How to Prove Statements Obliviously?

Sanjam Garg* Aarushi Goel† Mingyuan Wang‡

Abstract

Cryptographic applications often require proving statements about hidden secrets satisfying certain circuit relations. Moreover, these proofs must often be generated obliviously, i.e., without knowledge of the secret. This work presents a new technique called — *FRI on hidden values* — for efficiently proving such statements. This technique enables a polynomial commitment scheme for values hidden inside linearly homomorphic primitives, such as linearly homomorphic encryption, linearly homomorphic commitment, group exponentiation, fully homomorphic encryption, etc. Building on this technique, we obtain the following results.

1. An efficient SNARK for proving the honest evaluation of FHE ciphertexts. This allows for an efficiently verifiable private delegation of computation, where the client only needs to perform *logarithmic* many FHE computations to verify the correctness of the computation.
2. An efficient approach for privately delegating the computation of zkSNARKs to a *single untrusted server*, without making any non-black-box use of cryptography. All prior works require multiple servers and the assumption that some subset of the servers are honest.
3. A weighted threshold signature scheme that does not require any setup. In particular, parties may sample their own keys independently, and no distributed key generation (DKG) protocol is needed. Furthermore, the efficiency of our scheme is *completely independent of the weights*.

Prior to this work, there were *no known black-box feasibility results* for *any* of these applications. We also investigate the use of this approach in the context of public proof aggregation. These are only a few representative applications that we explore in this paper. We expect our techniques to be widely applicable in many other scenarios.

*UC Berkeley sanjamg@berkeley.edu
†NTT Research aarushi.goel@ntt-research.com
‡UC Berkeley mingyuan@berkeley.edu

Contents

1 Introduction

Succinct Non-interactive Arguments of Knowledge (SNARKs) [Kil92, Mic94, BCC+17] allow a prover to generate extremely short proofs, certifying the validity of NP statements. Zero-Knowledge SNARKs (or zkSNARKs for short) additionally guarantee that the proof hides the prover's secret witness. Furthermore, these certificates can be verified quite efficiently. Over the last decade, significant research has been dedicated towards improving the asymptotic and concrete efficiency [Gro16, BCG+17, BCG+18, XZZ+19, GWC19, Set20, BCG20, CHM+20, Lee21, KMP20, ZLW+21, BCL22] of zkSNARKs. This effort has resulted in very practical zkSNARK systems with widespread real-world deployment (e.g., see [Gro16, BBHR18, GWC19]). Known zkSNARK constructions are based on a variety of computational and setup assumptions. Additionally, these constructions offer a whole gamut of performance tradeoffs in terms of proof generation cost [BBHR18, GLS+21, Plo21, XZS22, KPV22, CBBZ23], proof size [Gro10, GGPR13, PHGR13a, Gro16, GWC19], and verification time [Gro16, GWC19, CHM+20, KPV22]. These practical constructions have opened a floodgate of new applications (see, for example, [BCG+14, KGC+18, ZFZS20, ZkR21, RPX+22, GHAH+23]).

Proving Statements with Hidden Secrets Efficiently. Some important applications of zkSNARKs are those that involve checking some relationship between cryptographic group elements, cryptographic public keys, etc. For example, given inputs $\{x_i\}$ and group elements $\{\mathbf{h}_i = \mathbf{g}^{y_i}\}$, consider a prover generating a zkSNARK to prove that $\mathbf{a} = \prod_i \mathbf{h}_i^{x_i}$ has been generated correctly. In such cases, the prover may not have direct access to the cryptographic secret; i.e., $\{y_i\}$ values in our example and is therefore left with directly proving a relationship between $\mathbf{a}$ and the $\mathbf{h}_i$ values. This typically involves unrolling these cryptographic operations (in this case, group exponentiation) as a circuit in a non-black-box manner — leading to impractical constructions. We stress that we are interested in designing efficient zkSNARKs for such cases where the cryptographic secrets are not known to the prover.[1] Based on this, we ask:

Can we design efficient SNARKs for proving statements involving hidden values?

1.1 Our Contribution

We present a general approach, namely *FRI on hidden values*, that is specifically tailored to answer the above question. Our work builds upon and extends the renowned Fast Reed-Solomon Interactive (FRI) oracle proof of proximity protocol [BBHR18, BGKS20, BCI+20, BCKL22]. In particular, our technique yields the following main result.

Succinct Polynomial Commitment for Hidden Values. Consider a polynomial $f(x) = f_0 + f_1 \cdot x + \cdots + f_d \cdot x^d$ for some $f_0, \ldots, f_d \in \mathbb{F}$. Let $[\![f_0]\!], \ldots, [\![f_d]\!]$ be some encapsulations[2] of $f_0, \ldots, f_d$, which satisfies linear homomorphism (see Section 4.1 for a formal definition). For a committer, who *only has access to these encapsulations* $[\![f_0]\!], \ldots, [\![f_d]\!]$, we present a succinct polynomial commitment for the polynomial $f(x)$. It is a polynomial commitment scheme in that, given any (public) input x^*, the committer can compute an encapsulation $[\![f(x^*)]\!]$ and a succinct proof certifying that $[\![f(x^*)]\!]$ is the encapsulation of an honest

[1] If these secrets are available to the prover, then various approaches can help provide efficient constructions, e.g., sigma protocols [Sch90], bulletproofs [BBB+18], etc.

[2] For instance, $[\![m]\!]$ could be a linearly homomorphic encryption of m, a linearly homomorphic commitment of m, group exponentiation (i.e., $[\![m]\!] = g^m$), or fully homomorphic encryption of m. We stress that this encapsulation could either be hiding (e.g., encryption/commitment) or not hiding (e.g., group exponentiation).

evaluation[3] of the committed polynomial at $x = x^*$. Notably, this scheme only makes *black-box use* of the underlying encapsulation scheme.

Our polynomial commitment is based on the standard FRI-based polynomial commitment scheme [BGKS20]. In particular, we first show how the FRI protocol can be generalized to work on hidden values and then use it to design a polynomial commitment scheme for hidden polynomials. The efficiency of our scheme is similar to the standard FRI-based polynomial commitment scheme. In particular, the size of the commitment is $O(1)$; the time to generate proofs of openings is $O(d)$; the proof size is $O(\log^2 d)$; and the verification time is $O(\log^2 d)$.[4]

This versatile tool, resulting from a seemingly simple observation, immediately gives rise to a wide range of applications. We explore a few representative applications in this paper. We stress that, prior to our work, there were *no known black-box feasibility results* for all of these applications.

Application I: Efficiently Verifiable Private Delegation of Computation. Consider the problem of *verifiable* private delegation of computation, where a client holds some input x and wants to delegate an expensive (public) computation C on x to a cloud server. Furthermore, it wants the server to append a proof certifying that it did the computation honestly. A standard approach for this task involves sending fully homomorphic encryptions (FHE) [Gen09] $\mathsf{Enc}(x)$ of the client's input x to the server, which performs computations on these ciphertexts and returns the resulting ciphertext $C(\mathsf{Enc}(x))$. In the end, the client can locally decrypt this resulting ciphertext to learn the output $C(x)$. However, while this solution ensures privacy, it offers no integrity regarding whether or not the computation was done honestly. One potentially simple way to add integrity would be to ask the server to provide a short proof (a.k.a., a SNARK proof) attesting to the validity of the computation. However, this inevitably requires making non-black-box uses of the FHE scheme.

Utilizing our technique for FRI on hidden values, we present the first *efficient* solution for this problem. Our scheme incurs *minimal overhead* for the server on top of the necessary work of computing C on the FHE ciphertexts $\mathsf{Enc}(x)$.

At a high level, our idea is to adapt existing polynomial IOP-based SNARK proof systems in order to allow the server to generate a proof certifying that computation C on *encrypted values* was performed honestly, without making non-black-box use of the FHE scheme. For this, we crucially rely on our polynomial commitment scheme that allows the server to commit to polynomials defined by values encrypted inside the FHE ciphertexts. Our succinct proof is *privately verifiable* in that one needs the FHE secret key for verification. This is not an issue since the client does hold the secret key. When applied to the state-of-the-art SNARK proof system (e.g., Fractal [COS20]),[5] we achieve the following efficiency.

- Apart from the necessary computation C on the FHE ciphertext $\mathsf{Enc}(x)$ to obtain $C(\mathsf{Enc}(x))$, the server only needs to perform $O(|C| \cdot \log |C|)$ FHE evaluations (with constant multiplicative depth) for proof generation.
- Similar to any polynomial IOP-based SNARKs instantiated with FRI-based polynomial commitment scheme, the proof size is $O(\log^2 |C|)$ and verification time is $O(\log^2 |C|)$.

To facilitate fast verification (as is the case with all SNARKs with sublinear verification time), a one-time

[3] Note that the prover is able to compute this evaluation because of the linear homomorphism of the encapsulation scheme.

[4] To be precise, the proof size and verification time is $O(\log^2 d \cdot \lambda)$, where λ is the security parameter. Throughout the paper, this λ factor is implicitly assumed in the proof size of all FRI-based SNARKs.

[5] Our approach can be combined with any polynomial IOP-based SNARKs. See Section 6 for more details.

deterministic preprocessing of time $O(|C| \cdot \log |C|)$ is required.[6] We emphasize that this preprocessing is only for the *arithmetization* of the circuit C. In particular, it is *independent* of the FHE scheme. Furthermore, this one-time preprocessing can be reused for an arbitrary number of times (e.g., if the client wishes to delegate many instances of the same computation C to the server).

The only existing (black-box) solutions for this problem [GGP10, AIK10, CKV10, BGV11] require the client to perform a preprocessing step, where it computes C on FHE encryptions of λ-many different random sets of inputs (where λ is the security parameter). Unfortunately, this hugely expensive preprocessing step is inherently *not reusable*. An alternate approach for making this preprocessing resuable, requires the server to perform homomorphic evaluations on *FHE encrypted* FHE ciphertexts of the form $\mathsf{Enc}(\mathsf{Enc}(x))$, which results in a non-black-box use of the FHE scheme. We refer the reader to Section 1.2 for a more detailed comparison with these works.

Application II: Private Outsourcing of zkSNARKs to a Single Server. zkSNARKs offer a substantial advantage to verifiers due to their succinctness and efficient verification process. However, the generation of zkSNARKs can be resource-intensive for the prover. Recent works (zkSaaS [GGJ+23] and EOS [CLMZ23]) have introduced privacy-preserving approaches that leverage cloud computing to alleviate the prover's burden. This is achieved by letting the prover secret share its witness with a number of third-party servers, who then participate in an MPC protocol to compute the zkSNARK. The main drawback of delegating zkSNARK generation to multiple servers in this manner, is that it necessitates trust in at least a subset of the servers — which is highly undesirable.

Alternatively, to avoid this trust assumption, one could use a standard FHE-based approach to delegate the computation of zkSNARKs to a *single untrusted server*. However, this would inevitably result in a non-black-box use of the zkSNARKs. To the best of our knowledge, no black-box solutions are known for the problem of private outsourcing of zkSNARKs to a single untrusted server. We present the first black-box solution for this problem.

At a high level, our idea involves the client sending fully homomorphic encryptions of its secret witness to the server. The server can then perform FHE operations on these ciphertexts to compute FHE ciphertexts corresponding to the extended witness and, subsequently, generate a zkSNARK (borrowing ideas used in our previous application) for this extended witness,[7] without being non-black-box in the FHE and zkSNARK.

We note that, while this approach shares similarities with the previous application we discussed, there is a crucial distinction. In the context of verifiable private delegation of computation, it suffices for the server to produce a zkSNARK at the end that is *privately verifiable*, meaning that only the client, holding the FHE secret key, can verify the correctness of the computation. However, in the current application, since we are delegating computation of a zkSNARK, we require the server's output to be a *publicly verifiable* zkSNARK. To achieve this, we demonstrate that, by replacing FHE with fully homomorphic commitments (FHCom), we can modify the approach from our previous application to make the resulting zkSNARK publicly verifiable. In a nutshell, this involves the client decommitting appropriate parts of the privately verifiable zkSNARK and turning it into a publicly verifiable one.[8]

As before, this approach can be combined with any polynomial IOP-based zkSNARK. When combined

[6] It is typically assumed that this pre-processing is done by a trusted party and the verifier only gets oracle access to it. However, one could also imagine the client himself doing this one-time reusable pre-processing.

[7] The process of computing zkSNARKs typically starts with the computation of an *extended witness*. This extended witness can be seen as a computation trace generated by evaluating the relation circuit with the statement and a (short) secret witness as input.

[8] The client's assistance is inevitable. As we explained in Remark 2, any kind of single-server privacy-preserving SNARKs delegation protocol would necessarily require the client's assistance to make it publicly verifiable.

with Fractal [COS20], we achieve the following efficiency: (1) The server performs $O(|\mathcal{R}| \cdot \log |\mathcal{R}|)$ FHE evaluations for generating a zkSNARK for the NP relation $\mathcal{R}$.[9] (2) the size of the resulting zkSNARK is $O(\log^2 |\mathcal{R}|)$ and verification time is also $O(\log^2 |\mathcal{R}|)$.

Aside from avoiding the need to place trust in any third-party server, another significant advantage of our approach over zkSaaS [GGJ+23] and EOS [CLMZ23] is that the workload and the communication between the server and the client are independent of the size of the relation circuit. Instead, they depend solely on the size of the client's secret witness and the size of the final zkSNARK. In contrast:

- In both zkSaaS and EOS, the extended witness must (1) either be generated by the client (i.e., the person who wants to outsource zkSNARK computation), meaning that the client must first do work proportional to the size of the relation circuit and communicate secret shares of this extended witness to the servers, (2) or it can be generated by the servers using any generic MPC protocol — this, however, requires the total communication amongst the servers to grow with the size of the relation circuit and the number of servers. Furthermore, even aside from generating the extended witness, the total communication required in zkSaaS and EOS still grows with the size of the relation circuit.
- Moreover, in EOS, the client is required to actively participate and communicate with the servers throughout the zkSNARK computation. As a result, in addition to helping with the generation of the extended witness, the total work done by the client in EOS also grows with the size of the relation circuit.

Lastly, it is important to highlight that our goal in this work is to propose the first black-box feasibility result in the single server setting. Unlike zkSaaS and EOS, our current approach is primarily of theoretical significance. It is an exciting future research direction to design concretely efficient protocols in the single server setting.

Application III: Weighted Threshold Signature without Setup. An (n, t)-threshold signature scheme enables distributing the signature signing process among n parties. It guarantees that any subset of $\geqslant t$ parties can generate a valid signature, while any subset of $< t$ parties cannot forge a valid signature. The standard way of constructing threshold signature schemes relies on t-out-of-n linear secret sharing schemes (e.g., Shamir's secret sharing [Sha79]), where the secret signing key is shared among n parties. This standard framework suffers from several drawbacks, as discussed below.

1. *Need for Interactive Setup.* To set up, parties must collectively sample a secret signing key and the corresponding secret shares, which is usually done by a multiparty distributed key generation (DKG) protocol. Despite many years of research [GJKR07, TCZ+20, KMS20, Gro21, DYX+22, GHL22], multiparty DKG protocol still remains a major efficiency bottleneck in practice due to its high communication/computation and/or synchronous communication requirement, especially in the malicious setting. This severely hinders the real-world deployment of threshold signatures for large numbers of parties (e.g., $n > 1000$). Moreover, this expensive interactive setup *needs to be repeated* every time the universe of participating parties or the threshold changes, which is indeed the case for many real-world scenarios (see, e.g., [BGJ+23, GJM+24]).

2. *Limitation on More Expressive Policies.* Due to the use of linear secret sharing schemes, this framework is not well-suited for more expressive policies[10] beyond the (unweighted) threshold access structure. As a

[9] Specifically, the server performs $O(|\mathcal{R}|)$ FHE evaluations (that have the same multiplicative depth as $\mathcal{R}$) to generate the extended witness and then $O(|\mathcal{R}| \cdot \log |\mathcal{R}|)$ additional FHE evaluations with a constant multiplicative depth.

[10] That is, the rule that defines which subsets are authorized. For instance, for (unweighted) threshold access structure, subsets are authorized if and only if its cardinality is $\geqslant t$.

prominent example, for weighted threshold access structures, linear secret sharing schemes are highly inefficient [AN21, GJM+23, BHS23], and constructing more efficient weighted threshold signatures has been the focus of many recent works [MRV+21, CK21, BHS23, GJM+23].

Given these limitations, it is desirable to have a threshold signature scheme where (1) parties could independently sample their own keys, which completely eliminates any need for (interactive) setup, and (2) it naturally supports the weighted access structure *without any efficiency degradation*. We note that SNARKs do provide a generic solution to this. In particular, each party may sample its key pair and sign the message. The aggregator may take the signatures signed by individual parties and generate a succinct proof as the aggregated signature, which certifies that it possesses sufficiently many (measured by the cumulative weights) signatures. This generic solution, however, inevitably makes non-black-box use of the underlying signature schemes. We emphasize that the requirement of black-box use of the underlying signature schemes is not only of theoretical interest but also crucial for concrete efficiency.

Building upon our FRI on hidden values techniques, we give a black-box solution to this problem. For a weighted threshold signature amongst n parties, *irrespective of how large the weight is*, our scheme achieves the following efficiency: the signature size is $O(\log^2 n)$, verification time is $O(\log^2 n)$, and the partial signature aggregation time is $O(n \log n)$. Our solution follows the recently proposed approach [GJM+24, DCX+23] of thresholdizing multisignature schemes. It is highly versatile, as discussed below.

1. One can go beyond the weighted access structure and consider any access structure defined by a circuit C.[11] Our construction can be modified accordingly to realize it. In this case, the aggregation time would be $O(|C| \log |C|)$, and the signature size and verification time would be $O(\log^2 |C|)$.

2. In this work, we use Schnorr signature [Sch90] as an example to demonstrate the utility of our framework (this choice is to highlight the fact that our techniques do not require pairing). In fact, it could be applied to other group-based signature schemes with a linearly aggregatable verification key (e.g., BLS signature [BLS01] resulting in a setup-free threshold signature with non-interactive signing, as we sketched in Section 8.2).

These extensions are discussed in more detail in Section 8.2. Finally, we remark that before this work, the only black-box construction of a weighted threshold signature without setup was proposed by Micali et al. [MRV+21]. However, [MRV+21] only considered a *ramp setting*, where the reconstruction threshold is *much larger* than the privacy threshold. For a *sharp threshold* setting, their aggregated signature size would not be succinct.[12] We refer the readers to Section 1.2 for a more detailed discussion on the related works.

Public Proof Aggregation. Finally, existing works on proving statements involving hidden secrets are mostly motivated in the context of public proof aggregation. To this end, we demonstrate the utility of our techniques in public proof aggregation as well. In particular, we consider the example of aggregating multiple KZG polynomial commitment [KZG10] opening proofs of different polynomials evaluated at different locations. We give an aggregation scheme for a batch of n KZG opening proofs that achieves the following efficiency. The aggregation cost is $O(n \log n)$, the aggregated proof size is $O(\log n^2)$, and the verification cost is $O(n)$. This result is presented in Appendix A.

