Compute, but Verify: Efficient Multiparty Computation over Authenticated Inputs

Moumita Dutta[1], Chaya Ganesh[1], Sikhar Patranabis[2], and Nitin Singh[2]

[1] Indian Institute of Science
{moumitadutta,chaya}@iisc.ac.in
[2] IBM Research, India
sikharpatranabis@ibm.com,nitisin1@in.ibm.com

Abstract. Traditional notions of secure multiparty computation (MPC) allow mutually distrusting parties to jointly compute a function over their private inputs, but typically do not specify how these inputs are chosen. Motivated by real-world applications where corrupt inputs could adversely impact privacy and operational legitimacy, we consider a notion of *authenticated* MPC where the inputs are authenticated, e.g., signed using a digital signature by some certification authority. We propose a generic and efficient compiler that transforms any linear secret sharing based honest-majority MPC protocol into one with input authentication.

Our compiler incurs significantly lower computational costs and competitive communication overheads when compared to the best existing solutions, while entirely avoiding the (potentially expensive) protocol-specific techniques and pre-processing requirements that are inherent to these solutions. For n-party honest majority MPC protocols with abort security where each party has ℓ inputs, our compiler incurs $O(n \log \ell)$ communication overall and a computational overhead of $O(\ell)$ group exponentiations per party (the corresponding overheads for the most efficient existing solution are $O(n^2)$ and $O(\ell n)$). Finally, for a corruption threshold $t < n/3$, our compiler preserves the stronger identifiable abort security of the underlying MPC protocol. No existing solution for authenticated MPC achieves this regardless of the corruption threshold.

Along the way, we make several technical contributions that are of independent interest. This includes the notion of distributed proofs of knowledge and concrete realizations of the same for several relations of interest, such as proving knowledge of many popularly used digital signature schemes, and proving knowledge of opening of a Pedersen commitment.

1 Introduction

Secure multiparty computation (MPC) allows two or more parties to jointly compute a function f of their private inputs. The guarantees of such a protocol are privacy of the inputs and correctness of the output, even in the presence of some corrupt parties. Security definitions model the behavior of corrupt parties as either semi-honest (who follow the prescribed protocol, but might analyze the messages received in order to learn unauthorized information), or malicious (who arbitrarily deviate from the protocol).

Traditional security notions for MPC ensure the correctness of the output and privacy, that is, nothing is revealed beyond the output of the computation. However, no assurance is given about what input parties use in the protocol. The protocol does not specify how the parties choose their private inputs, irrespective of whether they follow the protocol or not. Parties may modify their "real" input affecting correctness and security, but this is outside the scope of MPC security and is allowed by security definitions. However, several applications are sensitive to "ill-formed" inputs; such inputs can either corrupt the output or reveal the output on arbitrary uncertified inputs which compromise privacy. Such attacks are of practical concern in applications of MPC in computation on genomic data [12]. As an example, if a set of individuals on a job portal wish to compute "industry average compensation" for their expertise and experience in a privacy preserving manner (like services provided by glassdoor), one would want them to input payslips that are authenticated by their employer (e.g., using digital signatures). Similarly, in applications of hospitals performing joint computations on patient data for treatment efficacy, it is desirable to ensure that the data used is signed by a regulatory authority.

The above examples illustrate that many real-world applications of MPC require the inputs used for computing the function to be *authentic*. For such applications, the guarantees provided by traditional MPC notions are clearly inadequate. A natural question that confronts us then is: *"Which inputs should be considered authentic? How do we ensure that authentic inputs are used in a secure computation?"*

Input Authenticity. In real life, data rarely originates in a "vacuum". Almost all of the data is vetted by a relevant authority such as universities for academic records, banks for financial transactions, accredited auditors for financial statements, several government bodies for individual attributes such as name, age, employment status, etc. In all such cases, the data is considered authentic if it has a suitable attestation from the relevant *certifying authority*. Since the certifying authority cannot be omnipresent to vouch for authenticity of the data, it enables individuals to claim and verify this attestation increasingly through *digital signatures*. More recently, several digital signature schemes such as [14,43,19] have been proposed which enable an individual to establish attestation by a certifying authority with minimal disclosure of attributes. Further the attestation can be established in an *unlinkable* manner, where several usages of the same credential cannot be linked.

Unfortunately, all of the above benefits, which allow authentic data to be used securely in the individual context are negated when computing securely over data from *multiple* data owners, if one adheres to the vanilla security guarantees of MPC protocols. For instance, consider an application such as secure collaboration for key performance indicators (KPI) where parties would like to collaboratively compute aggregate statistics on joint datasets [3]. Since such a collaboration is aimed at computing accurate industry-wide metrics, it is pertinent that participants supply "authentic" inputs to the computation. While MPC is a promising solution for applications requiring privacy-preserving collaborative computations, traditional MPC security notions are not sufficient for guaranteeing input authenticity in such applications.

In this paper, we make substantial progress to address the above shortcoming, by efficiently augmenting existing MPC protocols to additionally ensure that inputs have a valid attestation (in the form of a digital signature) from a relevant certifying authority. Moreover, we illustrate our solution with the BBS+ [14,5] and PS [43] signature schemes which efficiently support minimal disclosure features as mentioned before.

Why naïve solutions are not satisfactory. A straightforward way to achieve input authenticity is to run the MPC protocol on inputs that are signed by some certification authority. This can be achieved by having the protocol first verify the signature on the inputs, and if validated, proceed to compute the original functionality. In certain applications, authenticity could mean that inputs are expected to satisfy a certain predicate or property. This can be achieved by verifying that the inputs are consistent with global commitments, and then various properties can be proved about the committed value. Regardless of the particular notion of authenticity, MPC on certified inputs can be achieved in general by augmenting the function f to be computed with the verification function of a signature or a commitment scheme. However, signature and commitment verification typically involves hashing the message which is expensive in MPC, or expressing algebraic operations as arithmetic circuits which blows up the size of the circuit to be computed (see Table 1 for a more detailed overview).

Another approach is to have the certifying authority sign public commitments to the inputs, and then have the parties prove that their inputs are valid openings to the corresponding public commitments. Using Pedersen commitments, and customized zero-knowledge protocols, this approach can be more efficient than authenticating inside MPC. However, this approach does not satisfy the property of *unlinkability*, since same inputs used across different protocols can be linked when using a signed commitment. Unlinkability – ensuring that (same) inputs used by a party across different protocols cannot be linked – is an essential privacy requirement.

Our goal is to lift existing MPC protocols into ones ensuring that an additional predicate (such as possession of valid signature on inputs) is satisfied by the inputs; and we want to achieve this (i) without changing the underlying MPC protocol, (ii) without representing the predicate as a circuit, (iii) incurring communication overhead that is succinct in the size of the inputs (which are large for our applications), and (iv) maintaining unlinkability. This precludes prior approaches requiring the authentication relation to be expressed as a circuit [15,35], as well as the approach based on signed public commitments outlined above. We adopt an approach that works over the shares of the input as opposed to identifying the input via public commitments. At a high level, our solution is a *distributed proof of knowledge* of Pedersen commitment openings, which turns out to be as efficient (or more, with optimizations) as verifying signature on commitments, while additionally ensuring unlinkability (see Section 1.2 for additional discussions).

[3] Leading cloud providers have "clean room" offerings to enable such collaborations `https://docs.aws.amazon.com/clean-rooms/latest/userguide/what-is.html`

1.1 Our Contributions

In this work, we study *authenticated MPC* and propose a generic compiler to efficiently transform an MPC protocol into an MPC protocol with input authentication. Towards this goal, as a contribution of independent interest, we put forth a notion of distributed zero-knowledge proof of knowledge (DPoK).

