthen the connections between polynomial commitment schemes and communication protocols, and get provable guarantees to actually prove optimality of the verifier of polynomial commitments (and potentially other cryptographic protocols) using communication complexity? To the best of our knowledge, we are the first to explore such a connection and the first to consider verifier time lower bounds, so there is immense potential in this line of work to gain a deeper understanding of polynomial commitments and SNARKs at a deeper level.

We also believe that our **SMH** model is of independent interest in the field of communication complexity. Though we can simulate any **SMH** model protocol as an **OMA** model protocol, it is natural to wonder if we can obtain different (better or worse) lower bounds on computing functions in the **SMH** model versus what is known for the **OMA** model.

References

[Abl96] Farid M. Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 157(2):139–159, 1996. DOI: 10.1016/0304-3975(95)00157-3. URL: https://doi.org/10.1016/0304-3975(95)00157-3.

[ACF+24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 380–413. Springer, 2024. DOI: 10.1007/978-3-031-68403-6_12. URL: https://doi.org/10.1007/978-3-031-68403-6_12.

[ACL+22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based snarks: publicly verifiable, preprocessing, and recursively composable - (extended abstract). In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 102–132. Springer, 2022. URL: https://doi.org/10.1007/978-3-031-15979-4_4.

[AFG+16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *J. Cryptol.*, 29(2):363–421, 2016. DOI: 10.1007/S00145-014-9196-7. URL: https://doi.org/10.1007/s00145-014-9196-7.

[AFL+24] Martin R. Albrecht, Giacomo Fenzi, Oleksandra Lapiha, and Ngoc Khanh Nguyen. SLAP: succinct lattice-based polynomial commitments from standard assumptions. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 90–119. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-58754-2_4.

[AHI+23] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *Des. Codes Cryptogr.*, 91(11):3379–3424, 2023. URL: https://doi.org/10.1007/s10623-023-01222-8.

[AST24] Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: snarks for virtual machines via lookups. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024. DOI: 10.1007/978-3-031-58751-1_1. URL: https://doi.org/10.1007/978-3-031-58751-1_1.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009. DOI: 10.1145/1490270.1490272. URL: https://doi.org/10.1145/1490270.1490272.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018. DOI: 10.1109/SP.2018.00020. URL: https://doi.org/10.1109/SP.2018.00020.

[BBH+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018*, volume 107 of *LIPIcs*, 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI: 10.4230/LIPICS.ICALP.2018.14. URL: https://doi.org/10.4230/LIPIcs.ICALP.2018.14.

[BBH+22] Laasya Bangalore, Rishabh Bhadauria, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. On black-box constructions of time and space efficient sublinear arguments from symmetric-key primitives. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 417–446. Springer, 2022. DOI: 10.1007/978-3-031-22318-1_15. URL: https://doi.org/10.1007/978-3-031-22318-1_15.

[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 Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 327–357. Springer, 2016. DOI: 10.1007/978-3-662-49896-5_12. URL: https://doi.org/10.1007/978-3-662-49896-5_12.

[BCF+22] David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Functional commitments for circuits from falsifiable assumptions. *IACR Cryptol. ePrint Arch.*:1365, 2022. URL: https://eprint.iacr.org/2022/1365.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 742–773. Springer, 2021. DOI: 10.1007/978-3-030-84242-0_26. URL: https://doi.org/10.1007/978-3-030-84242-0_26.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020. URL: https://doi.org/10.1145/3372297.3417893.

[BFK+24] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 276–307. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-68403-6_9.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-45721-1_24.

[BFS86] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347. IEEE Computer Society, 1986. DOI: 10.1109/SFCS.1986.15. URL: https://doi.org/10.1109/SFCS.1986.15.

[BGK+03] László Babai, Anna Gál, Peter G. Kimmel, and Satyanarayana V. Lokam. Communication complexity of simultaneous messages. *SIAM J. Comput.*, 33(1):137–166, 2003. DOI: 10.1137/S0097539700375944. URL: https://doi.org/10.1137/S0097539700375944.

[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 - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 168–197. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-64378-2_7.

[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 Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152. Springer, 2021. DOI: 10.1007/978-3-030-84259-8_5. URL: https://doi.org/10.1007/978-3-030-84259-8_5.