[11] That is, given an indicator vector $B \in \{0,1\}^n$ for a subset $B \subseteq [n]$, $C(B) = 1$ if and only if B is authorized.
[12] In fact, it would be linear in the threshold.

1.2 Related Works

Proving Statements on Group Elements. Proving statements regarding linear operations on group elements in a pairing-friendly group (a.k.a., inner product argument (IPA)) has been extensively studied [BCC+16, BBB+18, BMM+21, Lee21]. In such an argument, we have two commitments σ_1 and σ_2 committing to two vectors A and B of dimension N. The prover wants to convince the verifier that the inner product $c = \langle A, B \rangle$ is some c. Here, both A and B could either be a vector of group elements or a vector of field elements. The state-of-the-art scheme of [Lee21] enjoys a transparent setup, proof size $O(\log N)$, and verification time $O(\log N)$.

These schemes rely on an innovative folding technique that was initially proposed in [BCC+16]. This folding technique makes crucial use of the homomorphic properties of the commitment schemes. For this particular reason, these works use a pairing-lifted version [AFG+10] of the Pedersen commitment scheme [Ped92] to commit to a vector of group elements. Consequently, these arguments inherently require pairing[13] (even if the statement to prove does not involve pairing, e.g., an inner product between a group vector and a field vector).

In comparison, one could also use our techniques to prove an IPA-type statement. For instance, one could give polynomial commitments to the polynomials encoding these vectors of hidden values and use existing SNARK techniques to generate a proof certifying their inner product. This will give a slightly larger proof size and verification time $O(\log^2 N)$ (indeed, this is what we show in Appendix A). However, our techniques are *significantly more general*, as it applies to any homomorphic encapsulation of values. For example, one could apply it to group elements that do not live in a pairing-friendly group or linear homomorphically encrypted ciphertexts.

Moreover, our scheme gives a *polynomial commitment* for a vector of hidden values. This greatly extends the inner product argument, which can only be used to prove linear operations. As we have shown in this paper, one could utilize our polynomial commitment with existing SNARK proof systems to prove *arbitrary computation* on these hidden values as long as the encapsulation supports the necessary homomorphism.

Verifying Privately Delegated Computation. The problem of verifying privately delegated computation has been studied in prior works. For instance, the works of [FGP14, FNP20, BCFK21] considered designing efficient *non-black-box solutions* for verifying specific FHE schemes. Discussing these non-black-box solutions is beyond the scope of this work. Instead, we focus on the prior *black-box solutions* [GGP10, AIK10, CKV10, BGV11] for this problem.

To verify an arbitrary computation C, the state-of-the-art (black-box) scheme [CKV10] requires a preprocessing step by the client, which does $|C| \cdot \lambda$ FHE operations, where λ is the security parameter. In the online phase, the server also needs to perform $|C| \cdot \lambda$ FHE computation. At a high level, their scheme relies on the idea that the client can hide his actual input $\mathsf{Enc}(x)$ by a random input $\mathsf{Enc}(r)$. The server, which receives $\{\mathsf{Enc}(x), \mathsf{Enc}(r)\}$, cannot distinguish which one is the actual input and, hence, must do the computation correctly (otherwise, the client will notice, with probability 1/2, by checking it against the ciphertext $C(\mathsf{Enc}(r))$ that it computed in the preprocessing phase). For the purpose of verification, the client must preprocess/pre-compute $C(\mathsf{Enc}(r))$, which takes $|C|$ FHE operations. Moreover, to boost soundness, the client must repeat this step λ times. Finally, this preprocessing is *not reusable*. Indeed, after one honest execution, the server learns the information of $C(\mathsf{Enc}(r))$, which makes it insecure to reuse. To make the preprocessing reusable, existing schemes require adding *another layer of FHE*. That is, the

[13] The case where both vectors are field vectors does not require pairing. However, this is not the interesting case we consider in this paper. That is, the prover knows all the witnesses in the clear.

client sends $\mathsf{Enc}(\mathsf{Enc}(r))$ such that, after performing C, the server only learns $\mathsf{Enc}(C(\mathsf{Enc}(r)))$ (but not $C(\mathsf{Enc}(r))$), which makes the preprocessing securely reusable. Besides the inefficiency of doing FHE on FHE ciphertext, this also comes at the cost of turning a black-box construction into a non-black-box one.

In comparison to the existing scheme, by composing our polynomial commitment scheme with the state-of-the-art polynomial IOP-based SNARK, our scheme only requires preprocessing and online prover time $O(|C| \cdot \log |C|)$. Moreover, the preprocessing step is only for the arithmetization of the circuit C, which is (1) independent of FHE, (2) arbitrarily reusable, and (3) deterministic (in particular, this means that the preprocessing is public and can be done by any party). Additionally, note that the client only needs to perform logarithmically many FHE computations in the online phase to verify the proof.

Weighted Threshold Signatures without Setup. Weighted threshold signatures have been studied in many recent works [MRV+21, CK21, BHS23, GJM+23] due to their increasing popularity in proof-of-stake-based blockchain settings. Typically, the DKG-based solutions for this problem suffer from an efficiency degradation linear in the weights, i.e., the efficiency of the signers depends linearly on the weight w associated with them[14]. The overhead in efficiency is incurred due to the size of the secret key, the computation required for producing the partial signature, the aggregation of the partial signatures, etc. Recently, two lines of work construct weighted threshold signatures without incurring this degradation in efficiency, which we discuss next.

The work of Micali et al. [MRV+21] considers the exact same problem that we consider, i.e., a weighted threshold signature without any setup. At a very high level, their construction works as follows. Given sufficiently many valid signatures signed by individual parties, the aggregator uses a Merkle tree to commit to this vector of individual signatures. Now, given a number of random challenges (obtained by the Fiat-Shamir heuristic [FS87]), the aggregator reveals the authenticated root-to-leaf path to prove that it possesses sufficiently many valid signatures from certain parties. The final aggregated signature consists of the root of the Merkle tree and the set of opened root-to-leaf paths. Evidently, the signature size depends on the number of challenges sampled, which, in turn, depends on the ratio between the total weight α of the valid signatures that the aggregator receives and the total weight β of valid signatures that it aims to prove. In particular, [MRV+21] showed that the number of challenges required is proportional to $\frac{1}{\log(\alpha/\beta)}$. In the case where $\alpha \approx \beta$, their signature size would be linear in α. Therefore, [MRV+21] only considers a relaxed *ramp setting*, e.g., $\beta = \alpha/2$. In comparison, our scheme realizes the most stringent sharp threshold setting, where $\alpha = \beta$. In particular, the size of the signature for our scheme is always $O(\log^2 n)$.

Two other concurrent works [GJM+24, DCX+23] propose a SNARK-based approach for designing weighted threshold signatures based on BLS signatures [BLS01]. These works still require a structured reference string (SRS) and require parties to publish (linear-size) additional information besides their public keys. Our construction builds on top of their construction and removes the need for these setups. We discuss their framework next.

Intuitively, these constructions work as follows. Each party will pick its own keys $(\mathsf{sk}_i, \mathsf{pk}_i)$ independently. Now, suppose a subset $B \subseteq [n]$ of the parties have signed the message msg. The aggregator will produce $(\mathsf{apk}, \sigma, \pi)$ as the final signature where (1) $\mathsf{apk} = \prod_{i \in B} \mathsf{pk}_i$ is the aggregation of the public keys from the set B, (2) $\sigma = \prod_{i \in B} \sigma_i$ is the aggregation of the partial signatures from B, and (3) the proof π is a SNARK proof certifying that apk is *an honest aggregation of some parties' public keys with sufficiently high weights*. The verifier accepts the aggregated signature as long as σ verifies under the aggregated public key apk and π is a valid proof for apk. As a prominent feature of this framework, the efficiency of the entire construction could be independent of the weights.

[14] Since there are no known linear secret sharings with weight-independent secret shares.

Now, in this framework, the aggregator does need to prove a statement that involves parties' secret keys hidden inside the public key, which it does not possess. Therefore, one cannot hope to generically use a SNARK in a black-box manner. In order to overcome this barrier, [GJM+24, DCX+23] asks each party to publish suitable *hints*[15] regarding their secret keys, which enables the aggregator to generate a SNARK proof for such statements. In particular, the size of the hints published by each party is *linear in the number of parties*, resulting in a *quadratic overall communication complexity* for communicating the hints from all parties. Similar to DKG-based solutions, this quadratic complexity severely limits the efficiency when the number of parties is large.

Our construction follows the framework of [GJM+24, DCX+23]. In comparison, the techniques we develop in this paper allow the aggregator to directly prove statements about the public keys' aggregation, without any help from individual parties. Therefore, parties do not need to publish any information besides its public key. Furthermore, their construction relies on a trusted setup — a KZG-style SRS [KZG10]; while our scheme does not require any setup. We note that their construction does enjoy a constant size signature due to the use of KZG-style polynomial commitment. In comparison, due to the FRI-style polynomial commitment, our signature size is $O(\log^2 n)$.

Finally, we emphasize that our construction is significantly more general. In particular, their construction *inherently requires pairing*. Consequently, it is not clear if one could extend it to, for instance, the Schnorr signature as the public keys in the Schnorr signature do not necessarily come from a pairing-friendly group. Our construction, on the other hand, could be applied to either Schnorr or BLS regardless of whether the underlying group is pairing-friendly or not.

Additional Related Works. Recently, Bhadauria et al. [BHV+23] proposed a new primitive called *private polynomial commitment schemes*. In a private polynomial commitment scheme, the committer only has access to an encrypted polynomial $f(z) = f_0 + f_1 \cdot z + \cdots + f_d \cdot z^d$. That is, it holds encryptions $[\![f_0]\!], \ldots, [\![f_d]\!]$. The committer wants to send the verifier a succinct (private) polynomial commitment to $[\![f_0]\!], \ldots, [\![f_d]\!]$ such that, given any *private input* x,[16] the committer can give a succinct proof to convince the verifier of $[\![f(x)]\!]$ *without leaking any information of* x. In their paper, they construct this primitive by relying on the inner product argument. In particular, the committer proves $[\![f(x)]\!]$ by proving the inner product between $([\![f_0]\!], \ldots, [\![f_d]\!])$ and $([\![1]\!], [\![x]\!], \ldots, [\![x^d]\!])$. Due to the use of the inner product argument, they could only give a construction using a pairing-based homomorphic encryption scheme. Furthermore, [BHV+23] shows that the private polynomial commitment schemes are useful in multiparty private set intersection (PSI) schemes. Evidently, our polynomial commitment for any linear homomorphic encrypted ciphertexts gives rise to constructions of the private polynomial commitment scheme based on *any* linear homomorphic encryption scheme. It is an interesting question to investigate whether these different instantiations of the private polynomial commitment schemes give any concrete efficiency improvement for their multiparty PSI protocols.

2 Technical Overview

We now discuss the main ideas underlying our results. We start by introducing the notion of *linearly homomorphic encapsulation*, to capture some common properties of linearly homomorphic cryptographic primitives.

[15] In more details, party i holding secret key sk_i needs to send $(g^{\mathsf{sk}_i \cdot \tau}, g^{\mathsf{sk}_i \cdot \tau^2}, \ldots, g^{\mathsf{sk}_i \cdot \tau^n})$ as its hint, where $(g^\tau, g^{\tau^2}, \ldots, g^{\tau^n})$ is the KZG-style SRS.

[16] The committer knows the input x, but the receiver does not.

Linearly Homomorphic Encapsulation. In order to formally capture the idea of hidden values, we introduce a notion of *linearly-homomorphic encapsulations* LHEncap. Informally speaking, such an encapsulation scheme maps elements from a field $\mathbb{F}$ to elements in some output space $\mathcal{S}$, while preserving linearly-homomorphic operations on the original field elements. LHEncap could be a keyed (e.g., encryption) or keyless (e.g., group exponentiation) mapping. Additionally, it could either be randomized (e.g., encryption/commitment) or deterministic (e.g., group exponentiation). As a consequence of these choices, the mapped value in $\mathcal{S}$ may or may not "hide" the input field element — we do not assume any privacy guarantees from LHEncap. We use $[\![m]\!] \in \mathcal{S}$ to denote encapsulation of $m \in \mathbb{F}$.[17] Some common examples of LHEncap are homomorphic encryption, homomorphic commitments, group exponentiations, and fully homomorphic encryptions.

2.1 FRI on Hidden Values

SNARKs for Low-Degree Testing on Hidden Values. Recall that the FRI protocol [BBHR18] is an interactive oracle proof of proximity (IOPP) that can be transformed into a SNARK for the following problem: given a commitment to evaluations of a function (on some domain), the prover wants to convince the verifier that this set of evaluations is "very-close" (where this proximity is usually measured by relative Hamming distance) to a codeword of an error-correcting code.

We consider a variant of this problem on hidden values, where the prover proves proximity of *encapsulated* evaluations, to a codeword of an error-correcting code — without knowledge of the encapsulated values. We refer to the SNARKs for this problem as *SNARKs for low-degree testing on LHEncap*. While one can consider this problem with respect to any error-correcting code, in this work, we only focus on Reed-Solomon codes. Our key observation is that the FRI protocol can be modified to first obtain an IOPP for encapsulated values — we refer to this modified protocol as FRI on LHEncap. Next, this protocol can be transformed into a SNARK for low-degree testing on LHEncap in the random oracle model, using the [BCS16] compiler and the Fiat-Shamir heuristics [FS87].

Overview of FRI. We start by giving an overview of the original FRI protocol [BBHR18]. A naïve approach for checking whether a set of $|D|$ values (where $|D| > d$) correspond to evaluations of a degree d polynomial is to try and interpolate evaluations on any $d+1$ randomly chosen points and check if they lie on a degree d polynomial. Clearly, this requires the verifier to run in time linear in d. FRI gives a way to reduce the verifier run time to be logarithmic in d. FRI is a logarithmic-round, folding-argument-based IOPP, inspired by observations from Fast Fourier Transform (FFT). At a high level, FRI performs a random reduction in each round, at least halving the instance size via a linear folding procedure. In more detail, FRI is based on the following two observations:

- *Obs 1:* Any degree-d polynomial $f(x) = \sum_{i=0}^{d} c_i \cdot x^i$ can be written as the following combination of two degree $d/2$ polynomials f_E, f_O:

  $$f(x) = f_E(x^2) + x \cdot f_O(x^2),$$

  where $f_E(x) = c_0 + c_2 \cdot x + c_4 \cdot x^2 + \cdots$ is defined by the even coefficients of f and $f_O(x) = c_1 + c_3 \cdot x + c_5 \cdot x^2 + \cdots$ by the odd coefficients.

[17] We remark that we slightly abuse notation here and use $[\![m]\!]$ to denote an encapsulation of m, even when the encapsulation scheme is randomized. In particular, for simplicity of notation, we do not make the randomness explicit in this representation. Looking ahead, whenever relevant, the use of randomness will be made clear in the accompanying text.

- *Obs 2:* The second observation used in the FRI protocol is that $f_E(x^2)$ and $f_O(x^2)$ can be re-written as

$$f_E(x^2) = \frac{f(x) + f(-x)}{2}, \qquad f_O(x^2) = \frac{f(x) - f(-x)}{2x}.$$

  Moreover, when x is a d^{th} root of unity (say ω^i), then $-x = -\omega^i = \omega^{d/2+i}$.

Given these observations, the FRI protocol proceeds in two phases, namely the *commit phase* and the *query phase* as follows:

Commit Phase: The commit phase consists of $O(\log d)$ rounds, in which the prover sends domain evaluation oracles to the verifier, who then responds with a random challenge. This phase of FRI performs the random reduction, where the instance size gets halved with each step.

At the beginning, the verifier is given oracle access to evaluations of the polynomial f on domain $D = \{\omega^0, \ldots, \omega^{|D|-1}\}$, which is the set of $|D|^{\text{th}}$ roots of unity. This is referred to as the *domain evaluation oracle*. In the first round, given a random challenge (say α_0) from the verifier, the prover sends another domain evaluation oracle (on the new domain $D_1 = \{\omega^0, \omega^2, \omega^4, \ldots, \omega^{|D|-1}\}$) of the folded polynomial $f_1(x) = f_E(x) + \alpha_0 \cdot f_O(x)$. Note that the degree of the folded polynomial f_1 is at most $d/2$ and $|D_1| = |D|/2$. A similar randomized reduction is then performed on this new polynomial f_1 in the next round and so on. This continues until we are left with a constant polynomial at the end of $(\log_2 d)^{\text{th}}$ step, whose degree can be easily verified in constant time.

Note that, since the prover only sends domain evaluation oracles to the verifier in every round, it does not necessarily need to compute evaluations of $f_E(x), f_O(x)$ from scratch. This is where FRI leverages *Obs 2* from above. Evaluations of $f_E(x), f_O(x)$ on domain D_1, can be computed using simple *linear combinations* of evaluations of $f(x)$ on domain D, which it already knows (as described in *Obs 2*). The same observation is used in each of the $\log_2 d$ steps. Thus, the domain evaluation oracle in each round i can be computed in $O(|D_{i-1}|)$ time.

Query Phase: At the end of the commit phase, the verifier queries each of the $\log_2 d$ oracles at random points within their respective domains of definition. These openings are subsequently utilized to verify the consistency of the folding (done by the prover) in each reduction step from the commit phase.

Overview of FRI on LHEncap. Towards adapting the above protocol to work on "hidden" values, we make the following simple observations:

1. Given the evaluations of the original instance f on domain D, throughout the protocol, the prover only performs *linear operations* on these evaluations.[18]

2. The prover does not need the description of f in any other representation besides these domain evaluations.

3. Given oracle access to these domain evaluations, the verifier also only performs *linear operations* on the queried evaluations to check consistency.

These observations lead us to conclude that rather than possessing them in the clear, it suffices for the prover to have access to the domain evaluations encapsulated inside an LHEncap (and similarly, it suffices for the verifier to have oracle access to these encapsulations and not the actual evaluations). Since

[18] If instead, the prover starts with domain evaluations of f on domain $\mathbb{H} \subset \mathbb{F}$, which is a multiplicative subgroup of size $d+1$, they can use FFT to compute evaluations on the larger domain D. Since FFT is a linear operation, this translation also requires the prover to only perform linear operations on the evaluations on $\mathbb{H}$.

LHEncap (see Definition 3) preserves linear operations of encapsulated values, given *encapsulated domain evaluations*, it is possible to emulate the FRI protocol almost as is, albeit on encapsulations. We refer to this variant as FRI on LHEncap.

Public/private-Verifiability. We remark that, if we want the above protocol to be publically verifiable, then we crucially require the underlying LHEncap scheme to be linearly-homomorphic w.r.t. randomness. This is because, in the query phase, the verifier performs two kinds of checks: (1) if the random folding is done correctly (e.g., $a \cdot [\![m_1]\!] + b \cdot [\![m_2]\!] = [\![m_3]\!]$) and (2) if the final reduced polynomial is a constant polynomial (e.g., $[\![m_1]\!] = [\![m_2]\!]$). This essentially checks if certain linear constraints are satisfied on the encapsulated values. For instance, checking if the final polynomial is a constant polynomial requires checking that the encapsulations of all evaluations of this polynomial on the final reduced domain are the same. Note that, while each of those encapsulations was computed using the same linear function over the original encapsulations of the domain evaluations of f (on $\mathbb{H}$), the operations within the linear function may not have been computed in the same order. Consider the following simple example: given 2 encapsulations $[\![m_1]\!]$ and $[\![m_2]\!]$, the encapsulations of $a \cdot m_1 + a \cdot m_2$ resulting from $(a \cdot [\![m_1]\!] + a \cdot [\![m_2]\!])$ and $(a \cdot ([\![m_1]\!] + [\![m_2]\!]))$ are only *guaranteed* to be the same when the underlying LHEncap scheme satisfies linear homomorphism w.r.t. to randomness.