Compressed Distributed Proofs of Knowledge. We consider a setting with multiple provers and a single verifier where the witness is secret shared among the provers. The verifier has as input an instance x, and each prover has as input a share w_i such that $(x, w) \in \mathcal{R}$ where $w = \mathsf{Reconstruct}(w_1, \ldots, w_n)$.

As discussed earlier, using generic MPC to achieve this is inefficient. Moreover, participants in our protocol communicate in restricted manner: (i) the provers do not communicate with each other, and (ii) the verifier communicates only via a broadcast channel and is public coin. Looking ahead, the use of only broadcast channels and public coins also facilitate *public verifiability*. In our authenticated MPC application, each party plays the role of the prover, and all other parties verify the corresponding proof. The prover's role itself is then distributed among all parties. Public verifiability implies that we can go from one verifier to many verifiers by using the Fiat-Shamir transform to non-interactively derive the verifier's messages using a random oracle (RO).

Our definition of distributed proof of knowledge (DPoK) is a natural distributed analogue of honest-verifier public coin protocols. In Section 3, we construct a distributed proof of knowledge for the discrete logarithm relation. We then show how to apply the compression technique from Attema et al. [3] to improve the communication complexity of our protocol from being linear to logarithmic in the size of the witness. Our techniques are modular, and we believe that they are more generally applicable to distributed sigma protocols for algebraic languages used in other applications.

Robust Complete DPoKs. The ideas outlined above will not prevent malicious provers from disrupting the protocol execution by using bad shares and causing abort. To tackle this, we put forth a notion of robustness which additionally provides tolerance against abort in the presence of $n/3$ malicious provers. That is, when the shares indeed reconstruct a valid witness, the protocol will lead the verifier to accept even if up to $n/3$ provers deviate from the protocol. To achieve such error-correction over messages "in exponents", we leverage results from low degree testing (Lemma 2) used in constructions of efficient zkSNARKs [1,9]. Informally, the results state that to check that a set of k sharings of messages $s_1, \ldots, s_k$ have not been tampered (by corrupt provers), it is sufficient to publicly reveal a suitably blinded linear combination of the above sharings. The deviant positions in the revealed sharing (from a consistent sharing) with overwhelming probability capture deviations across all the sharings. Note that, while achieving robust completeness is straightforward if we do not care about succinctness, the main technical novelty of our constructions is to achieve both properties simultaneously via low-degree testing.

Generalization to Threshold Linear Secret Sharing. Our techniques for obtaining DPoKs as discussed above generalize to any *threshold linear secret sharing* (TLSS) scheme. In particular, for robust complete DPoKs, we characterize the robustness threshold in terms of the minimum distance of the linear code associated to the TLSS scheme. Concrete bounds are obtained for Shamir Secret Sharing and Replicated Secret Sharing schemes (see Appendix A for details).

Authenticated MPC. We consider a notion of input authenticity for MPC where the inputs possess a valid signature from a certification authority. This is standard in applications where a publicly known certifying authority (external to the MPC protocol) signs an input to certify that the input satisfies certain properties[4]. Informally, we construct an n-party protocol in the honest majority setting realizing the following authenticated MPC functionality $\mathcal{F}$.

- For each $i \in [n]$, party-i sends its input x_i and the associated signature σ_i on x_i to the functionality $\mathcal{F}$.
- The functionality $\mathcal{F}$ checks that σ_i is a valid signature on x_i for all $i \in [n]$. Let $S \subset [n]$ be the subset of parties that produced invalid signatures on their inputs. If S is non-empty, the functionality $\mathcal{F}$ outputs $(\texttt{abort}, S)$ to all the parties. Otherwise it computes $y = f(x_1, \ldots, x_n)$ and outputs y to all parties.

[4] Our techniques extend to other notions of authenticity like proving that the inputs open publicly known commitments.

Compiler for Authenticated MPC. We propose a generic compiler that transforms any (threshold linear) secret-sharing based honest-majority MPC protocol Π based on TLSS scheme into a protocol Π' realizing the above *authenticated* MPC functionality. In Section 6, we describe our compiler for maliciously secure honest-majority MPC protocols based on Shamir secret sharing, though it can be generalized to any TLSS-based protocol, using our generalized DPoK protocols from Appendix A. The compiled protocol Π' inherits the security of Π as long as the inputs are authentic(by definition, we abort if this is not the case)[5]. If Π guarantees security with identifiable abort, then the same holds for Π'; if Π achieves guaranteed output delivery, then so does Π' (albeit for $t < n/3$ by crucially using the *robustness* property of our proposed DPoK protocols).

Generality. We note that our approach works in general for: (a) any (threshold linear) secret-sharing based MPC protocol, and (b) any signature scheme such that the associated proof of knowledge can be modeled as a proof of knowledge of the opening of a Pedersen commitment. We use a broadcast channel in our protocols. For broadcasting ℓ bits among n parties, state-of-the-art broadcast protocols incur a communication complexity of $O(\ell n)$ when $\ell >> n$ [11,33]. In our applications, as we discuss next, we indeed expect ℓ to be $\Omega(\lambda n)$ where λ is a security parameter.

We present specific instances of this general approach for signature schemes that are algebraically compatible while also being popular candidates for applications such as verifiable credentials for self-sovereign digital identity: these include Camenisch-Lysyanskaya (CL) signatures [18], Boneh-Boyen-Shaham (BBS) signatures (and variants such as BBS+) [14,5,17][6], and Pointcheval-Sanders (PS) signatures [43]. These are signature schemes that support efficient zero-knowledge proofs of knowledge of a valid message-signature pair. We ue BBS+ signatures to describe and prototype-implement our compiler. Our compiler incurs negligible communication overhead over the original MPC protocol. This is demonstrated by our implementation results which we present in Section 7.

We also illustrate the generality of our techniques by describing an alternative way of instantiating our compiler for authenticated MPC based on PS signatures in Appendix C. We believe that our techniques would naturally extend to other structured algebraic signatures such as CL signatures [18].
DPoK vs Authenticated Secret-sharing. Certain prior works [2,48] proposed using authenticated secret-sharing in order to certify inputs to MPC. However, authenticated secret-sharing only provides stand-alone guarantee about the shares themselves, and additional techniques would be needed to ensure that malicious parties actually use these authenticated shares in the execution of the actual MPC protocol (the details of such techniques are not specified completely in prior works [2,48]). We address this in our DPoK-based compiler, which realizes an ideal functionality that ties input authentication into the underlying MPC, thus preventing malicious parties from using different inputs as compared to the authenticated ones. Using DPoKs also offers more flexibility in proving a wider class of expressive predicates (beyond just input authentication) over secret-shared inputs. For instance, parties can provide a proof that committed input satisfies certain conditions by using a DPoK to prove that the shares reconstruct to the value in the commitment. If a different application requires new/additional properties to be checked, one need not go back to the certifying authority. Instead one can prove knowledge of the signature in our current framework, where we: (i) publish a commitment, (ii) prove the consistency of secret-shared input with a commitment using a DPoK, and finally (iii) give proofs that the committed value satisfies a certain predicate. Hence, our solution is applicable to a wide range of applications with diverse proof requirements (e.g., federated learning), while avoiding the need to involve the certification authority every time.

Prototype Implementation and Performance Overheads. Consider again the motivating application of authenticated MPC outlined earlier, where n shipping companies with private datasets wish to securely compute aggregate statistics on some subset of their combined data. Note that this is indeed an application where the number of inputs of each party is much larger than the number of parties involved in the protocol, and is consistent with our assumption above. Specifically, we consider each dataset $D_i = (C_i, S_i)$ to be partitioned into k *categorical* columns C_i and ℓ numeric columns D_i. A sample query specifies $\{(j, v_j)\}_{j \in J}$ for $J \subset [k]$. The goal is to compute means of numeric columns on the subset of rows satisfying the selection predicate $C[j] = v_j$ for $j \in J$, i.e the subset of rows with specified values of some categorical features. We also assume a non-participating certifying entity $\mathcal{T}$

[5] In some applications, it is acceptable to continue computation on default inputs instead of aborting when authentication fails.