[BHV+23] Rishabh Bhadauria, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Wenxuan Wu, and Yupeng Zhang. Private polynomial commitments and applications to MPC. In *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 127–158. Springer, 2023. URL: https://doi.org/10.1007/978-3-031-31371-4_5.

[BJS+25] Elette Boyle, Abhishek Jain, Sacha Servan-Schreiber, and Akshayaram Srinivasan. Simultaneous-message and succinct secure computation. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part V*, volume 15605 of *Lecture Notes in Computer Science*, pages 207–239. Springer, 2025. DOI: 10.1007/978-3-031-91092-0_8. URL: https://doi.org/10.1007/978-3-031-91092-0_8.

[BK97] László Babai and Peter G. Kimmel. Randomized simultaneous messages: solution of a problem of yao in communication complexity. In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity, Ulm, Germany, June 24-27, 1997*, pages 239–246. IEEE Computer Society, 1997. DOI: 10.1109/CCC.1997.612319. URL: https://doi.org/10.1109/CCC.1997.612319.

[BKL95] László Babai, Peter G. Kimmel, and Satyanarayana V. Lokam. Simultaneous messages vs. communication. In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 1995. DOI: 10.1007/3-540-59042-0_88. URL: https://doi.org/10.1007/3-540-59042-0_88.

[CCG+14] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 687–706. SIAM, 2014. DOI: 10.1137/1.9781611973402.52. URL: https://doi.org/10.1137/1.9781611973402.52.

[CCM+14] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Algorithms*, 11(1):7:1–7:30, 2014. DOI: 10.1145/2636924. URL: https://doi.org/10.1145/2636924.

[CCM+19] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and arthur-merlin communication. *SIAM J. Comput.*, 48(4):1265–1299, 2019. DOI: 10.1137/17M112289X. URL: https://doi.org/10.1137/17M112289X.

[CDG+24] Graham Cormode, Marcel Dall'Agnol, Tom Gur, and Chris Hickey. Streaming zero-knowledge proofs. In Rahul Santhanam, editor, *39th Computational Complexity Conference, CCC 2024, Ann Arbor, MI, USA, July 22-25, 2024*, volume 300 of *LIPIcs*, 2:1–2:66. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. DOI: 10.4230/LIPICS.CCC.2024.2. URL: https://doi.org/10.4230/LIPIcs.CCC.2024.2.

[CFL83] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 94–99. ACM, 1983. DOI: 10.1145/800061.808737. URL: https://doi.org/10.1145/800061.808737.

[CMN+24] Valerio Cini, Giulio Malavolta, Ngoc Khanh Nguyen, and Hoeteck Wee. Polynomial commitments from lattices: post-quantum security, fast verification and transparent setup. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 207–242. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-68403-6_7.

[CMT13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013. DOI: 10.1007/S00453-011-9598-Y. URL: https://doi.org/10.1007/s00453-011-9598-y.

[FMN24] Giacomo Fenzi, Hossein Moghaddas, and Ngoc Khanh Nguyen. Lattice-based polynomial commitments: towards asymptotic and concrete efficiency. *J. Cryptol.*, 37(3):31, 2024. URL: https://doi.org/10.1007/s00145-024-09511-8.

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 839–842. North-Holland, 1977.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. DOI: 10.1007/3-540-47721-7_12. URL: https://doi.org/10.1007/3-540-47721-7_12.

[Gab24] Ariel Gabizon. https://x.com/rel_zeta_tech/status/1794842766708535544, May 2024. See also https://x.com/rel_zeta_tech/status/1794780789789376774.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. DOI: 10.1007/978-3-031-38545-2_7. URL: https://doi.org/10.1007/978-3-031-38545-2_7.

[Gro09] Jens Groth. Homomorphic trapdoor commitments to group elements. *IACR Cryptol. ePrint Arch.*:7, 2009. URL: http://eprint.iacr.org/2009/007.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016. DOI: 10.1007/978-3-662-49896-5_11. URL: https://doi.org/10.1007/978-3-662-49896-5_11.