As a special case, if the underlying LHEncap is deterministic, the above construction is always publicly verifiable. As a notable example, group exponentiation (i.e., $m \mapsto g^m$) is a deterministic encapsulation and our FRI on LHEncap will henceforth be publicly verifiable in this case.

Finally, observe that if the LHEncap is decryptable and the verifier knows the key corresponding to the underlying LHEncap scheme, one can always decrypt the encapsulations of all evaluations and check if the linear constraints are satisfied. That is, instead of checking $a \cdot [\![m_1]\!] + b \cdot [\![m_2]\!] = [\![m_3]\!]$, the verifier first decrypts and then checks $a \cdot m_1 + b \cdot m_2 = m_3$. But, of course, this only gives us a *privately verifiable* version of the above protocol.

Remark 1 (Necessity of FFT-friendly Field.)**.** *The original FRI protocol (which is what we present above) requires an FFT-friendly field (a.k.a.,* smooth field *[BCKL23, BCKL22]). That is, it is required that the multiplicative group $\mathbb{F}^*$ contains a large enough subgroup of order 2^ℓ such that $2^\ell \geqslant d$. However, not all prime fields $\mathbb{F}_p$ satisfy this property. In particular, a prime field $\mathbb{F}_p$ has this property only if 2^ℓ divides $p-1$. This seems to limit the applicability of our scheme as it only applies to encapsulated values from an FFT-friendly field.*

Fortunately, a recent line of works [BCKL23, BCKL22] have addressed this issue and extended FRI to all finite fields. Crucially, the extended FRI protocol still only requires the prover and the verifier to perform linear operations. Consequently, one can still apply the extended protocols to the (encapsulated) values analogously. We refer the readers to [BCKL22, Appendix B] for a detailed description of the extended FRI protocol for arbitrary fields. For the sake of simplicity, throughout this paper, we present our results for FFT-friendly fields. But they all extend to arbitrary fields using the techniques from [BCKL23, BCKL22].

2.2 Polynomial Commitments on Hidden Values

Next, we formalize a variant of polynomial commitments, referred to as *polynomial commitments on LHEncap*. We start by recalling polynomial commitments.

Polynomial Commitments. A polynomial commitment scheme enables the committer C to generate a succinct commitment to a polynomial. Subsequently, the committer can disclose the evaluation of this polynomial at any (public) location and provide a convincing proof to the receiver R that the disclosed value is consistent with the original commitment. [KZG10, BGKS20, Lee21] are some common examples of polynomial commitment schemes.

Polynomial Commitments on LHEncap. We extend the above notion to polynomial commitments for hidden values. Given access to a set of encapsulated evaluations of a polynomial, a polynomial commitment for hidden values allows the committer C to create a short commitment, such that it can later reveal the *encapsulations of evaluations* of the polynomial at any locations, while being able to convince the receiver R that the opening is consistent with the committed encapsulated values.

Polynomial Commitments on LHEncap from FRI on LHEncap. We then present a construction of a polynomial commitment scheme on LHEncap using our FRI on LHEncap scheme. [BGKS20] showed how the FRI protocol can be used to construct a simple polynomial commitment scheme, where the size of opening proofs and the time required to verify these opening proofs are both polylogarithmic in the size of the instance. We observe that the same ideas can be generalized to obtain a polynomial commitment on LHEncap.

Overview of FRI Based Polynomial Commitments [BGKS20]. The FRI-based polynomial commitment scheme works as follows. Given polynomial evaluations $y_1, \ldots, y_{d+1}$, the committer C starts by using FFT on these evaluations to obtain evaluations on a larger domain D. It then computes and sends a Merkle Hash of these evaluations to the receiver. When asked to give an opening proof for some pair $(x^*, y^*) \in \mathbb{F}^2$, where $y^* = f(x^*)$, the committer uses the fact that there exists a quotient polynomial $h(x)$ of degree at most $d-1$, such that

$$f(x) - y^* = h(x) \cdot (x - x^*)$$

to convince the receiver R that $h(x) = \frac{f(x)-y^*}{x-x^*}$ is of degree at most $d-1$. This proof is given using the FRI protocol. Note that, in the FRI protocol (for h), the verifier needs to have oracle access to the evaluation of $h(x)$. This is not a problem since the verifier does have access to f (which was committed using the Merkle Hash and the response to queries made to f on the specified domain can be answered by giving out the corresponding Merkle proof), and any query to h can be equivalently treated as one query to f. For example, if the verifier wants to query $h(x')$, it may query f at $x = x'$ and compute $h(x')$ as $h(x') = \frac{f(x')-y^*}{x'-x^*}$.

Our Observation. We observe that if instead the committer starts with encapsulations of evaluations of a polynomial, then the above ideas can be used to obtain a polynomial commitment on LHEncap, if we use our FRI on LHEncap protocol from the previous section. This protocol proceeds exactly as the one described above, except that the prover now has encapsulated evaluations (on some domain D), which it will commit to using a Merkle Hash. For the opening proof, it will make use of FRI on LHEncap to prove that $[\![h(x)]\!] = \frac{[\![f(x)]\!]-[\![y^*]\!]}{x-x^*}$ is a set of encapsulated evaluations of a polynomial of degree at most $d-1$. (Observe that computing $[\![h(x)]\!]$ is again a linear operation. For any $x' \in D$, computing $[\![h(x')]\!]$ as $\frac{[\![f(x)]\!]-[\![y^*]\!]}{x'-x^*}$ is linear because both x' and x^* are in the clear.)

This finishes the overview of our polynomial commitment scheme for hidden values. In the subsequent sections, we present an overview of how this tool enables a wide range of applications.

2.3 Application I: Efficiently Verifiable Private Delegation of Computation

We present a solution for *efficiently verifiable* FHE-based private delegation of computation, which does not require any non-black-box use of the FHE scheme. At a high level, we show how our polynomial commitment scheme on LHEncap can be combined with state-of-the-art polynomial IOP-based SNARKs (e.g., [COS20, GWC19]) to obtain this solution.

By FHE-based delegation, we are referring to the setting where the client encrypts its input (on which the computation needs to be performed) using FHE and sends the corresponding ciphertexts to the server.

The server then carries out computations on these encrypted data and returns the resulting encrypted output. Ultimately, the client can decrypt the output locally to learn the desired information. In order to add integrity to this delegation and to verify that the computation was done honestly, our high-level idea is to have the service provider attach a succinct proof for this computation. For this, we want to leverage the techniques developed in the previous sections.

Before describing our approach, let us briefly recall the structure of existing state-of-the-art SNARKs such as Fractal [COS20] and Plonk [GWC19]. Most existing efficient constructions of SNARKS start by designing a *polynomial-IOP* and then transforming that into a non-interactive proof with the help of polynomial commitment schemes (and Fiat-Shamir heuristics). A polynomial IOP is a variant of IOP, where the oracles sent by the prover to the verifier are essentially polynomials. In other words, in each round of a polynomial-IOP, the prover provides the verifier with an oracle access to a polynomial function which the verifier can query at any location of its choice.

Going back to our application, the relation we are interested in proving is that the FHE evaluation was done correctly. In other words, the output ciphertext is an encryption of the result of computing some function (say C) on the input. This is essentially similar to proving the following relation on values encapsulated inside FHE ciphertexts: $\mathcal{R} : \mathsf{output} = C(\mathsf{input})$ (as opposed to $\mathcal{R}' : \mathsf{Enc}(\mathsf{output}) = C(\mathsf{Enc}(\mathsf{input}))$). However, we want to do this without making non-black-box use of the underlying FHE scheme.

We leverage the structure of polynomial-IOP and the homomorphism of the FHE scheme as follows. Given the relation circuit in the clear and FHE ciphertexts for the extended-witness (or the computation-trace of $C(\mathsf{input})$), the prover can compute the same operations that it needs to perform in a regular polynomial-IOP, except that the computations involving the extended witness will have to be performed as FHE evaluations. As a result of this, the resulting polynomial oracles that the prover needs to send to the verifier in IOP are also hidden inside the FHE ciphertexts. This is exactly the type of application for which our polynomial commitment on LHEncap is most useful. In order to transform this IOP, where the polynomial oracles are encapsulated into an interactive/non-interactive proof system, the server can use this polynomial commitment scheme to emulate sending an oracle and then answering oracle queries.

Using the above transformation, the server is able to generate a SNARK proof encapsulated inside an FHE ciphertext. The client can now decrypt this proof and check if it passes verification to verify the correctness of the delegated computation. In Section 6, we show how this idea can be used with any polynomial IOP-based SNARK. In particular, when instantiating this approach with the underlying polynomial IOP that Fractal [COS20] is based on, we show that in addition to performing $O(|C|)$ FHE evaluations to compute the output, the server only needs to perform $O(|C| \log |C|)$ FHE evaluations with a constant multiplicative depth.

2.4 Application II: Private Outsourcing of zkSNARKs to a Single Server

We now present our second application — private delegation of zkSNARK computation to a single untrusted server. As mentioned, while succinct and quick to verify, proof generation in existing zkSNARKs is a computation and memory-intensive task. Thus, it would be beneficial to leverage the power of cloud computing and enable the prover to delegate this task to an external server. However, to ensure the privacy of the secret witness employed in the generation of zkSNARK, it is highly desirable or, in many cases, necessary that this delegation remains privacy-preserving. We demonstrate how this can be accomplished in a black-box manner by modifying our protocol from the previous section.

As discussed in the introduction, there is one crucial barrier in directly using the ideas from the previous section to obtain a solution for this application. Checking the validity of the SNARK generated by the server in the previous application required knowledge of the secret key associated with the FHE scheme for

decryption, and hence, that proof was only *privately verifiable*. However, in our current application, since the computation being delegated to the server is the generation of a zkSNARK, the resulting zkSNARK must, therefore, be *publicly verifiable*. To this end, our idea is to use fully homomorphic commitments (FHCom) instead of FHE. FHCom is an FHE scheme where the ciphertexts are statistically binding, and given knowledge of the secret key, it is possible to generate a decommitment to any ciphertext without leaking the secret key. For instance, Gorbunov, Vaikuntanathan, and Wichs [GVW15] showed that the LWE-based Gentry-Sahai-Waters FHE scheme [GSW13] is an FHCom.

Given an FHCom, our main idea is the following. The client starts by encrypting the witness using FHCom. It sends these ciphertexts to the server. The server uses FHCom evaluation on these ciphertexts to obtain encryptions of the extended witness. Given encryptions of the extended witness, it then uses additional FHCom evaluations and our polynomial commitment on LHEncap to generate a zkSNARK exactly as described in the previous section. It then sends the resulting encrypted zkSNARK back to the client. Now, the client will decrypt/decommit all the FHCom contained in this encrypted SNARK. By doing so, any party can perform the required verification step on these decommitted values in the clear, rendering it publicly verifiable. The encrypted zkSNARK sent by the server, along with these decommitments, jointly constitute a publicly verifiable zkSNARK.

2.5 Application III: Weighted Threshold Signature without Setup

Finally, we present an overview of our construction for weighted threshold signatures without setup. Our starting point is a multisignature scheme[19] [Bol03, BDN18]. Unlike threshold signature schemes, multisignature scheme enjoys the great advantage that parties' secret keys are not correlated. Therefore, parties may sample their key pairs independently, and no interactive/trusted setup is required. Similar to [GJM+24, DCX+23], our high-level idea is to first turn a multisignature scheme into a trivial threshold signature scheme where the verification key and aggregated signature are *not succinct*. Afterward, we will use the techniques developed in this paper to *compile* this scheme into another one that does have a succinct verification key and signature.

Let us use Schnorr multisignatures [MPSW19, NRS21] as an example. Suppose we have n parties and the weights and threshold are $(w_1, w_2, \ldots, w_n)$ and T, respectively. Each party independently samples their key pair $\{\mathsf{sk}_i, \mathsf{pk}_i = g^{\mathsf{sk}_i}\}$ (here, g is the generator of some cryptographic group). In the trivial scheme, the (linear-size) verification key is $\mathsf{vk} = (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n)$. Now, on some message msg, suppose a subset of parties $B \subseteq [n]$ engage in a signing session and generate a multisignature σ. Then, the aggregator will simply output (B, σ) as the threshold signature. Here, we abuse notation and treat B as the indicator vector for the set $B \subseteq [n]$. That is, $B = (b_1, \ldots, b_n) \in \{0,1\}^n$ such that $b_i = 1$ if and only if $i \in B$. Now, given the signature (B, σ), the verifier will do the following: (1) it checks that the cumulative weight is sufficiently high, i.e., $\sum_i b_i \cdot w_i \geqslant T$; (2) it computes the aggregated public key $\mathsf{apk} = \prod_{i \in B} \mathsf{pk}_i$;[20] (3) it verifies σ under apk. It is not hard to see that this trivial threshold scheme is secure as long as the underlying multisignature scheme is secure.[21] However, the signature size, the verification key size, and the verification time all grow

[19] A multisignature scheme between n parties allows each party to sample its own key pair and sign messages independently, after which, a *succinct* aggregated signature can be generated to convince the verifier that all n parties have signed the message. Intuitively, a multisignature scheme is an n-out-of-n threshold signature scheme.

[20] In order to prevent the *rogue key attack* [BDN18], the aggregation of the public keys typically uses a random linear combination $\mathsf{apk} = \prod_i (\mathsf{pk}_i)^{\alpha_i}$ given by the random oracle $\alpha_i = \mathsf{RO}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{pk}_i)$. This can be equivalently treated as each public key pk_i is reset to be $(\mathsf{pk}_i)^{\alpha_i}$. For simplicity of presentation, we omit this detail.

[21] Typically, in a multisignature scheme, we require all n parties to sign the message. In these cases, the aggregated public key apk can be *precomputed* and, hence, the verification is indeed succinct. In our setting, we do not require all parties to sign and will compute apk *on the fly*. Most known multisignature schemes [BDN18, MPSW19, NRS21] support such a modification, although

linearly in n.

We now describe how our techniques can enable succinct verification. Note that, the only part in the verification process that is not succinct is to compute the aggregation of the public key $\mathsf{apk} = \prod_{i \in B} \mathsf{pk}_i$ and verify that it has sufficient weight. Our idea is to let the verifier ask the aggregator to compute this for her. However, to ensure security, the verifier must verify that the aggregated public key is indeed an honest aggregation of parties' public keys with that satisfy the total weight requirement. This leaves us with the following question:

How can the aggregator generate a succinct proof for the honest aggregation of the public key?

If the secret keys are known to the aggregator, then this problem has a trivial solution — applying any generic SNARK would suffice. However, the aggregator does not possess sk_i's in the clear, but only its encapsulation $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. This is where our FRI on hidden values can be of help. In particular, it can be used to enable the aggregator to commit to polynomials described by sk_i, even though the aggregator only possesses pk_i. Now, this observation, together with other known techniques from the SNARK literature, gives rise to our construction, which we summarize next.

The common tool that is used in the SNARKs literature is *generalized sumcheck* (Lemma 1), which is a polynomial identity test for proving that the inner product $\langle X, Y \rangle$ between two vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$, encoded as polynomials, is equal to some specified value. We refer the readers to the technical sections for this lemma and proceed to describe our scheme. We change the verification key vk to be the polynomial commitments of two polynomials: (1) an encoding of the secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and (2) an encoding of the weights $(w_1, \ldots, w_n)$.[22]

To generate a succinct proof π for apk, the aggregator needs to prove that there is some subset $B \subseteq [n]$ such that (1) the inner product between B and $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ is $\mathsf{dLog}(\mathsf{apk})$, i.e., $\mathsf{apk} = g^{\langle B, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \rangle}$, and (2) the inner product between B and $(w_1, \ldots, w_n)$ is sufficiently high $\geqslant T$. Therefore, the aggregator will first commit to some polynomial that encodes a set B. Next, it will generate inner product proofs using the generalized sumcheck. Finally, the aggregator must also prove that B is a binary vector.[23] Otherwise, the aggregator may use $B = (T, 0, \ldots, 0)$ to prove that apk contains sufficient weight even though the adversary can sign under apk as long as the first party signs. With these suitable changes, the aggregated signature will consist of $(\mathsf{apk}, \sigma, \pi)$, and the verifier can succinctly verify. Again, we remind the reader that the efficiency of the entire scheme is independent of the weights.

This summarizes the high-level structure of the construction. In addition to being setup-free and naturally supporting weights, our scheme also enjoys many other advantages. One notable feature is that, unlike regular threshold signature schemes, where there is a *predefined threshold*, each aggregated signature in our scheme comes with a *signature-specific threshold* (refer to Remark 3). Intuitively, this allows the aggregator to generate more fine-grained proofs (i.e., the aggregated signature), certifying exactly how many parties sign a given message. We leave these details to the technical sections.

3 Preliminaries

In this section, we recall the definitions of some well-known primitives.

---

the resulting verification is no longer succinct.

[22] Note that this is a deterministic process that any party can compute given $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and $(w_1, \ldots, w_n)$.

[23] Which can be checked as $B(x) \cdot (1 - B(x))$ should be 0 on the information locations.