[6] There are standardization efforts for using BBS+ signatures in verifiable credentials for Web 3.0, leading to a recent RFC draft [40].

(e.g. a financial auditor) which independently verifies the correctness of sales data reported by different organizations and issues a digital signature to attest the same. We present a prototype implementation of our BBS+ based authenticated MPC protocol in the above setting. In particular, we implement the BBS+ based instance of our compiler and use it to transform an implementation of a native MPC (instantiated via MP-SPDZ [25,39,37]) into an authenticated MPC. The low overhead incurred by our protocol to achieve authentication on top of native MPC is illustrated in Table 1. For comparison, we also show the overhead incurred by a naïve way of authenticating inside the MPC (showing consistency of inputs with a public digest). The overheads are substantial even when an MPC-friendly hash function like MiMC is used.

# Parties	Vanilla MPC	Auth MPC with MiMC Hash	DPoK Overhead
3	33s/8437 MB	273s/13979 MB	5.7s/14.4 KB
5	125s/43823 MB	1369s/14498 MB	6.2s/30 KB
7	386.2s/127057 MB	3645.33s/207427 MB	8.2s/52 KB

Table 1: Comparison of our DPoK-based approach for MPC input authentication with the naïve approach of validating BBS+ signatures (which involves computing MiMC hashes) inside MPC. These results correspond to datasets of size 500×10 in the KPI application.

1.2 Related Work

We summarize some relevant related work, and compare our framework with prior approaches.

Certified Inputs. The works of [36,6,48] achieve input validation for the special case of *two*-party computation using garbled circuit (GC) based techniques. The work of [13] constructs MPC with certified inputs, albeit using techniques that are specific to certain MPC protocols [24,23]. A recent work [2] develops techniques for computing bilinear pairings over secret shared data, which aims to enable signature verification inside MPC for the PS signature scheme [43]. Both works [13,2] emulates a functionality similar to authenticated secret-sharing protocol, where shares of an input certified by some certification authority are provided. While the goal of authenticated MPC has been studied, none of these prior works formalize this as an ideal functionality and prove that the proposed constructions realize this. For instance, consider the scenario where a malicious party receives the shares of a certified input held by an honest party, which is done via an authenticated secret-sharing protocol, however while running the MPC itself it chooses to not use the shares received during the previously run authenticated secret-sharing protocol and uses an arbitrarily chosen share instead. The definitions in [2,48] fails to safeguard against such an attack.

To be precise, protocol $\Pi_{\mathsf{CertInput}}$ in [2] (Section 5.1) emulates the authenticated secret-sharing, such that at the end of the protocol, if an input corresponds to a valid signature, the shares of that input is available to every party. This protocol first secret-shares the input, then using the shares held by everyone as input invokes another protocol Π_{Verify} to ascertain if the shares obtained in the previous phase corresponds to an input for which there is a valid signature. However, note that only Step 3 of Π_{Verify} considers the shares of the input, which need not be the shares used for running the MPC. Hence, the input may be authenticated using different shares than what is used for the MPC itself, which does not provide the envisioned authenticated MPC. [48] also follows a similar template of authenticated secret-sharing. Their techniques consider two specific MPC protocols [24,23] for input certification, and provides detailed analysis in their Section 6 (Using Certified Inputs in Secure Computation). Theorem 8 for input certification in [24] and Theorem 10 for input certification in [23] ensure that a malicious prover cannot feed an input which does not correspond to the valid signature, however it is unclear how it is done since their invoked protocol Batch using CL signatures and Elgamal signatures do not specify that the commitment to the inputs used for the batch verification of signatures are same as the inputs used for the rest of the proof of knowledge statements. The above uncertainties with respect to "linking" the shares of the input used in MPC with the shares used during input certification arise

due to lack of a functionality capturing this. We recognize the need for a definition to capture the consistency of shares of input used in authentication and the MPC, which is missing in prior works, and provide an ideal functionality to ensure the same, together with a construction satisfying this ideal functionality. This precludes the possibility of using different inputs for certification and MPC by enforcing that the honest party shares must completely determine the reconstructed input which is being authenticated. While this restriction would also ensure that the same holds for constructions in [2,48] as well, this observation has not been specified in either of the works.

Our compiler uses efficient compressed distributed sigma protocol proofs for signature verification instead of verifying signatures inside the MPC protocol, and differs from both [13] and [2] in terms of techniques used and properties achieved. In particular, our compiler is modular, fully generic (works in a plug-and-play manner with any threshold linear secret sharing based MPC protocol), and avoids the (potentially expensive) protocol-specific techniques and pre-processing requirements that are inherent to [13,2]. Our compiler also enables stronger security guarantees as compared to abort security, namely identifiable abort (and even full security/guaranteed output delivery in certain cases), which neither [13] nor [2] achieves.

Distributed Zero-knowledge. Various notions of distributed zero-knowledge have appeared in literature. The notion of distributed interactive proofs appeared in [42], in the context of relations describing the verification of signatures, where the signature is public and the secret key is shared. The notion in [47] considers a distributed prover in order to improve prover efficiency, but the witness is still held by one entity. In Feta [7], the distributed notion is a generalization of designated verifier to the threshold setting where a set of verifiers jointly verify the correctness of the proof. Prio [21] proposes secret shared non-interactive proofs where again, there is a single prover and many verifiers.

Our formulation of distributed proofs of knowledge also differs from recent works on distributed zkSNARKs [44,41,26], where the focus is on jointly computing a non-interactive publicly verifiable proof (with specific focus on Groth16 [34], Plonk [30] and Marlin [20]). Their constructions require additional interaction among the workers over private channels; on the other hand, we consider distributed proofs of knowledge where all interaction with the verifier takes place over a public broadcast channel. We also study the notion of *robust completeness* that guarantees that the protocol runs to completion even in the presence of malicious behavior, which was not considered in prior works.

Proofs on Secret-shared Data. Notions of zero-knowledge proofs on distributed data is explored in recent works [15,35]. The former work proposes the abstraction of a fully linear PCP (FLPCP) where each verifier only has access to a share of the statement, and the latter work is based on MPC-in-the-head paradigm. We provide below a high level comparison of our work with aforementioned works in terms of definition, applications, and efficiency. *Efficiency:* The techniques of [15,35] assume the relation to be represented as an arithmetic circuit in order to achieve a DPoK. However, the languages we consider are algebraic and expressing them as a circuit is prohibitively expensive. Consider modeling an algebraic relation like knowledge of discrete logarithm in the distributed setting. If the statement were distributed, one has to materialize the circuit representing the relation (so that intermediate values act as witness). In the natural representation of the discrete log relation $\mathcal{R}^{\mathrm{DL}}$ as pairs $((g, z), s)$ satisfying $g^s = z$, s is not part of the statement. The relation can be transformed into a circuit-based relation $C((g^*, z^*, s^*), w)$ where (g^*, z^*, s^*) denotes the transformed statement involving (g, z, s), and w denotes the witness required to check the discrete-log relation in an arithmetic circuit. This introduces obvious inefficiencies since modular exponentiation has circuit size that is roughly cubic in the bit size of the modulus.