[GS24] Prantar Ghosh and Vihan Shah. New lower bounds in merlin-arthur communication and graph streaming verification. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, Berkeley, CA, USA, January 30 - February 2, 2024*, volume 287 of *LIPIcs*, 53:1–53:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. DOI: 10.4230/LIPICS.ITCS.2024.53. URL: https://doi.org/10.4230/LIPIcs.ITCS.2024.53.

[GW20] Ariel Gabizon and Zachary J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*:315, 2020. URL: https://eprint.iacr.org/2020/315.

[HSS24] Intak Hwang, Jinyeong Seo, and Yongsoo Song. Concretely efficient lattice-based polynomial commitment from standard assumptions. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 414–448. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-68403-6_13.

[Kla03] Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 118–134. IEEE Computer Society, 2003. DOI: 10.1109/CCC.2003.1214415. URL: https://doi.org/10.1109/CCC.2003.1214415.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997. ISBN: 978-0-521-56067-2.

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discret. Math.*, 5(4):545–557, 1992. DOI: 10.1137/0405044. URL: https://doi.org/10.1137/0405044.

[KS93] Tracy Kimbrel and Rakesh K. Sinha. A probabilistic algorithm for verifying matrix products using $O(n^2)$ time and $\log_2 n + O(1)$ random bits. *Inf. Process. Lett.*, 45(2):107–110, 1993. DOI: 10.1016/0020-0190(93)90224-W. URL: https://doi.org/10.1016/0020-0190(93)90224-W.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. DOI: 10.1007/978-3-642-17373-8_11. URL: https://doi.org/10.1007/978-3-642-17373-8_11.

[Lee21] Jonathan Lee. Dory: efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 2021. DOI: 10.1007/978-3-030-90453-1_1. URL: https://doi.org/10.1007/978-3-030-90453-1_1.

[LFK+92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. DOI: 10.1145/146585.146605. URL: https://doi.org/10.1145/146585.146605.

[Lu22] Frank Y. C. Lu. Kevlar: transparent, efficient, polynomial commitment scheme with logarithmic verification and communication costs on efficient groups. *IACR Cryptol. ePrint Arch.*:702, 2022. URL: https://eprint.iacr.org/2022/702. Withdrawn.

[Lu24] Frank Y. C. Lu. Smartbean: transparent, concretely efficient, polynomial commitment scheme with logarithmic verification and communication costs that runs on any group. *IACR Cryptol. ePrint Arch.*:785, 2024. URL: https://eprint.iacr.org/2024/785. Withdrawn.

[NS24] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: fast polynomial commitments from lattices. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 243–275. Springer, 2024. URL: https://doi.org/10.1007/978-3-031-68403-6_8.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991. DOI: 10.1007/3-540-46766-1_9. URL: https://doi.org/10.1007/3-540-46766-1_9.

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM J. Comput.*, 9(2):230–250, 1980. DOI: 10.1137/0209022. URL: https://doi.org/10.1137/0209022.

[PT12] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14:1–14:50, 2012. DOI: 10.1145/2220357.2220361. URL: https://doi.org/10.1145/2220357.2220361.

[PT13] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 209–228. SIAM, 2013. DOI: 10.1137/1.9781611973105.16. URL: https://doi.org/10.1137/1.9781611973105.16.

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2024. DOI: 10.1007/978-3-031-58751-1_7. URL: https://doi.org/10.1007/978-3-031-58751-1_7.

[SWY+21] Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. *ACM Trans. Algorithms*, 17(4):31:1–31:19, 2021. DOI: 10.1145/3470566. URL: https://doi.org/10.1145/3470566.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. DOI: 10.1561/3300000030. URL: https://doi.org/10.1561/3300000030.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 926–943. IEEE Computer Society, 2018. DOI: 10.1109/SP.2018.00060. URL: https://doi.org/10.1109/SP.2018.00060.

[XZZ+19] Tiancheng 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 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-26954-8_24.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213. ACM, 1979. DOI: 10.1145/800135.804414. URL: https://doi.org/10.1145/800135.804414.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 138–169. Springer, 2024. DOI: 10.1007/978-3-031-68403-6_5. URL: https://doi.org/10.1007/978-3-031-68403-6_5.