Definition 1 (Reed-Solomon Codes). *For a subset of some field $D \subseteq \mathbb{F}$, and a rate parameter $\rho \in (0,1]$, we denote by* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *the set of all functions $f : D \to \mathbb{F}$ for which there exists $\hat{f} \in \mathbb{F}^{<\rho|D|}[x]$ agreeing with f on D.*

Definition 2 (SNARKs). *Let $\mathcal{R}$ be an NP relation. A succinct non-interactive argument of knowledge (SNARK) for $\mathcal{R}$ is defined by a triple of algorithms* $(\mathsf{SNARK.Setup}, \mathsf{SNARK.Prover}, \mathsf{SNARK.Verify})$ *with the following syntax:*

1. $\mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda)$*: On input the security parameter λ, the setup algorithm outputs a common reference string* crs*.*
2. $\pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, \mathsf{st}, \mathsf{wit})$*: On input the common reference string* crs*, statement* st *and witness* wit*, the prover algorithm outputs a proof π.*
3. $b \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{st}, \pi)$*: On input the common reference string* crs*, the statement* st*, and proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.*

These algorithms satisfy the following properties:

- ***Completeness:*** *For any statement* st*, and witness* wit*, such that $\mathcal{R}(\mathsf{st}, \mathsf{wit}) = 1$ it holds that*

$$\Pr\left[1 \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{st}, \pi) \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, \mathsf{st}, \mathsf{wit}) \end{array}\right] \geqslant 1 - \mathsf{negl}(\lambda).$$

- ***Knowledge Soundness:*** *There exists an extractor $\mathcal{E}$ and knowledge error $\varepsilon : [0,1] \to [0,1]$, such that, for any PPT algorithm* P^**, the following holds.*

$$\Pr\left[\mathcal{R}(\mathsf{st}, \mathsf{wit}) = 0 \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda) \\ (\mathsf{st}, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}) \\ 1 \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{st}, \pi) \\ \mathsf{wit} \leftarrow \mathcal{E}_{P^*}(\mathsf{crs}) \end{array}\right] \leqslant \varepsilon(\theta).$$

- ***Succinctness:*** *The proof size should be sublinear in the witness size, i.e., $|\pi| \in o(|\mathcal{R}|)$.*

Additionally, we say that the SNARK has ***zero-knowledge*** *if it does not leak any information besides the truth of the statement. In more detail, let* $\mathsf{SNARK.Setup}(1^\lambda)$ *be such that it outputs* crs *along with a simulation trapdoor τ. Then there exists a simulator* Sim*, such that for all statement-witness pairs* st*,* wit*, where $\mathcal{R}(\mathsf{st}, \mathsf{wit}) = 1$ and for all PPT adversaries $\mathcal{A}$, the following holds:*

$$\left|\Pr\left[\mathcal{A}(\mathsf{crs}, \mathsf{st}, \pi) = 1 \mid (\mathsf{crs}, \tau) \leftarrow \mathsf{SNARK.Setup}(1^\lambda); \pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, \mathsf{st}, \mathsf{wit})\right]\right.$$
$$\left. - \Pr\left[\mathcal{A}(\mathsf{crs}, \mathsf{st}, \pi) = 1 \mid (\mathsf{crs}, \tau) \leftarrow \mathsf{SNARK.Setup}(1^\lambda); \pi \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{st}, \tau)\right]\right| = \mathsf{negl}(\lambda)$$

4 FRI on Hidden Values

In this section, we present our main result, i.e., a modification to the FRI [BBHR18] protocol that enables operations on "hidden values". To formally capture the idea of hidden values, we start by introducing the notion of linearly-homomorphic encapsulations in Section 4.1. In Section 4.2, we define SNARKs for low-degree testing on hidden values, and finally, in Section 4.3, we present the construction of FRI on hidden values.

4.1 Linearly-Homomorphic Encapsulation

We first formally define LHEncap, and then proceed to give examples of known cryptographic/non-cryptographic primitives that satisfy the properties of LHEncap. For the applications that we consider in this work, it suffices for us to view LHEncap as a secret-key primitive. A public-key variant can be defined analogously.

Definition 3 (Linearly-Homomorphic Encapsulation)**.** *A* linearly-homomorphic encapsulation *(LHEncap) with key-space $\mathcal{K}$, message-space $\mathbb{F}$, randomness-space $\mathcal{R}$,*[24] *and output-space $\mathcal{S}$ consists of a tuple of PPT algorithms* (Setup, KeyGen, Encap, Eval) *defined as follows:*

1. $\mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$*: The setup algorithm takes as input the security parameter λ and outputs some public parameters* pp*.*
2. $\mathsf{k} \leftarrow \mathsf{KeyGen}\left(\mathsf{pp}\right)$*: The key generation algorithm takes as input the public parameters* pp *and outputs a key* $\mathsf{k} \in \mathcal{K}$*.*
3. $[\![m]\!] \leftarrow \mathsf{Encap}\left(\mathsf{pp}, \mathsf{k}, m; r\right)$*: The encapsulation algorithm takes as input the public parameters* pp*, the key* $\mathsf{k} \in \mathcal{K}$*, a message $m \in \mathbb{F}$ and a random value $r \in \mathcal{R}$, and outputs an encapsulation $[\![m]\!] \in \mathcal{S}$.*
4. $[\![m]\!] \leftarrow \mathsf{Eval}\left(\mathsf{pp}, [\![m_1]\!], \ldots, [\![m_n]\!], f\right)$*: The evaluation algorithm takes as input the public parameters* pp*, a set of encapsulations $[\![m_1]\!], \ldots, [\![m_n]\!] \in \mathcal{S}^n$ (where $n \in \mathbb{N}$), a linear function f and outputs a new encapsulation $[\![m]\!] \in \mathcal{S}$.*

These algorithms (KeyGen, Encap, Eval) *must crucially satisfy* linear-homomorphism*. Additionally, an LHEncap must either be* **linearly-homomorphic w.r.t. randomness** *or be* **decryptable**. *These properties are defined as follows.*

- ***Linear-Homomorphism:*** *For every $n \in \mathbb{N}$, any $m_1, \ldots, m_n \in \mathbb{F}^n$, any $r_1, \ldots, r_n \in \mathcal{R}^n$ and any linear functions f, it holds that:*

$$\Pr\left[\begin{array}{c|c} \exists r \in \mathcal{R} \text{ s.t. } \mathsf{Eval}\left(\mathsf{pp}, \{[\![m_i]\!]\}_{i\in[n]}, f\right) & \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right) \\ \quad = \mathsf{Encap}\left(\mathsf{pp}, \mathsf{k}, f(\{m_i\}_{i\in[n]}); r\right) & \mathsf{k} \leftarrow \mathsf{KeyGen}\left(\mathsf{pp}\right) \\ & \forall i \in [n], m_i \leftarrow \mathsf{Encap}\left(\mathsf{pp}, \mathsf{k}, m_i; r_i\right) \end{array}\right] = 1.$$

- *We say that LHEncap is* **linearly-homomorphic w.r.t. randomness** *if, in the above equation, we have $r = f\left(\{r_i\}_{i\in[n]}\right)$. We note that this is only possible when the randomness-space $\mathcal{R} = \mathbb{F}$. Moreover, any* deterministic *LHEncap trivially satisfies linearly-homomorphic w.r.t. randomness.*
- ***Decryptable:*** *We say that LHEncap is decryptable, if there exists a polynomial-time deterministic algorithm* $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{k}, [\![m]\!])$*, that takes the public parameters* pp*, the key* $\mathsf{k} \in \mathcal{K}$*, and an encapsulation $[\![m]\!] \in \mathcal{S}$ as input, and outputs a message $m^* \in \mathbb{F}$, such that the following holds for every $m \in \mathbb{F}$, $r \in \mathcal{R}$:*

$$\Pr\left[m^* = m \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right),\ \mathsf{k} \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ [\![m]\!] \leftarrow \mathsf{Encap}(\mathsf{pp}, \mathsf{k}, m; r) \\ m^* = \mathsf{Decrypt}(\mathsf{pp}, \mathsf{k}, [\![m]\!]) \end{array}\right] = 1.$$

[24] If the LHEncap scheme is keyless and/or deterministic, then $\mathcal{K} = \emptyset$ and/or $\mathcal{R} = \emptyset$, respectively.

Notation. As mentioned earlier, throughout the rest of this paper, we will use $[\![m]\!]$ as a shorthand notation to denote an encapsulation of m w.r.t. some LHEncap scheme. Unless necessary, we neither explicitly specify the randomness (if any) used in this encapsulation, nor do we mention the algorithm (KeyGen, Encap, Eval) associated with the LHEncap scheme. Moreover, for simplicity of notation, instead of using the Eval function, we will use $[\![m_1]\!] + [\![m_2]\!]$ to denote addition (i.e., where the function f in $\mathsf{Eval}([\![m_1]\!], [\![m_2]\!], f)$ is the addition function). Similarly, we use $c \cdot [\![m_1]\!]$ to denote the multiplication by a constant c.

Examples. Following are some common examples of LHEncap:

- *Homomorphic Encryption:* Linearly (for instance, [ElG84, Pai99, Reg05]) or fully-homomorphic encryption [Gen09] are the most common examples of a *keyed, randomized, and decryptable* LHEncap. It is easy to see that the encapsulation algorithm in such schemes is the encryption algorithm, and the output space $\mathcal{S}$ is the ciphertext space. Linearly homomorphic encryption such ElGamal [ElG84] is also *linearly-homomorphic w.r.t. randomness*.

- *Homomorphic Commitments:* Linearly-homomorphic commitments such as Pedersen commitments [Ped92] and fully-homomorphic commitments such as the Gentry-Sahai-Waters FHE scheme [GSW13, GVW15] are another example of LHEncap. These are *keyless/keyed and randomized* encapsulations. The output space $\mathcal{S}$ is the commitment space. Pedersen commitments are also *linearly-homomorphic w.r.t. randomness*.

- *Group Exponentiation:* Mappings such as $m \mapsto g^m$ (where g is the generator of a group $\mathbb{G}$ that has the same order as $\mathbb{F}$) is another simple example of LHEncap. Unlike commitments and encryptions, this is an example of a *keyless and deterministic* encapsulation, that preserves linearly-homomorphic operations in the exponent.

4.2 SNARKs for Low-Degree Testing on LHEncap

In this section, we define SNARKs for the problem of low-degree testing on LHEncap. This problem is defined as follows: Let $\mathbb{F}$ be a field, $d \in \mathbb{N}$ a degree-bound, $\theta > 0$ a proximity parameter, $D \subseteq \mathbb{F}$ a domain, (Setup, KeyGen, Encap, Eval) an LHEncap scheme, and com a non-interactive vector commitment. The prover wants to convince the verifier that it knows $[\![y_1]\!], \ldots, [\![y_{|D|}]\!]$, such that:

1. com is a commitment to these encapsulations
2. These encapsulations form a low-degree polynomial. This is formalized in two ways depending on the underlying LHEncap.
   - If LHEncap is decryptable, then there exists some degree d polynomial f, i.e., $f \in \mathsf{RS}[\mathbb{F}, D, \rho = d/|D|]$ (see Definition 1) s.t., for each $i \in [|D|]$, the decryption y_i of $[\![y_i]\!]$ agrees with f on $D[i]$.
   - Else, if LHEncap is linearly-homomorphic w.r.t. randomness, then there exists some function $f : D \to \mathcal{S}$, i.e., $f \in \mathsf{RS}[\mathcal{S}, D, \rho = d/|D|]$ s.t., for each $i \in [|D|]$, $[\![y_i]\!]$ agrees with $f(x)$ on $D[i]$. Here, $\mathsf{RS}[\mathcal{S}, D, \rho]$ denotes the Reed-Solomon codes *over the output-space* $\mathcal{S}$ of LHEncap, i.e., it denotes the set of all functions $f : D \to \mathcal{S}$ for which there exists $\hat{f} \in \mathcal{S}^{<\rho|D|}[x]$ agreeing with f on D. As a concrete example, consider group exponentiation. The evaluation of $g^{f(x)}$ on D is a Reed-Solomon codes $\in \mathsf{RS}[\mathbb{G}, D, \rho = d/|D|]$ over $\mathbb{G}$.

We give a formal definition of SNARKs for low-degree testing on LHEncap below. We use $\delta(f,g)$ to denote the *fractional Hamming distance*, i.e.,

$$\delta(f,g) = \frac{1}{|D|} \cdot |\{x \in D : f(x) \neq g(x)\}| .$$

Definition 4 (SNARKs for Low-Degree Testing on LHEncap)**.** *Let $\mathbb{F}$ be a field, $d \in \mathbb{N}$ a degree-bound, $\theta > 0$ a proximity parameter and $D \subseteq \mathbb{F}$ a domain. Let* (LHEncap.Setup, LHEncap.KeyGen, LHEncap.Encap, LHEncap.Eval) *be an LHEncap scheme and let* (Com.Gen, Com.Commit, Com.Open) *be a deterministic vector commitment scheme.*[25] *A SNARK for low-degree testing on hidden values is a triple of algorithms* (SNARK.Setup, SNARK.Prover, SNARK.Verify) *with the following syntax:*

1. $\mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda)$*: On input the security parameter λ, the setup algorithm outputs a common reference string* crs*.*
2. $(\mathsf{com}, \pi) \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, [\![y_1]\!], \ldots, [\![y_{|D|}]\!])$*: On input the common reference string* crs *and a list of encapsulations $[\![y_1]\!], \ldots, [\![y_{|D|}]\!]$, the prover algorithm outputs a proof π and a vector commitment* com*.*
3. $b \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{com}, \pi)$*: On input the common reference string* crs*, a vector commitment* com*, and proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.*

These algorithms satisfy the following properties:

- ***Completeness:*** *For any vector of honestly generated encapsulations $([\![y_1]\!], \ldots, [\![y_{|D|}]\!])$, and any commitment* com *of this vector. If there exists an f^0, such that for each $i \in [|D|]$, y_i agrees with f^0 on $D[i]$ and $\delta(f^0, \mathsf{RS}[\mathbb{F}, D, \rho]) = 0$, then it holds that*

$$\Pr\left[1 \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{com}, \pi) \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{SNARK.Prover}(\mathsf{crs}, \mathsf{com}, [\![y_1]\!], \ldots, [\![y_{|D|}]\!]) \end{array}\right] \geqslant 1 - \mathsf{negl}(\lambda).$$

- ***Knowledge Soundness:*** *There exists an extractor $\mathcal{E}$ and knowledge error $\varepsilon : [0,1] \to [0,1]$, such that, for any PPT algorithm P^*, the following holds.*
  - *If LHEncap is* ***linearly-homomorphic w.r.t., randomness****, then the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{com} \neq \mathsf{Com.commit}([\![y_1]\!], \ldots, [\![y_{|D|}]\!]) \vee \\ ([\![y_1]\!], \ldots, [\![y_{|D|}]\!]) \text{ is } \theta \text{ far from RS code} \end{array} \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda) \\ (\mathsf{com}, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}) \\ 1 \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{com}, \pi) \\ [\![y_1]\!], \ldots, [\![y_{|D|}]\!] \leftarrow \mathcal{E}_{P^*}(\mathsf{crs}) \end{array}\right] \leqslant \varepsilon(\theta),$$

    where "θ far from RS codeword" means there exists some $f^0 : D \to \mathcal{S}$ such that $\forall i \in [|D|]$, $[\![y_i]\!]$ agrees with f^0 on $D[i]$ and $\delta(f^0, \mathsf{RS}[\mathcal{S}, D, \rho]) \geqslant \theta$.
  - *Else, if LHEncap is* ***decryptable****, then the following holds:*

$$\Pr\left[\begin{array}{c} \mathsf{com} \neq \mathsf{Com.commit}([\![y_1]\!], \ldots, [\![y_{|D|}]\!]) \vee \\ \textit{Decryption of } ([\![y_1]\!], \ldots, [\![y_{|D|}]\!]) \textit{ is } \theta \textit{ far from RS code} \end{array} \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{SNARK.Setup}(1^\lambda) \\ (\mathsf{com}, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}) \\ 1 \leftarrow \mathsf{SNARK.Verify}(\mathsf{crs}, \mathsf{com}, \pi) \\ [\![y_1]\!], \ldots, [\![y_{|D|}]\!] \leftarrow \mathcal{E}_{P^*}(\mathsf{crs}) \end{array}\right] \leqslant \varepsilon(\theta),$$

    where "decryption is θ far from RS codeword" means there exists an $f^0 : D \to \mathbb{F}$ such that $\forall i \in [|D|]$, the decryption y_i of $[\![y_i]\!]$ agrees with f^0 on $D[i]$ and $\delta(f^0, \mathsf{RS}[\mathbb{F}, D, \rho]) \geqslant \theta$.

[25] For instance, the Merkle hash tree-based commitment.

Note that here, the encapsulations $[\![y_1]\!], \ldots, [\![y_{|D|}]\!]$ output by the extractor $\mathcal{E}$ may or may not correspond to well-formed/honestly generated encapsulations.

- ***Succinctness:*** *The proof size should be sublinear in the instance size, i.e., $|\pi| \in o(d, |D|)$.*

4.3 FRI on LHEncap

In this section, we present our key observation about the FRI [BBHR18] protocol. In particular, we show how the FRI protocol can be modified to work over hidden values. We refer to the modified protocol as *FRI on LHEncap*. At a high-level, FRI on LHEncap is an interactive oracle proof [BCS16] for the following problem: Let $\mathbb{F}$ be a field, $d \in \mathbb{N}$ a degree-bound, $\theta > 0$ a proximity parameter, $D \subseteq \mathbb{F}$ a domain and (KeyGen, Encap, Eval) an LHEncap scheme. The prover is provided with encapsulations of the evaluations of some function f on domain D, i.e., $\{[\![f(i)]\!]\}_{i \in D}$ (we use the shorthand notation $[\![f(x)|_D]\!]$ to denote this set of encapsulations). The prover gives the verifier oracle access to these encapsulations. The prover wants to convince the verifier that $[\![f(x)|_D]\!]$ are encapsulations of evaluations of some degree d polynomial on this domain D. Namely, that $f \in \mathsf{RS}[\mathbb{F}, D, \rho = d/|D|]$.

We provide a formal description of this protocol below (as discussed in Remark 1, for simplicity, here we assume that $\mathbb{F}$ is an FFT-friendly field. However, similar ideas can be extended to arbitrary fields using techniques from [BCKL23, BCKL22]).

FRI on LHEncap

Setup: Let $D = \{\omega^0, \ldots, \omega^{|D|-1}\}$ be the set of $|D|^{\text{th}}$ roots of unity and $\mathbb{H} = \{\nu^0, \ldots, \nu^{d-1}\}$ be the set of d^{th} roots of unity. Let $[\![f|_{\mathbb{H}}]\!] = [\![f(\nu^0)]\!], \ldots, [\![f(\nu^{d-1})]\!] \in \mathcal{S}^d$ denote the instance, where $f \in \mathbb{F}^{\leq d}[x]$ and $\mathcal{S}$ denotes the output-space of some LHEncap.[a] Let $\mathsf{RS}[\mathbb{F}, D, \rho]$ be a familiy of Reed-Solomon codes. We assume that the prover P has $[\![f|_{\mathbb{H}}]\!]$.

Commit Phase: The commit phase proceeds as follows:

- P uses FFT techniques on the set $[\![f|_{\mathbb{H}}]\!]$ to compute encapsulations $[\![f|_D]\!] = [\![f(\omega^0)]\!], \ldots, [\![f(\omega^{|D|-1})]\!]$. It sends oracle access to these encapsulations $[\![f|_D]\!]$ to V.
- Set $f_0 = f$. For rounds $i = 0, \ldots, \log_2 d - 1$, the protocol proceeds as follows:
  1. V samples a random challenge $\alpha_i \in \mathbb{F}$ and sends it to P.
  2. For each $j \in [0, |D|/2^i - 1]$, P computes
  $$[\![f_{i+1}(\omega^{2^{i+1} \cdot j})]\!] = \left(\frac{[\![f_i(\omega^{2^i \cdot j})]\!] + [\![f_i(-\omega^{2^i \cdot j})]\!]}{2} + \alpha_i \cdot \frac{[\![f_i(\omega^{2^i \cdot j})]\!] - [\![f_i(-\omega^{2^i \cdot j})]\!]}{2 \cdot \omega^{2^i \cdot j}} \right).$$
  3. Let $D_{i+1} = \left\{\omega^{2^{i+1} \cdot j}\right\}_{j \in [0, |D|/2^i - 1]}$ and give the verifier oracle access to $[\![f_{i+1}|_{D_{i+1}}]\!]$.
- At the end, P simply sends encapsulations of all evaluations of the last constant polynomial on domain $D_{\log_2 d}$, i.e., $\left\{[\![f_{\log_2 d}(\beta)]\!]\right\}_{\beta \in D_{\log_2 d}}$ to V.