In general, the languages we consider are algebraic in nature and expressing them as a circuit is prohibitively expensive. Our observation is that algebraic relations like discrete log is naturally distributed witness relation. A public statement and shared witness is better suited for algebraic relations, and our distributed zero-knowledge definition captures such natural relations. Since the focus of our work is on concrete efficiency (prover overhead, communication overhead), we take advantage of the algebraic nature of the relation to design concretely efficient distributed sigma protocols by modeling the witness as being distributed and statement being public. In this approach, we expect rich classes of protocols (compressed sigma protocols, Bulletproofs etc that avoid circuit representation for several useful relations) to be amenable to be distributed under our definition. In addition, [15] provides sublinear communication only for special circuits (like degree 2) and the circuits of interest for us are unlikely to have this structure. *Robustness:* We note that [15] does not consider the robustness property. We put forth the robustness notion that guarantees that the protocol runs to completion even in the presence of malicious parties (when the prover is honest). This property is indeed important for

our applications, as this means that the compiled authenticated MPC protocol can identify malicious parties in the authentication stage. *Applications:* The motivating application for [15] is compiling passive security to active security, and therefore the statements that show up – like the next message function of the protocol -– have a low degree circuit representation. We consider the authenticated input application where our relations of interest are algebraic in nature (e.g. verification of an algebraic signature scheme) and admit efficient sigma protocols.

Public Commitments and Anonymity Sets. In the discussion of possible naïve solutions, we stated an alternative approach for achieving authenticated MPC based on having the certifying authority sign commitments to the private inputs of the parties, and then having the parties prove during the MPC protocol that their inputs indeed open the public commitments. However, as discussed earlier, this approach trivially violates the desired property of *unlinkability*, since one can link the usage of the same input across different protocol executions from the public commitments. A possible fix is to use *anonymity sets*: all commitments to the inputs are made publicly available, and instead of explicitly identifying which commitment is linked with each input, the party provides a zero knowledge proof of knowledge of an opening of one of the several signed commitments, along with a proof of membership of the commitment in the public set.

While this is a plausible solution, we believe that full unlinkability (as modeled implicitly by our ideal functionality and realized by our proposed solution) is a better solution that anonymity. First of all, the anonymity set needs to be large enough for any reasonable notion of unlinkability to hold; however, this is an issue as the size of the statement to prove increases with the size of the set, leading to additional overheads for the proof of knowledge. Additionally, one has to prove that a commitment used is a member of the accumulated set, requiring additional proofs of membership. Finally, in practical applications, it is unclear which entity will create and maintain this set accumulator: for instance, if a new data set to be used as input for a computation is signed by an authority, it must be added to the accumulator. This leads to additional overheads for accumulator maintenance.

1.3 Technical Overview

We begin by outlining ideas to distibute a Sigma protocol for proving knowledge of discrete logarithm of a public group element. This relation will be at the core of expressive algebraic relations that we will consider later.

Distributed Sigma protocol. Let $\mathbb{G}$ be a group of prime order p. Given $x \in \mathbb{G}$, consider Schnorr's protocol for proving knowledge of discrete logarithm w such that $x = g^w$ for some generator g. Let $(\mathcal{P}^1, \mathcal{P}^2, \mathcal{V})$ be the protocol where we denote by $\mathcal{P}^1$ and $\mathcal{P}^2$ the algorithms that compute, the prover's first message $a = g^\alpha$ for random $\alpha \in \mathbb{F}_p$, and the prover's last message (response) $z = \alpha + cw$, respectively, where c is the challenge from the space $\{0,1\}^l$ for some length l. Let $\mathcal{V}$ be the algorithm that takes x, transcript $\tau = (a, c, z)$ and accepts iff $g^z = ax^c$.

Now, in order to *distribute* this Sigma protocol, we begin by assuming n provers $\mathcal{P}_i$ who each hold a share w_i such that $w = w_1 + \cdots + w_n \pmod p$. Now, each prover runs Σ with their respective shares in parallel[7]. That is, $\mathcal{P}_i$ runs $\mathcal{P}^1$, broadcasts $a_i = g^{\alpha_i}$, receives challenge c from $\mathcal{V}$, and runs $\mathcal{P}^2$ and broadcasts z_i. The transcript is $\tau = (a_1, \ldots, a_n, c, z_1, \ldots, z_n)$, and the verifier accepts iff $g^{\Sigma z_i} = \prod a_i x^c = \prod_i a_i x^c$. This holds since $g^{\Sigma z_i} = g^{\Sigma(\alpha_i + cw_i)} = \prod_i a_i x^c$.

This idea generalizes to any linear secret sharing scheme, and also extends to other relations. For instance, to prove knowledge of representation of a vector of discrete logarithms with respect to public generators. In our final construction we use additional ideas like randomization of the first message of each $\mathcal{P}_i$ via a sharing of 0 in order to ensure zero-knowledge. This distributed Sigma protocol has linear communication complexity. To achieve succinctness, we instead use as a starting point a compressed sigma protocol [3] in order to achieve a distributed protocol with logarithmic communication complexity.

Robust Completeness. While the ideas described above result in protocols that are zero-knowledge and sound against a malicious adversary controlling up to t parties, completeness is guaranteed only

[7] This is a simplified description; in our actual protocol Π_{dlog} (Section 3.2), there are no parallel sessions, each instance uses a random share, ensuring that we do not reuse the shares, and in the FS-compiled version $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ (Section 4.2), parties send non-interactive proofs instead of sending the first-messages separately in parallel. We note that ROS attacks [10] in the context of concurrent signatures are therefore inapplicable in our setting. See also Section 6.2 for a more detailed discussion.

if all the provers follow the protocol. However, in the distributed setting, a stronger, but natural notion is a *robust* completeness property where completeness holds as long as the shares reconstruct a valid witness, even if some provers are malicious. The main technical challenge in achieving robust completeness for a distributed proof is to retain succinctness. Our key technical novelty is to achieve both robustness and succinctness *simultaneously* via ideas from low-degree testing. We achieve this by identifying and discarding corrupt shares. At a high level, the provers commit to their shares and then reveal a certain linear form determined by the challenge over their shares. Given a challenge $\boldsymbol{c} \in \mathbb{F}_p^m$, each $\mathcal{P}_i$ broadcasts $z_i = \langle \boldsymbol{c}, \mathbf{w}_i \rangle$. In the honest case, these opened linear forms are expected to be a sharing of the same linear form on the reconstructed witness: $\mathbf{z} = (z_1, \ldots, z_n)$ recombine to z where $z = \langle \boldsymbol{c}, \mathbf{w} \rangle$. The verifier error-corrects the received $\mathbf{z}'$ to the nearest codeword, and identifies the erroneous positions. By assumption our corruption threshold is smaller than half the minimum distance of the code, so the erroneous positions clearly come from corrupt provers. Can some corrupt provers strategically introduce errors in individual shares so that they "cancel out" in the inner product with $\mathbf{c}$? We lean on coding theoretic result (Lemma 2) for linear codes to claim that such a prover only succeeds with negligible probability. Finally, having identified the corrupt messages, we can reconstruct the claimed commitment in the exponent using commitments of honest shares (now identified). We need more details around this core idea to ensure the protocol is zero-knowledge.

A Generic Compiler. In order to construct an authenticated MPC protocol, our choice of signature scheme (and commitment scheme) are such that the verification can be cast as a relation for which we can construct a distributed protocol. The BBS signature scheme [14], the PS signature scheme [43] and Pedersen commitment [42] are some candidates for which our distributed protocol can be instantiated. Our compiler reuses the sharing that is already done as part of an honest-majority MPC protocol. Before proceeding with computation on the shares, the distributed zero-knowledge proof is invoked to verify authenticity, and then the rest of the MPC protocol proceeds. Since the shares of the witness come from a party in the MPC protocol, our robustness property guarantees that if the dealer is honest (that is, a valid witness was shared), then even if some parties acting as provers are dishonest, the authenticity proof goes through. We also introduce a modified formulation of proof of knowledge of BBS signatures (Section 5) and proof of knowledge of PS signatures (Section C), which leads to vastly more efficient distributed protocols.