Query Phase: The query phase proceeds as follows:

- V repeats the following ℓ times:
  1. Sample a random $s_0 \in D$.
  2. For each $i \in [0, \log_2 d - 1]$, set $s_{i+1} = s_i^2$. Let $s'_i = -s_i \neq s_i$ be the other element in D_i, such that $s_{i+1} = (s'_i)^2$. Query oracle $[\![f|_{D_i}]\!]$ on evaluation points s_i and s'_i to get $[\![f_i(s_i)]\!]$ and $[\![f_i(s'_i)]\!]$. Check if the following holds:
     - If LHEncap is *linearly-homomorphic w.r.t. randomness*, check if

$$[\![f_{i+1}(s_{i+1})]\!] = \left(\frac{[\![f_i(s_i)]\!] + [\![f_i(s'_i)]\!]}{2} + \alpha_i \cdot \frac{[\![f_i(s_i)]\!] - [\![f_i(s'_i)]\!]}{2 \cdot s_i} \right).$$

     - Else, if LHEncap is *decryptable*, then check if $[\![f_{i+1}(s_{i+1})]\!]$ and $\left(\frac{[\![f_i(s_i)]\!]+[\![f_i(s'_i)]\!]}{2} + \alpha_i \cdot \frac{[\![f_i(s_i)]\!]-[\![f_i(s'_i)]\!]}{2 \cdot s_i} \right)$ decrypt to the same value.
- If LHEncap is *linearly-homomorphic w.r.t. randomness*, check if, for all $\beta \in D_{\log_2 d}$, $[\![f_{\log_2 d}(\beta)]\!]$ is the same encapsulation. Else, if LHEncap is *decryptable*, then check if for all $\beta \in D_{\log_2 d}$, $[\![f_{\log_2 d}(\beta)]\!]$ decrypt to the same value.
- If all the above checks verify, output accept, else output reject.

[a] Note that this implies that the encapsulations $[\![f(\nu^0)]\!], \ldots, [\![f(\nu^{d-1})]\!]$ may or may not correspond to well-formed/honestly generated encapsulations.

Completeness and Soundness. It is easy to see that the completeness of the above protocol follows from the completeness of the FRI protocol and linearly-homomorphic property of LHEncap. Recently, Block et al. [BGK+23] presented a formal proof establishing that the FRI IOP satisfies round-by-round knowledge soundness [CCH+19]. Similar ideas also naturally extend to our FRI on LHEncap protocol. Formally speaking, the following theorem follows from [BGK+23].

Theorem 1 (Round-by-Round Knowledge Soundness of FRI on LHEncap)**.** *Let $\mathbb{F}$ be a finite field, a degree-bound $d \in \mathbb{N}$, $\rho = d/|D|$, a domain $D \subseteq \mathbb{F}$. For any integer $m \geqslant 3$, $\eta \in (0, \sqrt{\rho}/2m)$, $\theta \in (0, 1 - \sqrt{\rho} - \eta)$, encapsulations of evaluations of a function f that has relative distance $\geqslant \theta$ from* RS $[\mathbb{F}, D, \rho = d/|D|]$*, FRI on LHEncap has the following* round-by-round knowledge soundness error *(See [CCH+19] for the definition)*

$$\varepsilon = \max \left\{ \frac{(m + 1/2)^7 \cdot |D|^2}{3\rho^{3/2}|\mathbb{F}|}, (1 - \theta)^\ell \right\}.$$

Proof Sketch. If the underlying LHEncap is *decryptable*, then round-by-round knowledge soundness of FRI on LHEncap can be reduced to the round-by-round knowledge soundness of FRI [BGK+23]. In particular, assuming existence of a PPT adversary $\mathcal{A}$ who can break the round-by-round knowledge soundness of FRI on LHEncap, we can design another PPT adversary $\mathcal{B}$ who can break the round-by-round knowledge soundness of FRI as follows: Given an instance, $\mathcal{B}$ computes encapsulations of this instance and forwards them to $\mathcal{A}$. $\mathcal{B}$ forwards all the verifier challenges to $\mathcal{A}$. Whenever $\mathcal{A}$ sends an oracle of encapsulated values (which may or may not correspond to honestly generated encapsulations), $\mathcal{B}$ queries this oracle at all locations, decrypts the encapsulated responses, and gives oracle access to these decrypted values to its verifier (in case any of these responses are not well-formed or they decrypt to $\perp$, $\mathcal{B}$ samples some garbage

value and uses that instead of $\perp$). It is easy to see that if $\mathcal{A}$ manages to violate round-by-round knowledge soundness of FRI on LHEncap, then $\mathcal{B}$ will be able to break round-by-round knowledge soundness of FRI.

If the underlying LHEncap is *linearly-homomorphic w.r.t. randomness*, then round-by-round knowledge soundness of FRI on LHEncap can be argued in a similar manner as is done for FRI in [BGK+23]. This is because, in this case, all the checks by the verifier are performed on the *encapsulations* (and not on the encapsulated *values*). This is essentially equivalent to running the FRI protocol, albeit on the encapsulations. □

SNARK for Low-Degree Testing on LHEncap. Applying the [BCS16] transformation on the above protocol yields a SNARK (in the random-oracle model) for low-degree testing on LHEncap. It is easy to see that completeness of the resulting SNARK follows from the completeness of the FRI protocol, the [BCS16] transformation and linearly-homomorphic property of LHEncap. For knowledge soundness, we obtain the following corollary.[26]

Corollary 1 (Knowledge Soundness of FRI based SNARK for Low-Degree Testing on LHEncap)**.** *Let $\mathbb{F}$ be a finite field, $d \in \mathbb{N}$ a degree-bound, $\rho = d/|D|$, a domain $D \subseteq \mathbb{F}$. For any integer $m \geqslant 3$, $\eta \in (0, \sqrt{\rho}/2m)$, $\theta \in (0, 1 - \sqrt{\rho} - \eta)$, random oracle $\mathcal{H}$, query bound $Q \in \mathbb{N}$, compiling FRI on LHEncap on encapsulations of evaluations of a function f that has relative distance $\geqslant \theta$ from $\mathsf{RS}[\mathbb{F}, D, \rho = d/|D|]$, using the [BCS16] compiler yields a SNARK for low-degree testing on LHEncap with the following knowledge soundness error:*

$$\varepsilon = Q \cdot \max\left\{\frac{(m+1/2)^7 \cdot |D|^2}{3\rho^{3/2}|\mathbb{F}|}, (1-\theta)^\ell\right\} + \frac{3(Q^2+1)}{d}.$$

Complexity. The dominant part of the prover's work in the above protocol is the FFT operations for extending the evaluation domain of the polynomial. Assuming $|D| = O(d)$, this complexity is $O(d \log d)$. The proof involves sending oracles (i.e., commitments to these oracles in the non-interactive variant) to the verifier in each round. Additionally, to ensure negligible knowledge soundness error, the verifier makes $O(\lambda \cdot \log d)$ oracle queries; in particular, λ queries per layer. In the non-interactive variant, these oracle queries correspond to vector commitment openings – for Merkle Hash-based commitment, the size of the opening proof for each query is $O(\log d)$. As a result, in the non-interactive protocol, the proof size is $O(\lambda \cdot \log^2 d)$, and the verifier also runs in $O(\lambda \cdot \log^2 d)$ time. In the rest of the paper, we do not explicitly write the λ factor when describing the efficiency of the scheme. We emphasize that the above protocol does not make any non-black-box use of the underlying LHEncap scheme.

5 Polynomial Commitments on Hidden Values

In this section, we present our polynomial commitment scheme on hidden values. In Section 5.1, we formalize the notion of polynomial commitment schemes on hidden values. In Section 5.2, we present a construction of this primitive using the FRI-based SNARK for low-degree testing on LHEncap from Section 4.3.

5.1 Defining Polynomial Commitments on LHEncap

In this section, we formalize a variant of polynomial commitments, that we refer to as *polynomial commitments on LHEncap*.

[26] For appropriately chosen parameters, the knowledge soundness error ε in Corollary 1 can be made negligible in λ.

Definition 5 (Polynomial Commitments on LHEncap)**.** *A polynomial commitment on LHEncap for degree d polynomials consists of a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Com})$ *and a SNARK (see Definition 2) for a specific problem, defined as follows.*

1. $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$*: The generator takes the security parameter λ as input and outputs public parameters* pp*.*
2. $\sigma \leftarrow \mathsf{Com}(\mathsf{pp}, \{x_i, [\![y_i]\!]\}_{i\in[d+1]})$*: The commit algorithm takes as input the public parameters* pp*, the evaluation points $\{x_i\}_{i\in[d+1]} \in \mathbb{F}^{d+1}$ (in the clear) and the evaluations $\{[\![y_i]\!]\}_{i\in[d+1]}$ (hidden inside LHEncap). It outputs a succinct commitment σ.*
3. *A SNARK* $\Pi_{\mathsf{SNARK}} = (\mathsf{SNARK.Setup}, \mathsf{SNARK.Prove}, \mathsf{SNARK.Verify})$ *where the committer* C *acts as the prover and the receiver* R *acts as the verifier, for the following problem:*
   - *Statement:* $\mathsf{st} = \left(\mathsf{pp}, \sigma, \{x_i\}_{i\in[d+1]}, x^*, [\![y^*]\!]\right)$
   - *Witness:* $\mathsf{wit} = \left(\{[\![y_i]\!]\}_{i\in[d+1]}\right)$
   - *Relation $\mathcal{R}_{\mathsf{poly}}$: (1) σ is a commitment to* wit*. (2) Let $f \in \mathbb{F}^{\leqslant d}[x]$ be the polynomial defined by interpolating $y_1, \ldots, y_{d+1}$, then $[\![y^*]\!]$ is an encapsulation of $f(x^*)$.*

   We use the shorthand $b \leftarrow \Pi_{\mathsf{SNARK}}\langle \mathsf{C}(\mathsf{st}, \mathsf{wit}, \mathcal{R}_{\mathsf{poly}}), \mathsf{R}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}})\rangle$, where $b \in \{0,1\}$, to denote the output of the above SNARK.

A polynomial commitment on LHEncap must satisfy the following.

Correctness: *For correctness, the verifier should always accept honestly generated proofs. Formally speaking, for all polynomials $f \in \mathbb{F}^{\leqslant d}[x]$, any set of evaluation points $\{x_i\}_{i\in[d+1]}$ and any x^*, it holds that*

$$\Pr\left[b = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), \sigma \leftarrow \mathsf{Com}(\mathsf{pp}, \{x_i, [\![f(x_i)]\!]\}_{i\in[d+1]}) \\ [\![y^*]\!] = \mathcal{L}([\![y_1]\!], \ldots, [\![y_n]\!]) \\ \mathsf{st} = \left(\mathsf{pp}, \sigma, \{x_i\}_{i\in[d+1]}, x^*, [\![y^*]\!]\right) \\ \mathsf{wit} = \left(\{[\![f(x_i)]\!]\}_{i\in[d+1]}\right) \\ b \leftarrow \Pi_{\mathsf{SNARK}}\langle \mathsf{C}(\mathsf{st}, \mathsf{wit}, \mathcal{R}_{\mathsf{poly}}), \mathsf{R}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}})\rangle \end{array}\right] = 1,$$

where $\mathcal{L}$ is the linear function defined by the Lagrange interpolation w.r.t. $\{x_i\}_{i\in[d+1]}$ and x^*.

Binding/Extraction: *The binding property requires that there exists an extractor $\mathcal{E}$, such that for any PPT adversary $\mathcal{A}$, any $\{x_i\}_{i\in[d+1]}$, and any x^*, the following holds.*

- *If LHEncap is* ***linearly-homomorphic w.r.t. randomness****, then we have:*

$$\Pr\left[[\![y^*]\!] \neq \mathcal{L}([\![y_1]\!], \ldots, [\![y_n]\!]) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathcal{A}(\mathsf{pp}, \{x_i\}_{i\in[d+1]}) \\ \mathsf{st} = \left(\mathsf{pp}, \sigma, \{x_i\}_{i\in[d+1]}, x^*, [\![y^*]\!]\right) \\ 1 \leftarrow \Pi_{\mathsf{SNARK}}\langle \mathcal{A}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}}), \mathsf{R}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}})\rangle \\ \left(\{[\![y_i]\!]\}_{i\in[d+1]}\right) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{pp}) \end{array}\right] \leqslant \mathsf{negl}(\lambda),$$

  where $\mathcal{L}$ is the linear function defined by the Lagrange interpolation w.r.t. $\{x_i\}_{i\in[d+1]}$ and x^*.

- *Else, if LHEncap is* ***decryptable****, then we have:*

$$\Pr\left[\begin{array}{l} y^* \neq f(x^*) \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathcal{A}(\mathsf{pp}, \{x_i\}_{i\in[d+1]}) \\ \mathsf{st} = \left(\mathsf{pp}, \sigma, \{x_i\}_{i\in[d+1]}, x^*, [\![y^*]\!]\right) \\ 1 \leftarrow \Pi_{\mathsf{SNARK}}\langle \mathcal{A}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}}), \mathsf{R}(\mathsf{st}, \mathcal{R}_{\mathsf{poly}})\rangle \\ \left(\{[\![y_i]\!]\}_{i\in[d+1]}\right) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{pp}) \end{array} \right] \leqslant \mathsf{negl}(\lambda),$$

where $f \in \mathbb{F}^{\leqslant d}[x]$ is uniquely defined by, $\forall i \in [d+1]$, the decryption y_i of $[\![y_i]\!]$ agrees with $f(x_i)$.

5.2 Constructing Polynomial Commitments on LHEncap

We are now ready to present a construction of polynomial commitment on LHEncap using our construction of FRI based SNARK for low-degree testing on LHEncap (see Section 4.3). We present a formal description of this protocol below.

Polynomial Commitment on LHEncap

Gen: Let $D = \{\omega^0, \ldots, \omega^{|D|-1}\}$ be the set of $|D|^{\text{th}}$ roots of unity and $\mathbb{H} = \{\nu^0, \ldots, \nu^{d-1}\}$ be the set of d^{th} roots of unity. Let RS $[\mathbb{F}, D, \rho]$) be a familiy of Reed-Solomon Codes.

Commit: Commit to encapsulations $[\![f|_{\mathbb{H}}]\!] = [\![f(\nu^0)]\!], \ldots, [\![f(\nu^{d-1})]\!]$, where $f \in \mathbb{F}^{\leqslant d}[x]$ as follows:

- Using FFT techniques, the committer first expands the evaluation from $[\![f|_{\mathbb{H}}]\!]$ to $[\![f|_D]\!] = [\![f(\omega^0)]\!], \ldots, [\![f(\omega^{|D|-1})]\!]$.
- The commitment is computed as a Merkle hash on the vector $[\![f(w^0)]\!], \ldots, [\![f(w^{|D|-1})]\!]$.

Opening Proof: When asked to give an opening proof at $x = x^*$, the committer first computes $[\![f(x^*)]\!]$ as follows: (1) it uses FFT techniques on $[\![f(w^0)]\!], \ldots, [\![f(w^{|D|-1})]\!]$ to obtain encapsulations of the coefficients of $f(x)$ and then (2) it takes an inner product of these with $(1, x^*, (x^*)^2, \ldots, (x^*)^d)$.

Next, the committer outputs $[\![f(x^*)]\!]$ and runs the FRI-based SNARK for low-degree testing on LHEncap on $\left\{\frac{[\![f(x)-y^*]\!]}{x-x^*}\right\}_{x\in D}$ with domain D, degree bound $d-1$, and proximity parameter $\theta < \frac{1-\rho}{2}$. Recall that, as discussed in Section 2.2, oracle access to this encapsulated quotient polynomial can be emulated using oracle access to encapsulated evaluations $[\![f(x)]\!]$ of f.

Correctness and Binding. Correctness of this polynomial commitment scheme follows from the completeness of the FRI based SNARK for low-degree testing on LHEncap protocol and from linear homomorphism of the underlying LHEncap scheme. The binding property follows from the binding property of the vector commitment and from the knowledge soundness of the FRI based SNARK for low-degree testing on LHEncap. A formal proof for this follows exactly as that for arguing the binding property of the polynomial commitment scheme from FRI [BGKS20, Hab22].

Complexity. This scheme inherits the efficiency properties from FRI on LHEncap. In particular, the opening proof size and verification time for the opening proof are $O(\log^2 d)$. The time to generate the commitment is $O(d \log d)$.

6 Efficiently Verifiable Private Delegation of Computation

In this section, we present an approach for *efficiently verifiable* FHE-based private delegation of computation, which only makes black-box use of cryptography. As discussed in Section 2.3, our high-level idea is to have the client send FHE ciphertexts encrypting its input to the server. The server then FHE evaluates the given function on these ciphertexts (sends the resulting ciphertext to the client), and finally, it generates a SNARK proof to prove that the FHE evaluation was done correctly. We demonstrate how to implement this high-level approach using polynomial IOP based SNARKs and our polynomial commitment scheme on LHEncap (see Section 5). This section is organized as follows — in Section 6.1, we give an overview of polynomial IOP based SNARKs and then proceed to present our main construction in Section 6.2.

6.1 Overview of Polynomial IOP based SNARKs

We start by giving an overview of the basic blueprint of the polynomial IOP-based SNARK proof system.

Representing the NP Relation as a System of Constraints. Different SNARKs work with different representations of the relation associated with the statement being proven, e.g., quadratic arithmetic programs [PHGR13b, SVdV16], low-depth circuits [BTVW14, CMT12, GKR08, Tha13, VSBW13, WHG+16, WTS+18, XZZ+19], binary arithmetic circuits [GWC19], etc. Each of these representations can be viewed as a system of low-degree constraints. The most popular representation amongst state-of-the-art proof systems [BCR+19, CHM+20, Gro16, GM17, COS20] is known as the rank-1 constraint systems (R1CS) that generalizes arithmetic circuits.

The first step in all existing state-of-the-art SNARKs is to *extend* the given (short) statement-witness (x, w) pair into a satisfying assignment (also called the extended-witness) for the representative constraint system. This is typically done by evaluating the original relation circuit on the statement-witness pair to obtain a computation trace (from which the satisfying assignment can be derived using only linear operations). The next step is to design an argument of knowledge for this satisfying assignment w.r.t. the relevant constraint system.

Almost all state-of-the-art efficient SNARKs are designed using a polynomial IOP-based approach as follows: (1) Design an information-theoretic polynomial IOP for the corresponding system of constraints. (2) Compile the polynomial IOP into an interactive proof system for the same set of constraints using a polynomial commitment scheme. (3) Finally, transforming this interactive proof into a non-interactive proof system in the random oracle model using the Fiat-Shamir heuristics [FS87]. In more detail:

1. **Polynomial IOP.** A polynomial IOP [BCR+19] is a variant of IOP [BCS16], where the oracles sent by the prover to the verifier are essentially polynomials. In other words, in each round of the polynomial-IOP, the prover provides the verifier with an oracle access to a polynomial function, which the verifier can query at any location of its choice.

2. **Polynomial IOP + Polynomial Commitments = Interactive Proof.** A polynomial IOP can be transformed into an interactive protocol by enabling the prover to emulate the process of giving polynomial oracles to the verifier using a polynomial commitment scheme as follows: The prover sends a commitment of the polynomial to the verifier. Every time the verifier wishes to query this polynomial, the prover responds with the corresponding evaluation along with a succint proof certifying that this evaluation was consistent with the commitment.

3. **SNARK in the Random Oracle Model.** If the polynomial IOP is a public-coin protocol, then the above transformation yields a public-coin interactive protocol. Such a protocol can be easily

transformed into a non-interactive proof system in the random oracle model using the Fiat-Shamir heuristics [FS87].

Overview of the polynomial IOP used in Fractal [COS20] . As discussed in Section 2.3, while our approach can be made to work with any polynomial IOP based SNARK, in order to minimize the multiplicative depth of FHE evaluations, we instantiate it with Fractal [COS20]. Hence, it is instructive to recall the main operations used in the polynomial IOP that Fractal is based on. At a high level, this polynomial IOP performs the following operations on the extended witness (let $\mathcal{R}$ be the relation for which the proof is being generated):

- Encoding the extended witness as polynomials, which only requires linear operations of size $|\mathcal{R}|$ on the extended witness.
- Some additional linear operations of size $|\mathcal{R}|$ on these witness dependent polynomials.
- Fast Fourier transformations on these witness dependent polynomials, which also only require linear operations of size $|\mathcal{R}| \log |\mathcal{R}|$.
- Multiplying a constant number of witness-dependent polynomials of size $|\mathcal{R}|$.
- Sum-check on witness-dependent polynomials, which requires dividing the witness-dependent polynomial by a public polynomial, and hence only requires linear operations of size $|\mathcal{R}|$.

Overall, our main observation is that given the computation trace/extended witness, the prover only needs to perform constant-depth computations of size $|\mathcal{R}| \log |\mathcal{R}|$ in this polynomial IOP. Moreover, this is a *hologographic* polynomial IOP, meaning that the verifier only does not need to read the entire relation circuit. Instead, it suffices for the verifier to simply get oracle access to the relation. Holographic polynomial IOP can be transformed into a *pre-processing* SNARK, where the description of the relation is pre-processed by a trusted party and given to the verifier at the start of the protocol. This ensures that the overall runtime of the verifier remains sublinear in the size of the relation circuit. As discussed in Section 1, for our application, we assume that this pre-processing of the relation circuit is either done by a trusted party or can be done by the client himself.

Finally, we note that, while we explicitly list out the main computation steps in the Fractal polynomial IOP, most of these steps are, in fact, common to all known polynomial IOP constructions. For instance, the underlying polynomial IOP used in Plonk [GWC19] also uses the above steps, with the only difference that instead of using sum-check, it relies on a *product check*. This product check, however, requires the prover to perform computations on the extended witness that have a total multiplicative depth proportional to the size of the entire relation, which may not be desirable for efficiency reasons.

6.2 Our Construction

We now present our construction for verifiable private delegation of computation.

Notation. We start by establishing some notation. Given some input x, let C be the computation that the client wishes to delegate on this input. To verify that the computation was done correctly, we want the server to generate a SNARK attesting that the output ciphertext is an encryption of the result of computing function C on the input. This is essentially equivalent to proving the following relation on values hidden (or encapsulated) inside FHE ciphertexts: $\mathcal{R} : \mathsf{output} = C(\mathsf{input})$.

Let x be the client's input, let y be the output of C on x, i.e., $y = C(x)$, Let $\vec{z} = (z_1, \ldots, z_{|C|})$ be the list of values induced on all the intermediate wires in circuit C when it is evaluated using input x. Let $\mathsf{ExtWit}_{\mathcal{R}}(x, \vec{z}, y)$ be a *linear function* that allows the prover to generate an extended-witness for $\mathcal{R}$, given the computation-trace $(x, \vec{z}, y)$.

Protocol. Our construction works as follows:

- **Client → Server:** The client samples an FHE secret key, and encrypts x under this key. It then sends the resulting ciphertext (say $[\![x]\!]$) to the server.
- **Server:** The server proceeds as follows:
  1. *Output Computation:* It performs FHE evaluation (for C) on $[\![x]\!]$ to compute the output ciphertext (say $[\![y]\!]$). Without loss of generality, we assume that in the process of evaluating $[\![y]\!]$, the server also obtains FHE encryptions of the computation trace, i.e., $([\![z_1]\!], \ldots, [\![z_{|C|}]\!])$.
  2. *Extended Witness Generation:* It uses FHE evaluation (for $\mathsf{ExtWit}_{\mathcal{R}}$) on $[\![x]\!], [\![z_1]\!], \ldots, [\![z_{|C|}]\!], [\![y]\!]$ to obtain ciphertexts corresponding to the extended witness (say $\overrightarrow{[\![w]\!]}$).
  3. *SNARK Generation:* It uses FHE evaluations (for the operations needed to generate a polynomial IOP for relation $\mathcal{R}$) on $\overrightarrow{[\![w]\!]}$. As a result of this, it obtains FHE encryptions corresponding to the polynomials that the prover sends as oracles to the verifier, in the underlying polynomial IOP. It uses our polynomial commitment on $\mathsf{LHEncap}$ from Section 5.2 to compute commitments to these encrypted polynomials. As is the case with any Fiat-Shamir transformed protocol, the verifier messages and its oracle queries are derived by querying the random oracle.
- **Server → Client:** Finally, the server sends the encapsulated SNARK along with the output of the computation $[\![y]\!]$ to the client.
- **Client:** The client decrypts $[\![y]\!]$ and the encryptions sent as part of the SNARK. In order to verify if the decrypted y was honestly computed, it then checks if the decrypted values correspond to a valid SNARK. Note that, while the client performs all of the checks on decrypted values, the verifier's challenge sampled by Fiat-Shamir heuristics is computed using random oracle queries on the encrypted proof transcript sent by the server.

Private Verification. We note that the above approach only gives us a *privately verifiable* SNARK. This is because, in Plonk, the verifier's check involves checking if some function of the proof is equal to 0. While a verifier could always rely on the homomorphism of FHE and compute the necessary checking function on encryptions of the SNARK proof (without using the FHE secret key), it has no way of checking if *the resulting ciphertext is an encryption of 0 or not.* However, the verifier does hold the FHE secret key for the verification of this step. Hence, this is indeed a privately verifiable SNARK. We refer the reader to Section 1.2 for a detailed comparison of our construction with prior work.

Security. It is easy to see that the soundness of the above SNARK follows from the soundness of the underlying polynomial IOP and the binding property of our polynomial commitment on $\mathsf{LHEncap}$. This is similar to Theorem 1 in Section 4.3 on why the soundness of FRI on $\mathsf{LHEncap}$ reduces to the soundness of standard FRI when the underlying $\mathsf{LHEncap}$ is *decryptable*.

Efficiency. In the output computation step, the server needs to perform $O(|C|)$ FHE evaluations with the same multiplicative depth as $|C|$. Our approach can be instantiated using any polynomial IOP. When

instantiating the above approach using the Fractal polynomial IOP [COS20], extended witness and SNARK generation only require the server to perform $O(|C| \log |C|)$ additional FHE evaluations with a constant multiplicative depth. Similarly, when instantiating the above approach using the Plonk IOP [GWC19], the SNARK generation will require the server to perform $O(|C| \log |C|)$ additional linear operations on the FHE ciphertexts and $O(|C|)$ FHE evaluations with a $O(|C|)$ multiplicative depth. In either case, the work done by the client to verify the SNARK remains polylogarithmic in $|C|$.

Why SNARKs? We note that our high-level idea of avoiding non-black-box use of cryptography by computing the information-theoretic primitive (i.e., the polynomial IOP) inside the FHE and then using polynomial commitment on hidden values to compile it into the desired cryptographic proof, is not exclusive to SNARKs. It can also be used alongside other types of proof systems, such as those based on MPC-in-the-head [IKOS07]. We choose to work with SNARKs since these are succinct proofs and quick to verify. The size of MPC-in-the-head-based proof systems is typically much larger — in this case, the client would have to spend a significant amount of time simply to verify the computation. This would defeat the whole purpose of delegating computation to an external server in the first place.

7 Private Outsourcing of zkSNARKs to a Single Server

In this section, we demonstrate how a prover can delegate the generation zkSNARKs to a single untrusted server in a privacy-preserving manner, without making any non-black-box use of cryptography. For this, we show how our protocol from the previous section can be modified to enable the server to generate a *publicly verifiable* zkSNARK on hidden values. Towards this, our first idea is to use fully homomorphic commitments (FHCom) instead of FHE.

Fully-Homomorphic Commitments. A fully homomorphic commitment is an FHE scheme with the following additional properties:

- The ciphertexts should be statistically binding.
- Given knowledge of the secret key sk, and, it is possible to generate a decommitment string r for any ciphertext c encrypting a message m, such that it is possible for anyone to verify that c is indeed an encryption of m only given c, r, m. Importantly, revealing r does not leak the secret key or anything else beyond the decommitted value m.

Gorbunov, Vaikuntanathan, and Wichs [GVW15] noted that the LWE-based Gentry-Sahai-Waters FHE scheme [GSW13] satisfy both of the above properties and is an exampled of FHCom. However, one caveat is that the public key in this scheme must be sampled "honestly"; otherwise, the ciphertexts can be equivocated. An honest sampling of the public key can be ensured by either allowing a trusted party to generate it and give the corresponding secret key to the client, or having the client sample it and generate a proof attesting to the well-formedness of this public key.

Notation. Before describing the protocol, it would be helpful to establish some notation. Let $\mathcal{R}$ be the relation for which the client wants to generate a zkSNARK. Let x be the statement and w be the corresponding witness. Let $\vec{z} = (z_1, \ldots, z_{|\mathcal{R}|})$ be the list of values induced on all the intermediate wires in the circuit representing $\mathcal{R}$ when it is evaluated using input x, w. Let $\mathsf{ExtWit}_{\mathcal{R}}(x, \vec{z}, w)$ be a *linear function* that allows the prover to generate an extended-witness for $\mathcal{R}$, given the computation-trace $(x, \vec{z}, w)$.

Protocol. Our construction works as follows:

- **Prover → Server:** We assume that the prover either samples an FHCom public-secret key pair (and appends to the public-key, a proof certifying that the public key is well-formed) or it receives a public-secret key pair from a trusted party. It encrypts w under the secret key and then sends the resulting ciphertext (say $[\![w]\!]$) to the server.

- **Server:** The server proceeds as follows:

  1. *Computation trace:* It performs FHCom evaluation (for the circuit representing $\mathcal{R}$) on the public statement x and encrypted witness $[\![w]\!]$ to compute the FHE encryptions of the computation trace, i.e., $([\![z_1]\!], \ldots, [\![z_{|\mathcal{R}|}]\!])$.
  2. *Extended Witness Generation:* It uses FHCom evaluation (for $\mathsf{ExtWit}_{\mathcal{R}}$) on $[\![x]\!], [\![z_1]\!], \ldots, [\![z_{|\mathcal{R}|}]\!], [\![w]\!]$ to obtain ciphertexts corresponding to the extended witness (say $\overrightarrow{[\![w]\!]}$).
  3. *SNARK Generation:* It uses FHCom evaluations (for the operations needed to generate a polynomial IOP for relation $\mathcal{R}$) on $\overrightarrow{[\![w]\!]}$. As a result, it obtains FHE encryptions corresponding to the polynomials that the prover sends as oracles to the verifier, in the underlying polynomial IOP. It uses our polynomial commitment on $\mathsf{LHEncap}$ from Section 5.2 to compute commitments to these encrypted polynomials. As is the case with any Fiat-Shamir transformed protocol, the verifier messages and its oracle queries are derived by querying the random oracle.

- **Server → Prover:** Finally, the server sends the encapsulated SNARK along with the output of the computation $[\![y]\!]$ to the prover.

- **Prover → Verifier:** The prover decrypts all the encryptions sent as part of the zkSNARK and also generates the decommitment string for each of these ciphertexts. It forwards all the ciphertexts associated with the zkSNARK received from the server, along with the public key, the decrypted values of these ciphertexts and their corresponding decommitment strings to the verifier.

- **Verifier:** In order to verify the zkSNARK, the verifier first checks if all the decommitments are valid. It then proceeds to check if the decommitted values correspond to a valid SNARK. As before, we note that while the verifier performs all of the checks to verify the validity of zkSNARK on the decommitted values, it recomputes the random oracle queries on the encrypted proof transcript sent by the prover.

It is easy to see that the resulting zkSNARK is publicly verifiable. The efficiency of this scheme follows from the efficiency of the previous construction. Let $s(|\mathcal{R}|)$ be the size of the underlying zkSNARK used to instantiate the above protocol. The size of the resulting publicly verifiable zkSNARK is roughly $s(|\mathcal{R}|)$ plus the size of $s(|\mathcal{R}|)$ decommitments, which is $O(s(|\mathcal{R}|))$.

The verifier checks if the decommitments corresponding to the encrypted zkSNARK are valid and then checks the validity of the decrypted zkSNARK. Hence, the soundness of the resulting publicly verifiable zkSNARK can be reduced to the statistical binding of FHCom and to the soundness of the privately verifiable SNARK from our previous application. The zero-knowledge property of the resulting publicly verifiable zkSNARK follows from the zero-knowledge property of the underlying zkSNARK and the fact the revealing decommitment strings for certain ciphertexts, do not leak anything beyond the values encrypted in these ciphertexts.

Remark 2 (On the Necessity of Client's Assistance)**.** *The above construction crucially relies on the client to decrypt the message sent by the server in order to eventually obtain a publicly verifiable SNARK. We note*

that in the setting, where the server only computes on encrypted values, it is impossible *to design a delegation scheme (for zkSNARKs for any non-trivial class of languages), where the client does not need to intervene and the verifier can directly check the encrypted proof. This is because, such a scheme would be in clear violation of the semantic security of the encryption scheme.*

8 Weighted Threshold Signatures without Setup

We now demonstrate how our technique of FRI on hidden values can be used for constructing weighted threshold signatures without setup. This section is organized as follows. In Section 8.1, we describe one example construction based on the Schnorr signature. We choose Schnorr as an example to highlight the fact that our techniques do not require pairing. In Section 8.2, we discuss on the versatility and further extensions to our basic construction.

8.1 A Construction based on Schnorr

We start with a formal definition for weighted threshold signature without setup, which is adapted from [CKM23] for static security with *concurrent* three-round signing protocol.

Definition 6. *Let* $\mathbf{W} = (w_1, \ldots, w_n)$ *be the (public) weights associated with n parties. A* weighted threshold signature without setup *consists of a tuple of algorithms with the following syntax.*

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*: on input the security parameter* 1^λ*, parties use* KeyGen *to independently sample their respective key pairs* $(\mathsf{sk}, \mathsf{pk})$*.*
- $\mathsf{vk} \leftarrow \mathsf{Preprocess}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathbf{W})$*: on input the public keys and weights of all parties, a* succinct *verification key* vk *can be* deterministically *derived using the preprocessing algorithm.*
- $\left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i \in B} \leftarrow (\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3)$*: This is a three-round signing protocol that can be run by any subset B of parties. In particular, given a subset* $B \subseteq [n]$ *and a message* msg *to sign, each party* $i \in B$ *does:*
  - *Round 1:* $(\rho_1^i, \mathsf{st}_i) \leftarrow \mathsf{Sign}_1(B, \mathsf{msg}, \mathsf{sk}_i)$*: this outputs the secret state* st_i *and first-round message* ρ_1^i*.*
  - *Round 2:* $\rho_2^i \leftarrow \mathsf{Sign}_2\left(B, \mathsf{msg}, \mathsf{sk}_i, \mathsf{st}_i, \left\{\rho_1^i\right\}_{i \in B}\right)$*: given the first-round messages from all parties, this outputs the second-round message.*
  - *Round 3:* $\rho_3^i \leftarrow \mathsf{Sign}_3\left(B, \mathsf{msg}, \mathsf{sk}_i, \mathsf{st}_i, \left\{\rho_1^i, \rho_2^i\right\}_{i \in B}\right)$*: given the transcript from the first two rounds, this outputs the third-round message.*
- $(T, \sigma) \leftarrow \mathsf{Agg}\left(\mathsf{msg}, \mathbf{W}, \left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i \in B}\right)$*: on input the message* msg*, the transcript of a signing protocol, and the weights associated with all parties* $\mathbf{W}$*, this algorithm outputs an aggregated signature* σ *and a* claimed threshold $T = \sum_{i \in B} w_i$*.*
- $b \leftarrow \mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, (T, \sigma))$*: on input the verification key* vk*, the message* msg*, and the signature* (T, σ)*, this verification algorithm outputs a bit b indicating* accept *or* reject*.*

These algorithms must satisfy the following properties.

- ***Correctness.*** *The honestly generated signature should always verify. That is, for any security parameter λ, any set of weights $\mathbf{W}$, any message* msg*, and any subset of signers B, it holds that*

$$\Pr\left[\mathsf{Ver}(\mathsf{vk},\mathsf{msg},(T,\sigma))=1 \;\middle|\; \begin{array}{c} \forall i\in[n],\ (\mathsf{sk}_i,\mathsf{pk}_i)\leftarrow\mathsf{KeyGen}(1^\lambda) \\ \mathsf{vk}\leftarrow\mathsf{Preprocess}(\mathsf{pk}_1,\ldots,\mathsf{pk}_n,\mathbf{W}) \\ \left\{\rho_1^i,\rho_2^i,\rho_3^i\right\}_{i\in B}\leftarrow(\mathsf{Sign}_1,\mathsf{Sign}_2,\mathsf{Sign}_3) \\ (T,\sigma)\leftarrow\mathsf{Agg}\left(\mathsf{msg},\mathbf{W},\left\{\rho_1^i,\rho_2^i,\rho_3^i\right\}_{i\in B}\right) \end{array}\right]=1.$$

- ***Security.*** *Any PPT adversary, who may request an arbitrary polynomial many concurrent signing sessions, should not be able to forge a valid signature with non-negligible probability. In particular, the probability that the adversary wins the following game is* $\mathsf{negl}(\lambda)$.