We also note that, while we rely on broadcast for our protocols, all relevant related work on FLPCP [15] and previous works on authenticated MPC [13,2,35] also make use of a broadcast channel. A broadcast channel is not a limitation, and can be implemented using point-to-point channels. In the setting where the number of parties is not too large (as in the applications we consider), the communication overhead to realize broadcast is not prohibitive.

2 Preliminaries

Notation. We write $x \leftarrow_R \chi$ to represent that an element x is sampled uniformly at random from a set/distribution $\mathcal{X}$. The output x of a deterministic algorithm $\mathcal{A}$ is denoted by $x = \mathcal{A}$ and the output x' of a randomized algorithm $\mathcal{A}'$ is denoted by $x' \leftarrow_R \mathcal{A}'$. For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$. For $a, b \in \mathbb{N}$ such that $a, b \geq 1$, we denote by $[a, b]$ the set of integers lying between a and b (both inclusive). We refer to $\lambda \in \mathbb{N}$ as the security parameter, and denote by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified) polynomial function and negligible function in λ, respectively. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large.

Let $\mathbb{G}$ be a group and $\mathbb{F}_p$ denote the field of prime order p. We use boldface to denote vectors. Let $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n$, then $\mathbf{g}^{\mathbf{x}}$ is defined by $\mathbf{g}^{\mathbf{x}} = g_1^{x_1} \cdots g_n^{x_n}$. For $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\mathbf{h} = (h_1, \ldots, h_n) \in \mathbb{G}^n$, $\mathbf{g} \circ \mathbf{h}$ denotes component-wise multiplication, and is defined by $\mathbf{g} \circ \mathbf{h} = (g_1 h_1, \ldots, g_n h_n)$. For $\mathbf{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n$, $\mathbf{g}_L$ (similarly, $\mathbf{x}_L$) denotes the left half of the vector $\mathbf{g}(\mathbf{x})$ and $\mathbf{g}_R(\mathbf{x}_R)$ denotes the right half, such that $\mathbf{g} = \mathbf{g}_L \| \mathbf{g}_R$ and $\mathbf{x} = \mathbf{x}_L \| \mathbf{x}_R$.

2.1 Threshold Secret Sharing

For ease of exposition we define a special case of *threshold linear secret sharing* scheme below. For concreteness, the reader may assume a (t, n) Shamir Secret Sharing. The more general definition appears in Appendix A.

Definition 1 (Threshold Secret Sharing). *A (t, n) threshold secret sharing over finite field $\mathbb{F}$ consists of algorithms* $(\mathsf{Share}, \mathsf{Reconstruct})$ *as described below:*

- Share *is a randomized algorithm that on input $s \in \mathbb{F}$ samples a vector $(s_1, \ldots, s_n) \in \mathbb{F}^n$, which we denote as $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$.*
- $\mathsf{Reconstruct}$ *is a deterministic algorithm that takes a set $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| \geq t$, a vector $(s_1, \ldots, s_{|\mathcal{I}|})$ and outputs $s = \mathsf{Reconstruct}((s_1, \ldots, s_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$. We will often omit the argument $\mathcal{I}$ when it is clear from the context.*

A threshold secret sharing scheme satisfies the following properties:

- **Correctness***: For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q > t$, we have $\mathsf{Reconstruct}((s_{i_1}, \ldots, s_{i_q}), \mathcal{I}) = s$.*
- **Privacy***: For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q \leq t$, the tuple $(s_{i_1}, \ldots, s_{i_q})$ is information-theoretically independent of s.*

A concrete (t, n) sharing scheme over a finite field $\mathbb{F}$, known as the Shamir Secret Sharing is realized by choosing a set of distinct points $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_n\}$ in $\mathbb{F} \backslash \{0\}$. Then given $s \in \mathbb{F}$, the Share algorithm uniformly samples a polynomial p of degree at most t such that $p(0) = s$ and outputs $(p(\eta_1), \ldots, p(\eta_n))$ as the shares. The $\mathsf{Reconstruct}$ algorithm essentially reconstructs the value $s = p(0)$ using Lagrangian interpolation. We canonically extend the Share and $\mathsf{Reconstruct}$ algorithms to vectors by applying them component-wise.

Definition 2 (Linear Code). *An $[n, k, d]$-linear code $\mathcal{L}$ over field $\mathbb{F}$ is a k-dimensional subspace of $\mathbb{F}^n$ such that $d = \min\{\Delta(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{L}, \mathbf{x} \neq \mathbf{y}\}$. Here Δ denotes the hamming distance between two vectors.*

We say that an $m \times n$ matrix $\mathbf{P} \in \mathcal{L}^m$ if each row of $\mathbf{P}$ is a vector in $\mathcal{L}$. We also overload the distance function Δ over matrices; for matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{F}^{m \times n}$, we define $\Delta(\mathbf{P}, \mathbf{Q})$ to be the number of columns in which $\mathbf{P}$ and $\mathbf{Q}$ differ. For a matrix $\mathbf{P} \in \mathbb{F}^{m \times n}$ and an $[n, k, d]$ linear code $\mathcal{L}$ over $\mathbb{F}$, we define $\Delta(\mathbf{P}, \mathcal{L}^m)$ to be minimum value of $\Delta(\mathbf{P}, \mathbf{Q})$ where $\mathbf{Q} \in \mathcal{L}^m$.

Definition 3 (Reed Solomon code). *For any finite field $\mathbb{F}$, any n-length vector $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n) \in \mathbb{F}^n$ of distinct elements of $\mathbb{F}$ and integer $k < n$, the* Reed Solomon Code *$\mathcal{RS}_{n,k,\boldsymbol{\eta}}$ is an $[n, k, n-k+1]$ linear code consisting of vectors $\big(p(\eta_1), \ldots, p(\eta_n)\big)$ where p is a polynomial of degree at most $k-1$ over $\mathbb{F}$.*

We note that shares output by (t, n) Shamir secret sharing are vectors in $[n, t+1, n-t]$ Reed Solomon code. We can leverage tests for membership of a vector in a linear code (based on parity-check matrix) to check if a set of shares $\{s_i\}_{i \in \mathcal{H}}$ for $\mathcal{H} \subseteq [n]$ and $|\mathcal{H}| > t$ uniquely determine a shared value s for Shamir Secret Sharing scheme. Below, we formalise the notion of consistent shares and state a lemma to check such shares. In the interest of space, we directly state the results for general $m \in \mathbb{N}$, i.e. when vectors $\mathbf{s} \in \mathbb{F}^m$ are shared.

Definition 4 (Consistent Shares). *Let $\mathcal{L}$ be the linear code determined by a (t, n) Shamir secret sharing scheme over finite field $\mathbb{F}$. For $m \in \mathbb{N}$, we call a set of shares $\{\mathbf{s}_i\}_{i \in \mathcal{H}}$ for $\mathcal{H} \subseteq [n]$ with $|\mathcal{H}| \geq t+1$ to be* $\mathcal{L}^m$-consistent *if there exists $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathcal{L}^m$ such that $\mathbf{s}_i = \mathbf{v}_i$ for $i \in \mathcal{H}$. In this case $\mathbf{s} = \mathsf{Reconstruct}(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathbb{F}^m$ is the unique shared value determined by the shares $\{\mathbf{s}_i\}_{i \in \mathcal{H}}$.*

We define the predicate $\mathsf{Consistent} : \mathbb{F}^{\mathcal{H}+1} \to \{0, 1\}$ *as*

$$\mathsf{Consistent}(\{\mathbf{s}_i\}_{i \in \mathcal{H}}, \mathbf{s}) = \begin{cases} 1, & |\mathcal{H}| \leq t \\ 1, & |\mathcal{H}| > t \wedge \{\mathbf{s}_i\}_{i \in \mathcal{H}} \text{ is } \mathcal{L}^m\text{-consistent} \\ & \quad \bigwedge \mathsf{Reconstruct}(\{\mathbf{s}_i\}_{i \in \mathcal{H}}) = \mathbf{s} \\ 0, & \text{otherwise.} \end{cases}$$

We use this $\mathsf{Consistent}(.)$ predicate to determine if a vector $\mathbf{s}$ can be a possible candidate which could have been used to generate the set of shares held by the honest parties $\{\mathbf{s}_i\}_{i \in \mathcal{H}}$.