  1. *The adversary picks n, the weights $\mathbf{W}$, and a subset of parties to corrupt $A\subseteq[n]$.*
  2. *For every honest party $i\in[n]\setminus A$, the public key is sampled honestly by the challenger $(\mathsf{pk}_i,\mathsf{sk}_i)\leftarrow\mathsf{KeyGen}(1^\lambda)$. For every corrupted party $i\in A$, the adversary picks a public key pk_i and sends it to the challenger.*
  3. *The challenger invokes $\mathsf{vk}\leftarrow\mathsf{Preprocess}(\mathsf{pk}_1,\ldots,\mathsf{pk}_n,\mathbf{W})$.*
  4. *The adversary can repeatedly request new signing sessions. For each session, it specifies a message* msg *and a subset $B\subseteq[n]$. The adversary is rushing in that, for each round, the honest parties in B send the messages first. These sessions can be concurrent. Moreover, the adversary does not need to complete the protocol. Let $Q=\emptyset$. For each session requested, let $Q=Q\cup\{\mathsf{msg}\}$.*
  5. *In the end, the adversary outputs a message msg^* and a signature (T^*,σ^*). The adversary wins the forgery game if (1) $\mathsf{msg}^*\notin Q$, (2) $T^*>\sum_{i\in A}w_i$, and (3) $\mathsf{Ver}(\mathsf{vk},\mathsf{msg}^*,(T^*,\sigma^*))=1$. That is, the adversary produces a valid signature for a message he never requests any signing session and a threshold higher than the cumulative weights of the corrupted parties.*

Remark 3. *We make a few remarks to add some perspectives about this definition.*

- *As mandated by the "no setup" requirement, each party picks its own key pairs by* KeyGen*. To enable fast verification, a one-time* Preprocess *will produce a succinct verification key* vk*. This is a deterministic algorithm that can be run/checked by any party. Note that parties do not need to know their weights to sample their own keys; only the verification key* vk *will depend on the weights $\mathbf{W}$.*

- *In our definition, each signature comes with a* signature-specific threshold *T. This is in stark contrast to a standard threshold signature scheme where there is a* predefined threshold *T for all signatures. Intuitively, the adversary wins the forgery game as long as it can sign for any threshold T higher than the total corrupted weights $\sum_{i\in A}w_i$. This is another significant advantage of our scheme in cases where one wants to construct a threshold signature scheme with multiple thresholds.*[27] *For example, imagine a voting scheme based on threshold signatures, our scheme allows one to give a succinct proof for exactly how many weights support one candidate; standard threshold signature can only prove that, say, the majority of the weights support a candidate.*

[27] In order to enable this for secret sharing-based threshold signature schemes, parties need to run an instance of DKG for each threshold it aims to support, i.e., to sample a T-out-of-n secret sharing for each supported threshold T.

Before we describe our construction, we establish some notations and recall a useful lemma known as the generalized sumcheck [BCR+19, RZ21].

Notations for Polynomials. Let $\{\omega, \omega^2, \ldots, \omega^n = 1\} \subseteq \mathbb{F}$ be a multiplicative subgroup of the finite field $\mathbb{F}$. Let $L_1(x), L_2(x), \ldots, L_n(x)$ be the Lagrange basis polynomial. That is, L_i is the unique degree-$(n-1)$ polynomial defined by: $L_i(\omega^j)$ is 1 when $i = j$ and 0 when $i \neq j$. Let $Z(x) = \prod_{i=1}^{n}(x - \omega^i)$ be the vanishing polynomial. It is known that $Z(x) = x^n - 1$ and $L_i(x) = \frac{\omega^i}{n} \cdot \frac{x^n - 1}{x - \omega^i}$. Note that $L_i(0) = 1/n$. Our construction relies on the following lemma known as generalized sumcheck (refer to, for instance, Theorem 1 of [RZ21] for a proof).

Lemma 1 (Generalized Sumcheck [BCR+19, RZ21])**.** *Let $A(x) = \sum_{i=1}^{n} a_i \cdot L_i(x)$, $B(x) = \sum_{i=1}^{n} b_i \cdot L_i(x)$. It holds that*

$$A(x) \cdot B(x) = \frac{\sum_i a_i \cdot b_i}{n} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x),$$

where Q_x is a polynomial with degree $\leqslant n - 2$. Note that, given $A(x)$ and $B(x)$, the quotient polynomials $Q_x(x)$ and $Q_Z(x)$ can be computed efficiently (using the evaluation form of a polynomial) as

$$Q_x(x) = \left(C(x) - \frac{\sum_i a_i \cdot b_i}{n}\right) \cdot x^{-1} \quad \text{and} \quad Q_Z(x) = \Big(A(x) \cdot B(x) - C(x)\Big) \cdot Z(x)^{-1},$$

where $C(x)$ is the polynomial that interpolates $(a_1 b_1, \ldots, a_n b_n)$, i.e., $C(x) = \sum_i a_i b_i \cdot L_i(x)$.

We are now ready to present our construction. Our construction is based on the standard commit-and-sign three-round signing Schnorr-based scheme. We prove our security in the algebraic group model (AGM) [FKL18].[28]

Theorem 2. *Assuming that Discrete Log (DL) is hard, the construction in Figure 1 is a weighted threshold signature without setup (Definition 6) in the algebraic group model (AGM) and random oracle model, achieving the following efficiency: the signature size and verification time is $O(\log^2 n)$, the partial signature aggregation time is $O(n \log n)$.*

Notations: We use Com and HCom for polynomial commitment for polynomials defined in the clear and by LHEncap, respectively. Let RO, RO′, RO′′ be random oracles.

KeyGen(1^λ): Let $\mathbb{G}$ be the cryptographic group with order p and generator g defined by the security parameter 1^λ. Sample $\mathsf{sk} \leftarrow \mathbb{F}_p$ and set $\mathsf{pk} = g^{\mathsf{sk}}$.

Preprocess($\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathbf{W}$): Let the polynomials defined by the secret keys and weights be

$$\mathsf{SK}(x) = \sum_i \mathsf{sk}_i \cdot L_i(x) \qquad W(x) = \sum_i w_i \cdot L_i(x).$$

Note that $W(x)$ is in the clear, and $\mathsf{SK}(x)$ is encapsulated (as group exponentiation) by the public keys. Let $\mathsf{vk} = (\mathsf{HCom}(\mathsf{SK}), \mathsf{Com}(W))$. That is, the verification key is the polynomial commitment of the hidden polynomial SK and the polynomial W in the clear. Note that this is a deterministic process.

Signing. Given a message msg and a subset $B \subseteq [n]$ of signing parties, party P_i does the following.

- Sign_1: Sample $r_i \leftarrow \mathbb{F}_p$ and commits to g^{r_i} by sending $\rho_1^i = \mathsf{RO}(g^{r_i})$.

[28] In AGM, the adversary is assumed to be *algebraic*, which means that any group elements output by the adversary need to be explained by a linear combination of the group elements that it takes as input.

- Sign_2: Given all the commitments $\{\rho_1^i\}_{i\in B}$, it decommits by sending g^{r_i}.
- Sign_3: Given $\{\rho_1^i, \rho_2^i\}_{i\in B}$, it checks the consistency of the decommitments. If so, it computes $g^r = \prod_{i\in B} g^{r_i}$. The random challenge is computed as $c = \mathsf{RO}'(\mathsf{msg}, g^r)$. It sends $\rho_3^i = c \cdot \mathsf{sk}_i + r_i$.

$\underline{\mathsf{Agg}\left(\mathsf{msg}, \mathbf{W}, \left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i\in B}\right)}$. The aggregator verifies the partial signature by $g^{\rho_3^i} \stackrel{?}{=} \mathsf{pk}_i^c \cdot g^{r_i}$. If any of the partial signatures do not verify, it aborts. Otherwise, it proceeds as follows. It first sets the claimed threshold $T = \sum_{i\in B} w_i$, the aggregation of the public keys $\mathsf{apk} = \prod_{i\in B} \mathsf{pk}_i$, and the aggregated signature $\sigma' = (g^r, \sum_{i\in B} \rho_3^i)$. Next, it generates a proof π for the honest aggregation of apk.

1. Generates the polynomial commitment $\mathsf{Com}(B)$ to the polynomial $B(x) = \sum_i b_i \cdot L_i(x)$, where $b_i = \mathbb{1}_{i\in B}$. Computes the quotient polynomials as in Lemma 1.

$$\mathsf{SK}(x) \cdot B(x) = \frac{\sum_i \mathsf{sk}_i \cdot b_i}{n} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x) \tag{1}$$

$$W(x) \cdot B(x) = \frac{\sum_i w_i \cdot b_i}{n} + Q'_x(x) \cdot x + Q'_Z(x) \cdot Z(x) \tag{2}$$

   Note that, even though the aggregator does not know $\mathsf{SK}(x)$, it does have access to the encapsulation of $\mathsf{SK}(x)$. Consequently, it can also compute the *encapsulation* of $Q_x(x)$ and $Q_Z(x)$.

2. Computes the following quotient polynomial, which checks that $B(x) \in \{0, 1\}$ for all $x \in \mathbb{H}$.

$$B(x) \cdot (1 - B(x)) = Q(x) \cdot Z(x). \tag{3}$$

3. Generate the polynomial commitment to the quotient polynomials

$$\mathsf{commitments} = \left(\mathsf{HCom}(Q_x),\ \mathsf{HCom}(Q_Z),\ \mathsf{Com}(Q'_x),\ \mathsf{Com}(Q'_Z),\ \mathsf{Com}(Q)\right).$$

   Note that, $Q_x(x)$ and $Q_Z(x)$ are polynomials encapsulated using group exponentiation, while the rest are in the clear.

4. Sample a random challenge r via the random oracle as $r = \mathsf{RO}''(\mathsf{commitments})$. The aggregator now generates the opening proof proving that the polynomials at $x = r$ evaluate to

$$g^{\mathsf{SK}(r)},\ g^{Q_x(r)},\ g^{Q_Z(r)},\ W(r),\ B(r),\ Q'_x(r),\ Q'_Z(r),\ Q(r).$$

5. This completes the generation of proof π: it consists of (1) all the polynomial commitments, (2) the evaluations of all the polynomials at $x = r$, and (3) all the opening proofs.

The final signature is (T, σ), where $\sigma = (\mathsf{apk}, \sigma', \pi)$.

$\underline{\mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, (T, \sigma)).}$ Parse $\sigma = (\mathsf{apk}, \sigma', \pi)$. The verifier does the following.

- It first verifies the validity of apk under the claimed threshold T (i.e., apk is the aggregation of public keys with cumulative weights $\geqslant T$). This is done in two steps: (1) it checks that the opening proofs for all polynomial commitments at $x = r$ are correct; (2) it checks that Equation 1, 2, and 3 hold at $x = r$.[29]
- If apk verifies, it checks that σ' is a valid signature for msg under apk. This is done by parsing $\sigma' = (R, z) \in \mathbb{G} \times \mathbb{F}_p$ and checking $g^z = \mathsf{apk}^c \cdot R$, where $c = \mathsf{RO}'(\mathsf{msg}, R)$.

Figure 1: Our weighted threshold signature without setup based on Schnorr

Proof. The correctness is straightforward due to the correctness of the polynomial identities. On the efficiency side, the aggregated signature consists of a constant number of opening proofs for the FRI-based polynomial schemes plus a constant number of additional field and group elements. By the efficiency of FRI, the aggregated signature size and verification time is $O(\log^2 n)$. When aggregating the partial signatures, the aggregator needs to compute the quotient polynomials in Equation 1, 2, and 3. This computation requires FFT (as noted in Lemma 1) and, hence, the aggregation time is $O(n \log n)$.

Next, we prove the unforgeability in two steps. We consider an intermediate hybrid, where the only difference is in what is the aggregated signature and how the verification works. This hybrid is exactly the trivial threshold signature scheme with a large aggregated signature and linear-time verification algorithm, as we discussed in the technical overview (Section 2.5). In particular, the (large) aggregated signature is exactly (B, σ) and the verification key is $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. The verification works as (1) first compute apk herself as $\prod_{i \in B} \mathsf{pk}_i$ and (2) verify σ under apk. We prove that if there is an adversary $\mathcal{A}$ that breaks the unforgeability game, there is an adversary $\mathcal{A}'$ that breaks the unforgeability game in this intermediate hybrid. Finally, if there is an $\mathcal{A}'$ that breaks the intermediate hybrid, there is an adversary $\mathcal{A}''$ that breaks the DL assumption in the random oracle model.

Suppose there is an adversary $\mathcal{A}$ that breaks the original unforgeability game. We construct a $\mathcal{A}'$ that breaks the unforgeability of the intermediate hybrid as follows. $\mathcal{A}^*$ will simply simulate $\mathcal{A}$, i.e., set the same weights, corrupt the same party, and request the same signing sessions. In each signing session, it will send the same messages and relays the honest parties' message to $\mathcal{A}$. Finally, at some point, with a non-negligible probability, $\mathcal{A}$ will output a forgery. Now, by the knowledge soundness of the FRI-based polynomial commitment (Theorem 1), $\mathcal{A}'$ will extract the polynomial $B(x)$ from the forged aggregation signature. Since Equation 1, 2, and 3 hold at random point $x = r$, the polynomial identities must hold except with probablity $\leqslant \mathsf{poly}(\lambda)/|\mathbb{F}|$ by Schwartz-Zippel. This means that the set B encoded inside the polynomial $B(x)$ is a subset with cumulative weight $> T$. Moreover, the aggregated public key apk in the forged signature must be an honest aggregation of public keys in B. Therefore, if $\mathcal{A}'$ now simply sends (B, σ) as her forgery in this intermediate hybrid, it will also win the forgery game. Therefore, $\mathcal{A}'$ breaks the unforgeability in this hybrid with a non-negligible probability.

Finally, we show that if there is an adversary $\mathcal{A}'$ that breaks the unforgeability of the hybrid, we construct an adversary that breaks the DL assumption in AGM. This part is similar to typical proofs of Schnorr-based multisignature/threshold signature schemes. Let $\mathcal{A}''$ be an adversary for the DL problem. That is, it receives a challenge g^x from the external challenger. It proceeds to simulate a forgery game with $\mathcal{A}'$. For every honest party $i \in [n] \setminus A$, it picks a random a_i and sets its public key to be a rerandomization $\mathsf{pk}_i = g^{x+a_i}$ of g^x. $\mathcal{A}'$ will send the public keys of the corrupted party $\{\mathsf{pk}_i\}_{i \in A}$. In AGM, $\mathcal{A}''$ can extract

[29] Note that, even though Equation 1 is not in the clear, anyone can verify it at $x = r$ using group operations. In particular, no term involves the multiplication of an encapsulated polynomial with another encapsulated polynomial, i.e., no pairing is required.

the discrete log of these public keys as the algebraic adversary $\mathcal{A}'$ has to explain each group element as a linear combination of the group elements it takes as input, which is only g in this case. Now, for each signing session with signers from B, $\mathcal{A}''$ simulates as follows. It sends an arbitrary random string ρ_1^i as the commitment of honest parties' commitment. After $\mathcal{A}'$ sends its commitment, $\mathcal{A}''$ extracts the committed g^{r_i} from the malicious parties. It picks a random challenge c and sets g^r to be some $g^{y-c\cdot x} = g^y/(g^x)^c$ for a randomly sampled y. Next, it picks random ρ_2^i as the honest parties' decommitment conditioned on that $\prod_{i\in B}\rho_2^i = g^{y-c\cdot x}$. It further programs the random oracle to be consistent with the commitment $\rho_2^i = \mathsf{RO}(\rho_1^i)$ and the Fiat-Shamir challenge $c = \mathsf{RO}'(\mathsf{msg}, g^{y-c\cdot x})$. Finally, to send honest parties' partial signatures in the final round, it sends $\rho_3^i = c\cdot(x+a_i)+(y-c\cdot x) = c\cdot a_i + y$, which it can compute without the knowledge of x. This allows $\mathcal{A}''$ to simulate $\mathcal{A}'$ entirely according to the right distribution. Now, with a non-negligible probability, $\mathcal{A}'$ will generate a valid forgery $(B^*, \sigma^*) = (B^*, (R^*, z^*))$ under msg^*. Note that, to qualify as a forgery, B^* must have cumulative weights higher than the corrupted parties. Consequently, there must be an honest party in B^*. By our setting, we have $\mathsf{apk} = \prod_{i\in B^*}\mathsf{pk}_i = g^{b\cdot x+a}$ for some a and b. Here, b is the number of honest parties in B^*, which is non-zero and a is the sum of honest parties' a_i and the sum of the corrupted parties' sk_i. Since $\mathcal{A}''$ knows all of them, it knows both a and b in the clear. By AGM, $\mathcal{A}''$ could also extract R^* as some $g^{u\cdot x+v}$. Now, this gives a linear equation on x as $c\cdot(b\cdot x+a)+(u\cdot x+v)$, which $\mathcal{A}''$ can solve for x. This is because $cb+u$ is non-zero with overwhelming probability; this is even true when the adversary may pick the subset $B\subseteq[n]$, which determines b, even after knowing c and u. This finishes the reduction, which completes the entire proof. □

Remark 4. *In our definition of security, we consider the setting where the adversary picks its public key independent of the honest parties' public keys. If the adversary may pick its public key adaptively, there is a potential* rogue key attack *[BDN18]. There are two standard approaches to prevent this attack. First, we may rerandomize every public key from* pk_i *to* $(\mathsf{pk}_i)^{\alpha_i}$*, where* $\alpha_i = \mathsf{RO}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{pk}_i)$ *is sampled by some random oracle. In AGM, this means that one can extract the adversary's public key* pk *as* $g^{a\cdot x+b}$ *for some* a *and* b*, and the rest of the security proof is essentially the same.*[30] *Second, we may ask each party to provide a proof of knowledge of* sk_i*. Again, one could use the knowledge extractor to extract* sk_i *and the proof is essentially the same. We omit these details for ease of presentation.*

We also note that the use of AGM is somewhat inherent. If one wishes to use forking lemma [PS00, BN06, BDL19] and rewind-based proof strategies, the following issue (particular to our construction) would arise. Since the adversary may pick a different B *for each forgery, the security proof has to treat each corrupted party's secret key as a formal variable. In order to solve the linear system involving* x *and* $\{\mathsf{sk}_i\}_{i\in A}$*, one would need to rewind approximately* $|A|+1$ *times to solve* x*. However, the success probability would decay exponentially in the number of rewinds, making this proof strategy fail. We leave removing the need of AGM as an exciting future work.*

8.2 Extensions

In this section, we briefly discuss the possible extensions to our basic scheme.

Policies beyond the weighted setting. In our basic construction, the proof π in the aggregated signature can be divided into two parts π_1 and π_2. π_1 proves that apk is an honest aggregation of the public keys from some set B, and π_2 proves that B has sufficient high weights. In fact, this π_2 is nothing but a particular SNARK for the weighted threshold gate. One can replace this SNARK with any other SNARKs to prove any properties about B. In particular, for any access structure that can be described as an arithmetic circuit

[30] In particular, in AGM, we do not need the multiple rewind techniques employed in [MPSW19, NRS21].

C, one can use a SNARK to prove that B is an authorized set. To facilitate this, the following changes are necessary: (1) as part of the verification key, instead of generating a polynomial commitment to $\mathbf{W}$, the preprocessing will generate a succinct verification key for the circuit C; (2) for the aggregated signature, proof π_2 is generated to prove that B is an authorized set under C; and (3) the verification will first use the verification key to verify that B is an authorized set (instead of verifying that B has sufficient weights). This is highly modular, and one can plug in any SNARK of its choice.

Instantiating with other multisignature. In the basic construction, we use a Schnorr-based multisignature scheme with a three-round signing as one particular example. One may imagine plugging any multisignature scheme for this construction. As we discussed in Section 2.5, as long as the multisignature scheme supports public key aggregation and the trivial threshold signature scheme induced by the multisignature scheme is secure, our compiler simply serves to make the verification step succinct. For example, one may use Schnorr multisignature with two-round signing [NRS21, PW23] or BLS multisignature [BDN18] as the underlying multisignature schemes. There is no fundamental barrier to instantiating our construction with these other multisignature schemes. It is, however, beyond the scope of this paper to present the details of all these constructions. We will next present a brief sketch of the construction based on BLS and compare it with related works.

Sketch of the Construction based on BLS. In the BLS signature, parties' key pairs are $\{\mathsf{sk}, \mathsf{pk} = g^{\mathsf{sk}}\}$. To sign a message msg, the signature is $\sigma = \mathsf{RO}(\mathsf{msg})^{\mathsf{sk}}$, which can be verified using pairing as $e(\sigma, g) \stackrel{?}{=} e(\mathsf{RO}(\mathsf{msg}), \mathsf{pk})$. Due to its homomorphism, BLS naturally supports aggregation for the signatures and the public keys, which enables a multisignature scheme [BDN18] with *non-interactive* signing. In particular, each party signs the message as $\sigma_i = \mathsf{RO}(\mathsf{msg})^{\mathsf{sk}_i}$ and the aggregated signature $\sigma = \prod_{i \in B} \sigma_i$ will verify under the aggregation of the public key $\mathsf{apk} = \prod_{i \in B} \mathsf{pk}_i$. Again, in a trivial threshold signature scheme, the aggregator may simply output (B, σ) as the threshold signature and ask the verifier to compute apk and the cumulative weight herself. This can be similarly compiled into a weighted threshold signature with succinct verification, just like our construction in the previous section. As we discussed in Section 2.5, prior works [GJM+24, DCX+23] also construct a weighted threshold BLS signature using this framework. The comparison between our construction and their works is presented in Table 1.

	SRS	Hint (per party)	Aggregation time	Signature size	Verification time
[GJM+24, DCX+23]	KZG SRS	$O(n)$	$O(n)$	$O(1)$	$O(1)$
This work	N/A	N/A	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$

Table 1: Comparison between our scheme and [GJM+24, DCX+23] for weighted threshold BLS

Acknowledgement

The authors would like to thank Dario Fiore, Ignacio Cascudo, Daniele Cozzo, and Paola de Perthuis for useful discussions on the security of verifying FHE delegation and also for bringing several related works to our attention. The first and third authors are supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by Visa Inc, BAIR Commons Meta Fund, Stellar Development Foundation, and a Bakar Fellows Spark Award.

References

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_12. 8

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14165-2_14. 5, 8

[AN21] Benny Applebaum and Oded Nir. Upslices, downslices, and secret-sharing with complexity of 1.5^n. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 627–655, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84252-9_21. 7

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. doi:10.1109/SP.2018.00020. 3, 8

[BBHR18] 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, *ICALP 2018: 45th International Colloquium on Automata, Languages and Programming*, volume 107 of *LIPIcs*, pages 14:1–14:17, Prague, Czech Republic, July 9–13, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.14. 3, 11, 18, 22

[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, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 327–357, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_12. 8

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017. URL: https://doi.org/10.1007/s00145-016-9241-9. 3

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 528–558, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-75248-4_19. 8

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press. doi:10.1109/SP.2014.36. 3

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 336–365, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-70700-6_12. 3

[BCG+18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-03326-2_20. 3

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 19–46, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-64378-2_2. 3

[BCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st Annual Symposium on Foundations of Computer Science*, pages 900–909, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press. doi:10.1109/FOCS46700.2020.00088. 3

[BCKL22] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves - (ECFFT part II). In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 467–496, Chicago, IL, USA, November 7–10, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-22318-1_17. 3, 13, 22

[BCKL23] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ECFFT) part I: low-degree extension in time $O(n \log n)$ over all finite fields. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 700–737. SIAM, 2023. URL: https://doi.org/10.1137/1.9781611977554.ch30. 13, 22

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 275–304, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-07085-3_10. 3

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17653-2_4. 27, 34

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53644-5_2. 11, 22, 24, 27

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 649–680, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84242-0_23. 50

[BDL19] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 607–636, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-34618-8_21. 37

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-03329-3_15. 16, 37, 38

[BGJ+23] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 1454–1470. IEEE, 2023. 6

[BGK+23] Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zając. Fiat-shamir security of fri and related snarks. ASIACRYPT 2023, 2023. https://eprint.iacr.org/2023/1071. URL: https://eprint.iacr.org/2023/1071. 23, 24

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020: 11th Innovations in Theoretical Computer Science Conference*, volume 151, pages 5:1–5:32, Seattle, WA, USA, January 12–14, 2020. LIPIcs. doi:10.4230/LIPIcs.ITCS.2020.5. 3, 4, 13, 14, 26

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-22792-9_7. 5, 8

[BHS23] Fabrice Benhamouda, Shai Halevi, and Lev Stambler. Weighted secret sharing from wiretap channels. In *ITC 2023*, 2023. URL: https://eprint.iacr.org/2022/1578. 7, 9

[BHV+23] Rishabh Bhadauria, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Wenxuan Wu, and Yupeng Zhang. Private polynomial commitments and applications to mpc. In *PKC 2023*, 2023. URL: https://eprint.iacr.org/2023/680. 10

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-45682-1_30. 7, 9

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 65–97, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-92078-4_3. 8, 50

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. doi:10.1145/1180405.1180453. 37

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany. doi:10.1007/3-540-36288-6_3. 16

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. Cryptology ePrint Archive, Report 2014/846, 2014. https://eprint.iacr.org/2014/846. 27

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-30617-4_17. 3

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1082–1090, Phoenix, AZ, USA, June 23–26, 2019. ACM Press. doi:10.1145/3313276.3316380. 23

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of

Lecture Notes in Computer Science, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-45721-1_26. 3, 27

[CK21] Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. Cryptology ePrint Archive, Report 2021/916, 2021. https://eprint.iacr.org/2021/916. 7, 9

[CKM23] Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive schnorr threshold signatures. CRYPTO 2023, 2023. https://eprint.iacr.org/2023/445. 32

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_26. 5, 8

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6453–6469, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa. 5, 6

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 90–112, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090245. 27

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-45721-1_27. 4, 6, 14, 15, 27, 28, 30

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bunz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. CCS 2023, 2023. https://eprint.iacr.org/2023/598. URL: https://eprint.iacr.org/2023/598. 7, 9, 10, 16, 38

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. doi:10.1109/SP46214.2022.9833584. 6

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany. 20

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 844–855, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. doi:10.1145/2660267.2660366. 8

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96881-0_2. 34

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 124–154, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-45388-6_5. 8

[FS87] 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*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. doi:10.1007/3-540-47721-7_12. 9, 11, 27, 28

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. doi:10.1145/1536414.1536440. 4, 20

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-Knowledge SNARKs as a service. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4427–4444, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/garg. 5, 6

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_25. 5, 8

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_37. 3

[GHAH+23] Matthew Green, Mathias Hall-Andersen, Eric Hennenfent, Gabriel Kaptchuk, Benjamin Perez, and Gijs Van Laer. Efficient proofs of software exploitability for real-world processors. *Proceedings on Privacy Enhancing Technologies*, 2023(1):627–640, January 2023. doi:10.56553/popets-2023-0036. 3

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes*

in Computer Science, pages 458–487, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-06944-4_16. 6

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007. doi:10.1007/s00145-006-0347-3. 6

[GJM+23] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: Mpc, encryption and signatures. In *CRYPTO 2023*, 2023. URL: https://eprint.iacr.org/2022/1632. 7, 9

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. IEEE S&P 2024, 2024. https://eprint.iacr.org/2023/567. URL: https://eprint.iacr.org/2023/567. 6, 7, 9, 10, 16, 38

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 113–122, Victoria, BC, Canada, May 17–20, 2008. ACM Press. doi:10.1145/1374376.1374396. 27

[GLS+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021. https://eprint.iacr.org/2021/1043. 3

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-63715-0_20. 27

[GMN22] Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. In Ittay Eyal and Juan A. Garay, editors, *FC 2022: 26th International Conference on Financial Cryptography and Data Security*, volume 13411 of *Lecture Notes in Computer Science*, pages 203–229, Grenada, May 2–6, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-18283-9_10. 50

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-17373-8_19. 3

[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, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_11. 3, 27, 50

[Gro21] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. https://eprint.iacr.org/2021/339. 6

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_5. 16, 20, 30

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 469–477, Portland, OR, USA, June 14–17, 2015. ACM Press. doi:10.1145/2746539.2746576. 16, 20, 30

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953. 3, 14, 15, 27, 28, 30

[Hab22] Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022. https://eprint.iacr.org/2022/1216. 26

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30, San Diego, CA, USA, June 11–13, 2007. ACM Press. doi:10.1145/1250790.1250794. 30

[KGC+18] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 1353–1370, Baltimore, MD, USA, August 15–17, 2018. USENIX Association. 3

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 723–732, Victoria, BC, Canada, May 4–6, 1992. ACM Press. doi:10.1145/129712.129782. 3

[KMP20] Abhiram Kothapalli, Elisaweta Masserova, and Bryan Parno. A direct construction for asymptotically optimal zkSNARKs. Cryptology ePrint Archive, Report 2020/1318, 2020. https://eprint.iacr.org/2020/1318. 3

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1751–1767, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3423364. 6

[KPV22] Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 1725–1737, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. doi:10.1145/3548606.3560657. 3

[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*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-17373-8_11. 7, 10, 13, 50

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 1–34, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-90453-1_1. 3, 8, 13

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. doi:10.1109/SFCS.1994.365746. 3

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Des. Codes Cryptogr.*, 87(9):2139–2164, 2019. 16, 37

[MRV+21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *2021 IEEE Symposium on Security and Privacy*, pages 626–641, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. doi:10.1109/SP40001.2021.00096. 7, 9

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84242-0_8. 16, 37, 38

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany. doi:10.1007/3-540-48910-X_16. 20

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany. doi:10.1007/3-540-46766-1_9. 8, 20

[PHGR13a] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press. doi:10.1109/SP.2013.47. 3

[PHGR13b] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013. doi:10.1109/SP.2013.47. 27

[Plo21] Plonky2. Plonky2, 2021. URL: https://github.com/mir-protocol/plonky2. 3

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. doi:10.1007/s001450010003. 37

[PW23] Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 597–627, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-30589-4_21. 38

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. doi:10.1145/1060590.1060603. 20

[RPX+22] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. Cryptology ePrint Archive, Paper 2022/1286, 2022. https://eprint.iacr.org/2022/1286. URL: https://eprint.iacr.org/2022/1286. 3

[RZ21] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 774–804, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-84242-0_27. 34

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. doi:10.1007/0-387-34805-0_22. 3, 7

[SCP+22] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 3001–3018, Boston, MA, USA, August 10–12, 2022. USENIX Association. 50

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-56877-1_25. 3

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. 6

[SVdV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 346–366, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-39555-5_19. 27

[TCZ+20] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy*, pages 877–893, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press. doi:10.1109/SP40000.2020.00059. 6

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40084-1_5. 27

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237. IEEE Computer Society, 2013. doi:10.1109/SP.2013.48. 27

[WHG+16] Riad S. Wahby, Max Howald, Siddharth Garg, Abhi Shelat, and Michael Walfish. Verifiable asics. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 759–778. IEEE Computer Society, 2016. doi:10.1109/SP.2016.51. 27

[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. 27

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-15985-5_11. 3

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26954-8_24. 3, 27

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 2039–2053, Virtual Event, USA, November 9–13, 2020. ACM Press. doi:10.1145/3372297.3417278. 3

[ZkR21] ZkRollups. An incomplete guide to rollups, 2021. URL: https://vitalik.ca/general/2021/01/05/rollup.html. 3

[ZLW+21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 159–177, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press. doi:10.1145/3460120.3484767. 3

A Public Aggregation of KZG Opening Proofs

For completeness, we discuss the utility of our techniques in aggregating SNARK proofs. There are many works recently on publicly aggregating SNARK proofs: [BDFG21] for KZG opening proofs, [BMM+21, GMN22] for Groth16 proofs [Gro16], and [SCP+22] for vector commitment opening proofs. While our techniques can be applied to all these works to obtain similar results, it is beyond the scope of this work to present all of them in detail. Therefore, we will use aggregating KZG polynomial commitment opening proofs as one example to show the applicability of our technique.

Recap of KZG [KZG10]. Let $\mathbb{G}$ be a pairing-friendly group with generator g and pairing $e(\cdot,\cdot)$. In KZG polynomial commitment scheme, the SRS is $(g, g^{\tau}, \ldots, g^{\tau^D})$ for some random τ and a maximum degree D. The commitment to a polynomial $f(x) = a_0 + a_1 \cdot x + \cdots + a_d \cdot x^d$ is $\sigma = g^{f(\tau)}$, which can be computed by $\prod_{i=0}^{d} \left(g^{\tau^i}\right)^{a_i}$. To open the polynomial at x^*, one computes the quotient polynomial $Q(x) = \frac{f(x)-f(x^*)}{x-x^*}$. The opening proof is $\pi = g^{Q(\tau)}$. To verify the proof, one checks $e(\sigma, g) \stackrel{?}{=} e(\pi, g^{\tau}/g^{x^*})$.

It is well-known that one can batch KZG opening proof for different polynomials at the same location or the same polynomial at different locations. In this section, we focus on the case where the aggregator holds n opening proofs of *different* polynomials $f_1, \ldots, f_n$ at *different* locations $x = x_1, \ldots, x = x_n$. It wishes to aggregate these proofs into a succinct one proving all the statements simultaneously. Prior work [BDFG21] has shown how to publicly aggregate these proofs, where the verification time is $O(n^2)$.[31] Our scheme only requires a verification time of $O(n)$, but comes at a larger proof size.

A.1 The construction for aggregating KZG Opening Proofs

We use similar notations as described in Section 8. Our construction makes use of the following lemma, similar to Lemma 1.

Lemma 2 (More Generalized Sumcheck). *Let $A(x) = \sum_{i=1}^{n} a_i \cdot L_i(x)$, $B(x) = \sum_{i=1}^{n} b_i \cdot L_i(x)$, and $C(x) = \sum_{i=1}^{n} c_i \cdot L_i(x)$. It holds that*

$$A(x) \cdot B(x) \cdot C(x) = \frac{\sum_i a_i \cdot b_i \cdot c_i}{n} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x),$$

where Q_x is a polynomial with degree $\leqslant n-2$. As similar to Lemma 1, note that, given $A(x)$, $B(x)$, and $C(x)$, $Q_x(x)$ and $Q_Z(x)$ can be computed efficiently using FFT as

$$Q_x(x) = \left(D(x) - \frac{\sum_i a_i b_i c_i}{n}\right) \cdot x^{-1} \text{ and } Q_Z(x) = \Big(A(x) \cdot B(x) \cdot C(x) - D(x)\Big) \cdot Z(x)^{-1},$$

[31] One may use the inner product argument-based approach (similar to Snarkpack [GMN22]) to reduce this to $O(n)$. However, as far as we know, this is not explicitly written anywhere.

where $D(x)$ is the polynomial that interpolates $(a_1b_1c_1, \ldots, a_nb_nc_n)$, i.e., $D(x) = \sum_i a_ib_ic_i \cdot L_i(x)$.

Construction Sketch. Given the statements $\{\mathsf{Com}(f_i), x_i, y_i\}_{i=1}^n$ and proofs $\{\pi_i = \mathsf{Com}\,(Q_i)\}_{i=1}^n$ where $Q_i(x) = \frac{f_i(x)-y_i}{x-x_i}$. Our construction is based on the following idea. The aggregator will first commit to the vector of proofs $\pi_1, \ldots, \pi_n$. A random challenge v is picked, and the aggregator will prove to the verifier the random linear combination of

$$\alpha = \prod_i \left(e(\pi_i, g^\tau / g^{x_i})\right)^{v^i}.$$

The verifier will check if

$$\alpha \stackrel{?}{=} \prod_i \left(e(\mathsf{Com}(f_i)/g^{y_i}, g)\right)^{v^i}.$$

In more details, the proof aggregator does the following.

- Let $\Pi(x)$ be the polynomial interpolating $(\pi_1, \ldots, \pi_n)$. The aggregator generates $\mathsf{HCom}(\Pi)$ as the commitment to $\Pi(x)$.
- Using the random oracle with input $\mathsf{Com}(\Pi)$ to sample a random challenge v.
- Define the polynomial $T(x)$ that interpolates $(\tau - x_1, \tau - x_2, \ldots, \tau - x_n)$. Define the polynomial $V(x)$ that interpolates $(v, v^2, \ldots, v^n)$. Invoke Lemma 2 as

$$\Pi(x) \cdot T(x) \cdot V(x) = \frac{\sum_i Q_i(\tau) \cdot (\tau - x_i) \cdot v^i}{n} + Q_x(x) \cdot x + Q_Z(x) \cdot Z(x). \tag{4}$$

- The aggregator commits to $Q_x(x)$, $Q_Z(x)$. We note that these polynomials are encapsulated in the *target group*.
- A random challenge r is sampled by the random oracle. The aggregator gives an opening proof proving that the committed polynomials at $x = r$ evaluate to

$$g^{\Pi(r)}, g^{T(r)}, V(r), e(g,g)^{Q_x(r)}, e(g,g)^{Q_Z(r)}.$$

  Additionally, the aggregator computes and sends α, which should equal to

$$\alpha = \prod_i \left(e(\pi_i, g^\tau / g^{x_i})\right)^{v^i} = \prod_i \left(e(\mathsf{Com}(f_i)/g^{y_i}, g)\right)^{v^i}$$

To verify the proof, the verifier checks all the opening proofs. For the opening proof of the polynomial commitment related to the statements, i.e., $T(x)$ that interpolates $(\tau - x_i)$'s, the verifier needs to check the consistency of the first layer of the FRI openings with the statement that it holds. In particular, they should form a correct Reed-Solomon codeword uniquely defined by the statements. The verifier will check this using a parity check. In addition to verifying the opening proofs, the verifier also checks if Equation 4 holds at $x = r$.

Soundness. The soundness roughly is as follows. By the soundness of the polynomial commitment, there exists a polynomial $\Pi(x)$ (consistent with the opening proof) that encodes some proofs $(\pi_1, \ldots, \pi_n)$. If any one of the individual statements does not verify, by Schwartz-Zippel, with all but $\mathrm{poly}(\lambda)/|\mathbb{F}|$ probability, the merged statement using the challenge v will not verify. Finally, since Equation 4 holds at a random

location $x = r$, with all but $\text{poly}(\lambda)/|\mathbb{F}|$ probability, the polynomial identity holds, which proves that α is the correct computation of $\prod_i \left(e(\pi_i, g^\tau / g^{x_i})\right)^{v^i}$ based on the committed proofs π_i.

Efficiency. The aggregated proof size depends on the opening proof size of FRI, which is $O(\log^2 n)$. The aggregation time is dominated by FFT operations. Since the aggregator needs to do FFT on group elements, the aggregation time is $O(n \log n)$ group operations. The verification time is dominated by computing $\alpha = \prod_i \left(e(\pi_i, g^\tau / g^{x_i})\right)^{v^i}$, which takes $O(n)$ group operations.