Lemma 1. *Let $\mathcal{L}$ be the linear code determined by a (t, n) Shamir secret sharing scheme over finite field $\mathbb{F}$. Then for $m \in \mathbb{N}$ and all $\mathcal{H} \subseteq [n]$ with $q = |\mathcal{H}| \geq t + 1$, there exists $q \times (n - t)$ matrix $\mathbf{H}_{\mathcal{H}}\mathcal{H}$ over $\mathbb{F}$ such that shares $\{\mathbf{s}_i\}_{i\in\mathcal{H}}$ are $\mathcal{L}^m$-consistent and determine the value $\mathbf{s} \in \mathbb{F}^m$ if and only if $\mathbf{X}\mathbf{H}_{\mathcal{H}} = (\mathbf{s}, \mathbf{0}^{n-t-1})$ where $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_q)$ is some canonical ordering of $\{\mathbf{s}_i\}_{i\in\mathcal{H}}$.*

Proof. We sketch the proof. For a matrix $\mathbf{P} \in \mathcal{L}^m$, we have $\mathbf{PH} = \mathbf{0}^{n-t-1}$ where $\mathbf{H}$ is the parity check matrix for the $[n, t+1, n-t]$ code $\mathcal{L}$. Now for $\mathcal{H} \subseteq [n]$ with $|\mathcal{H}| \geq t+1$, and matrix $\mathbf{X}$ determined by $\mathcal{L}^m$-consistent shares $(\mathbf{s}_i)_{i\in\mathcal{H}}$, there exists a matrix $\mathbf{T}_{\mathcal{H}}$ such that $\mathbf{X}\mathbf{T}_{\mathcal{H}} \in \mathcal{L}^m$, and hence $\mathbf{X}\mathbf{T}_{\mathcal{H}}\mathbf{H} = \mathbf{0}^{n-t-1}$. Thus for $\mathbf{H}_{\mathcal{H}} = [\mathbf{k}, \mathbf{T}_{\mathcal{H}}\mathbf{H}]$ where $\mathbf{k}$ is the column of reconstruction coefficients for the set $\mathcal{H}$, we have $\mathbf{X}\mathbf{H}_{\mathcal{H}} = (\mathbf{s}, \mathbf{0}^{n-t-1})$.

The following coding theoretic result is used to identify malicious behaviour in the distributed proof of knowledge protocol in Section 3.2. It has been previously used in construction of zero knowledge proofs in the interactive oracle setting (e.g [1,9]), to check that the oracle represents "low degree polynomials".

Lemma 2 ([8], Theorem 1.2). *Let $\mathcal{L}$ be an $[n, k, d]$-linear code over finite field $\mathbb{F}$ and let $\mathbf{S}$ be an $m \times n$ matrix over $\mathbb{F}$. Let $e = \Delta(\mathbf{S}, \mathcal{L}^m)$ be such that $e < d/2$. Then for any codeword $\mathbf{r} \in \mathcal{L}$, and $\boldsymbol{\gamma}$ sampled uniformly from $\mathbb{F}^m$, we have $\Delta(\mathbf{r} + \boldsymbol{\gamma}^T\mathbf{S}, \mathcal{L}) = e$ with probability at least $1 - n/|\mathbb{F}|$. Furthermore, if E denotes the column indices where $\mathbf{S}$ differs from the nearest matrix $\mathbf{Q}$ in $\mathcal{L}^m$, with probability $1 - n/|\mathbb{F}|$ over choice of $\boldsymbol{\gamma}$, the vector $\mathbf{r} + \boldsymbol{\gamma}^T\mathbf{S}$ differs from the closest codeword $\mathbf{v} \in \mathcal{L}$ at precisely the positions in E.*

2.2 Arguments of Knowledge

Interactive Arguments. Let $\mathcal{R}$ be a NP-relation and $\mathcal{L}$ be the corresponding NP-language, where $\mathcal{L} = \{x : \exists\ w \text{ such that } (x, w) \in \mathcal{R}\}$. Here, x is called an *instance or statement* and w is called a *witness*. An *interactive argument system* consists of a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$. $\mathcal{P}$, known as the prover algorithm, takes as input an instance $x \in \mathcal{L}$ and its corresponding witness w, and $\mathcal{V}$, known as the verifier algorithm, takes as input an instance x. Given a public instance x, the prover $\mathcal{P}$, convinces the verifier $\mathcal{V}$, that $x \in \mathcal{L}$. At the end of the protocol, based on whether the verifier is convinced by the prover's claim, $\mathcal{V}$ outputs a decision bit. A stronger *argument of knowledge*[8] property says that if the verifier is convinced, then the prover knows a witness w such that $(x, w) \in \mathcal{R}$.

Honest-Verifier Zero-Knowledge and Special-Soundness. A protocol is said to be *honest-verifier zero-knowledge* (HVZK) if the transcript of messages resulting from a run of the protocol can be simulated by an efficient algorithm without knowledge of the witness. A protocol is said to have *k-special-soundness*, if given k accepting transcripts, an extractor algorithm can output a w' such that $(x, w') \in \mathcal{R}$. Furthermore, a protocol is said to have *$(k_1, \ldots, k_\mu)$-special-soundness* [16], if given a tree of $\prod_{i=1}^{\mu} k_i$ accepting transcripts, the extractor can extract a valid witness. Here, each vertex in the tree of $\prod_{i=1}^{\mu} k_i$ accepting transcripts corresponds to the prover's messages and each edge in the tree corresponds the verifier's challenge, and each root-to-leaf path is a transcript. An interactive protocol is said to be *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments using the Fiat-Shamir [28] heuristic by deriving the verifier's messages as the output of a Random Oracle. In this work, we consider public-coin protocols.

2.3 BBS+ Signatures and PoK for BBS

In this section, we recall the BBS+ signature scheme [14,40,17], and its proof of knowledge. We use the variant of BBS+ signatures and the proof of knowledge from [17], which is the currently adopted variant in the IETF standard for verifiable crendentials [40]. Later, we also describe a slight variant of the BBS+ proof of knowledge from [17], which leads to corresponding distributed proofs with better amortized complexity (i.e, when several DPoKs are required at a time).

Definition 5 (BBS+ Signature Scheme [14,40]). *The BBS+ Signature Scheme to sign a message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *described as follows :*

[8] We sometimes use *proof* and *argument* interchangeably, but we are only concerned with arguments (proofs with computational soundness) in this paper.

– $\mathsf{Setup}(1^\lambda)$ *: For security parameter* λ*, this algorithm outputs groups* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *of prime order* p*, with an efficient bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *as part of the public parameters* pp*, along with* g_1 *and* g_2*, which are the generators of groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *respectively.*
– $\mathsf{KeyGen}(\mathsf{pp})$ *: This algorithm samples* $(h_0, \ldots, h_\ell) \leftarrow_R \mathbb{G}_1^{\ell+1}$ *and* $x \leftarrow_R \mathbb{F}_p^*$*, computes* $w = g_2^x$ *and outputs* $(\mathsf{sk}, \mathsf{pk})$*, where* $\mathsf{sk} = x$ *and* $\mathsf{pk} = (g_1, w, h_0, \ldots, h_\ell)$*.*
– $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: This algorithm samples* $\beta, s \leftarrow_R \mathbb{F}_p$*, computes* $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$ *and outputs* $\sigma = (A, \beta, s)$*.*
– $\mathsf{Verify}(\mathsf{pk}, (m_1, \ldots, m_\ell), \sigma)$ *: This algorithm parses* σ *as* $(\sigma_1, \sigma_2, \sigma_3)$*, and checks*

$$e\left(\sigma_1, w g_2^{\sigma_2}\right) = e\left(g_1 h_0^{\sigma_3} \prod_{i=1}^{\ell} h_i^{m_i},\ g_2\right).$$

If yes, it outputs 1, and outputs 0 otherwise.

PoK for BBS+ Signature Scheme. Here, we recall the proof of knowledge for BBS+ signatures, which was originally proposed in [17].We present a modified version of the same subsequently.

– **Common Input**: Public Key $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$
– $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\mathbf{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
1. $\mathcal{P}$ samples $r_1 \leftarrow_R \mathbb{F}_p^*$ and computes $A' = A^{r_1}$ and $r_3 = r_1^{-1}$
2. $\mathcal{P}$ computes $\bar{A} = (A')^{-\beta} \cdot b^{r_1}$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$.
3. $\mathcal{P}$ samples $r_2 \leftarrow_R \mathbb{F}_p$ and computes $d = b^{r_1} \cdot h_0^{-r_2}$ and $s' = s - r_2 \cdot r_3$
4. $\mathcal{P}$ sends $(A', \bar{A}, d)$ to $\mathcal{V}$, and they run a ZKPoK for the relation:

$$(A')^{-\beta} h_0^{r_2} = \bar{A}/d \ \wedge\ d^{-r_3} h_0^{s'} \prod_{i=1}^{\ell} h_i^{m_i} = g_1^{-1}$$

where $(\mathbf{m}, r_2, r_3, \beta, s')$ is the witness.
5. $\mathcal{V}$ checks that $A' \neq 1_{\mathbb{G}_1}, e(A', w) = e\left(\bar{A}, g_2\right)$, verifies the ZKPoK proof and outputs 1 if all the checks pass, and 0 otherwise.

Modified PoK for BBS+ Signature Scheme. We now present a modified PoK for BBS+ signatures, building on the PoK above, wherein we split the relation $d^{-r_3} h_0^{s'} \prod_{i=1}^{\ell} h_i^{m_i} = g_1^{-1}$ by requiring the prover to equivalently show:

$$d^{-r_3} h_0^{s'-\eta} = C \ \wedge\ h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i} = D \ \wedge\ C \cdot D = g_1^{-1}$$

The above decomposition has advantage that the (long) message $\mathbf{m}$ appears only with public generators which leads to better aggregation of DPoKs over several messages. The complete modified protocol appears below.

– **Common Input**: Public Key $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$
– $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\mathbf{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
1. $\mathcal{P}$ samples $r_1 \leftarrow_R \mathbb{F}_p^*$ and computes $A' = A^{r_1}$ and $r_3 = r_1^{-1}$
2. $\mathcal{P}$ computes $\bar{A} = (A')^{-\beta} \cdot b^{r_1}$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$.
3. $\mathcal{P}$ samples $r_2 \leftarrow_R \mathbb{F}_p$ and computes $d = b^{r_1} \cdot h_0^{-r_2}$ and $s' = s - r_2 \cdot r_3$
4. $\mathcal{P}$ samples $\eta \leftarrow_R \mathbb{F}_p$ and sets $C = d^{-v} h_0^{s'-\eta}$, and $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$.
5. $\mathcal{P}$ sends $(A', \bar{A}, d, C, D)$ to $\mathcal{V}$.
6. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK for the discrete-logarithm relation:

$$(A')^{-\beta} h_0^{r_2} = \bar{A}/d \ \wedge\ d^{-r_3} h_0^{s'-\eta} = C \ \wedge\ h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i} = D$$

where $(\mathbf{m}, r_2, r_3, \beta, s', \eta)$ is the witness.
7. $\mathcal{V}$ checks that $A' \neq 1_{\mathbb{G}_1}, C \cdot D = g_1^{-1}, e(A', w) = e\left(\bar{A}, g_2\right)$, verifies the ZKPoK proof and outputs 1 if all the checks pass, and 0 otherwise.

2.4 Compressed Sigma Protocols

We recall the sigma protocol for vectors, for proving knowledge of discrete log $\mathbf{s} \in \mathbb{F}_p^\ell$ of a vector of group elements $\mathbf{g}$, such that $\mathbf{g}^\mathbf{s} = z$. Here, a prover $\mathcal{P}$ with knowledge of the secret vector $\mathbf{s}$, samples a random vector of scalars $\mathbf{r} \leftarrow_R \mathbb{F}_p^\ell$, and sends $\alpha = \mathbf{g}^\mathbf{r}$ to the verifier $\mathcal{V}$. $\mathcal{V}$ then samples a challenge $c \leftarrow_R \mathbb{F}_p$ and sends it to $\mathcal{P}$ and in the next round $\mathcal{P}$ replies with $\mathbf{x} = c\mathbf{s} + \mathbf{r}$ where $\mathcal{V}$ checks if $\mathbf{g}^\mathbf{x} = z^c\alpha$. Here, the size of the last message of $\mathcal{P}$ is linear in input size, and hence it makes the proof size linear. We note that, for the proof to be succeed, it suffices to convince the verifier $\mathcal{V}$ that $\mathcal{P}$ knows $\mathbf{x}$ such that $\mathbf{g}^\mathbf{x} = z^c\alpha$. Here, we recall the $\log_2 m - 1$ round protocol using the *split and fold* technique [3], which has logarithmic proof size, for proving knowledge of $\mathbf{x} \in \mathbb{F}_p^\ell$ such that $\mathbf{g}^\mathbf{x} = y$ where $y = z^c\alpha$:

– **Common input :** $\mathbf{g} \in \mathbb{G}^m$, $z \in \mathbb{G}$
– $\mathcal{P}$**'s input :** $\mathbf{x} \in \mathbb{F}_p^\ell$

1. $\mathcal{P}$ computes $A = \mathbf{g}_R^{\mathbf{x}_L}$, $B = \mathbf{g}_L^{\mathbf{x}_R}$ and sends them to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_p$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ comutes $\mathbf{x}' = \mathbf{x}_L + c\mathbf{x}_R$.
4. $\mathcal{P}$ and $\mathcal{V}$ independently computes $\mathbf{g}' = \mathbf{g}_L^c \circ \mathbf{g}_R \in \mathbb{G}^{\ell/2}$ and $z' = Ay^cB^{c^2}$.
5. If size$(\mathbf{g}') = 2$, $\mathcal{P}$ sends $\mathbf{x}'$ to $\mathcal{V}$, else $\mathcal{P}$ and $\mathcal{V}$ repeat the protocol from step 1 with $\mathbf{x} = \mathbf{x}'$, $\mathbf{g} = \mathbf{g}'$ and $y = z'$.

where for a vector $\mathbf{s}$, $\mathbf{s}_L$ denotes the left half of the vector and $\mathbf{s}_R$ denote the right half.

The underlying sigma protocol has perfect completeness, special honest-verifier zero-knowledge (SHVZK) and 2-special soundness, and the later protocol has perfect completeness and 3-special soundness at each step of the recursion. Hence, the overall protocol $\mathsf{CSP}\{(z, \mathbf{x}) : \mathbf{g}^\mathbf{x} = z\}$ has perfect completeness, SHVZK which comes from the underlying sigma protocol and $(2, k_1, \ldots, k_{(\log_2 \ell - 1)})$-special soundness, where $k_i = 3\ \forall i \in [\log_2 \ell - 1]$. The protocol can be compiled into a non-interactive argument of knowledge using Fiat-Shamir heuristic [28] in the random oracle model, which we denote by $\mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\{(z, \mathbf{x}) : \mathbf{g}^\mathbf{x} = z\}$ for the random oracle RO.

3 Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge (DPoK) in which multiple provers, each having a share of the witness engage in an interactive protocol with a verifier to convince it that their shares determine a valid witness. The provers do not directly interact with each other, and all the interaction with the verifier takes place over a public broadcast channel.

3.1 Defining a DPoK

Definition 6 (Distributed Proof of Knowledge). We define n-party *distributed proof of knowledge* for relation generator RGen and a secret-sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ by the tuple $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}, \Pi)$ where Setup is a PPT algorithm and Π is an interactive protocol between PPT algorithms $\mathcal{P}$ (prover), $\mathcal{V}$ (verifier) and $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (workers) defined as follows:

– **Setup Phase**: For relation $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $\mathsf{Setup}(\mathcal{R})$ outputs public parameters pp as $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$. The setup phase is required to be executed only once for a given relation $\mathcal{R}$. We assume $\mathcal{R}$ consists of pairs $(\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is parsed as $(\mathbf{s}, \mathbf{t})$) with $\mathbf{s} \in \mathbb{F}^m$. Looking ahead, we partition the witness as $(\mathbf{s}, \mathbf{t})$ to explicitly specify which parts of the witness later needs to be shared [9].
– **Input Phase**: The prover $\mathcal{P}$ receives $(\mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$ as input, while the worker $\mathcal{W}_i$, $i \in [n]$ receives $(\mathbf{x}, \mathbf{s}_i)$ as input, where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. All parties receive $\mathbf{x}$ as input.
– **Preprocessing Phase**: This is (an optional) phase where the prover $\mathcal{P}$ sends some auxiliary information aux_i to worker $\mathcal{W}_i$ using secure private channels.

[9] We specify $\mathbf{s} \in \mathbb{F}^m$ since our secret sharing works over a finite field. The witness component $\mathbf{t}$ need not, in general, be a field element. In fact, in our application, the witness is a message signature pair where the message is in $\mathbb{F}^m$ and the signature is a group element. This group element is not secret shared, yet, the DPOK guarantees extraction of a valid signature message pair.

- **Interactive Phase**: In this phase, the parties interact using a public broadcast channel according to the protocol Π. The protocol Π is a k-round protocol for some $k \in \mathbb{N}$, with $(\mathsf{pp}, \mathbf{x}, \mathbf{s}, \mathbf{t})$ as $\mathcal{P}$'s input, $(\mathsf{pp}, \mathbf{x}, \mathbf{s}_i, \mathsf{aux}_i)$ as the input of $\mathcal{W}_i$ and $(\mathsf{pp}, \mathbf{x})$ as the input of $\mathcal{V}$. The verifier's message in each round $j \in [k]$ consists of a uniformly sampled challenge $\mathbf{c}_j \in \mathbb{F}^{\ell_j}$ for $\ell_j \in \mathbb{N}$. In each round $j \in [k]$, the worker $\mathcal{W}_i$ (resp. the prover $\mathcal{P}$) broadcasts a message $\mathbf{m}_{ij}$ (resp., $\mathbf{m}_i$) which depends on it's random coins and the messages received in prior rounds (including pre-processing phase).
- **Output Phase**: At the conclusion of k rounds, verifier outputs a bit $b \in \{0, 1\}$ indicating accept (1) or reject (0).

A distributed proof of knowledge $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ as described above is said to be *t-private*, *ℓ-robust* if the following hold:

- **Completeness**: We say that completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$ and $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$, the honest execution of all the phases results in 1 being output in the output phase with probability 1.
- **Knowledge-Soundness**: We say that knowledge soundness holds if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ corrupts the prover $\mathcal{P}$ and subset of workers $\{\mathcal{W}_i\}_{i \in \mathsf{C}}$ for some $\mathsf{C} \subseteq [n]$, there exists an extractor Ext with oracle access to $\mathcal{A}_2$ (recall that the prover and the set of corrupt $\mathcal{W}_i$ are controlled by $\mathcal{A}_2$) such the following probability is negligible.

$$\Pr\left[\begin{array}{c} \mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp}, \mathbf{x}) = 1 \wedge \\ ((\mathbf{x}, (\mathbf{s}, \mathbf{t})) \notin \mathcal{R} \vee \\ \mathsf{Consistent}(\{\mathbf{s}_i\}_{i \notin \mathsf{C}}, \mathbf{s}) = 0) \end{array} \middle| \begin{array}{r} \mathcal{R} \leftarrow_R \mathsf{RGen}(\lambda) \\ \mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R}) \\ (\mathbf{x}, \{\mathbf{s}_i\}_{i \notin \mathsf{C}}) \leftarrow_R \mathcal{A}_1(\mathsf{pp}) \\ (\mathbf{s}, \mathbf{t}) \leftarrow_R \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp}, \mathbf{x}, \{\mathbf{s}_i\}_{i \notin \mathsf{C}}) \end{array}\right]$$

  In the above, $\mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp}, \mathbf{x})$ denotes the verifier's output in the protocol Π with its input as $(\mathsf{pp}, \mathbf{x})$ and $\mathcal{A}$ being the adversary. The extractor takes as input the shares of the honest parties specified by the adversary $\mathcal{A}_1$, and with all but negligible probability extracts a valid witness.
- **Honest Verifier Zero-Knowledge**: We say that a DPoK is honest verifier zero-knowledge if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{\mathcal{W}_i\}_{i \in \mathsf{C}}$, where $|\mathsf{C}| \leq t$, there exists a PPT simulator Sim such that $\mathsf{View}_{\mathcal{A},\Pi}(\mathsf{pp}, \mathbf{x})$ is indistinguishable from $\mathsf{Sim}(\mathsf{pp}, \mathbf{x})$ for $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$. Here, the view is given by $\mathsf{View}_{\mathcal{A},\Pi} = \{\mathbf{r}, (\mathbf{M}_i)_{i \in \mathsf{C}}\}$ where $\mathbf{r}$ denotes the internal randomness of $\mathcal{A}$ and $\mathbf{M}_i$ is the set of all messages received by $\mathcal{W}_i$ in Π. We remark that we define honest-verifier zero-knowledge as is standard for public-coin interactive protocols. After Fiat-Shamir compilation into a non-interactive proof, we get full zero-knowledge against a malicious verifier.
- **Robust-Completeness**: We say that robust-completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{\mathcal{W}_i\}_{i \in \mathsf{C}}$, where $|\mathsf{C}| \leq \ell$, $\mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp}, \mathbf{x}) = 1$ with overwhelming probability where $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$.

Remark 1. Robust completeness is a stronger notion of completeness in the sense that it holds even if some corrupt workers deviate maliciously from the protocol, as opposed to the standard notion of completeness which only holds if all the workers follow the protocol. Looking ahead, we use robust complete DPoKs to design authenticated MPC protocols that preserve the underlying protocol's resilience against malicious behavior.

Remark 2. The knowledge soundness extractor expects honest party shares in order to extract the witness. Since knowledge soundness is supposed to hold against a corrupt prover and some corrupt workers, it is meaningful to say that the adversary breaks knowledge soundness if no extractor can construct corrupt party shares that **together with the honest party shares** determine a valid witness.

Remark 3. We assume an honest verifier $\mathcal{V}$ for ease of exposition. In Section 4.2, we relax this assumption by transforming any $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ protocol that uses only public coins and communication over broadcast channels between the workers and the verifier (with no communication among the workers), into a round-efficient version $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ in the random oracle model, wherein the verifier's challenge is computed using the Fiat-Shamir heuristic [28].

3.2 Robust Complete DPoK for Discrete Log

In this section, we provide a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the discrete log relation based on Shamir Secret Sharing (SSS) [46]. Let $\mathsf{DlogGen}$ be a relation generator that on input $(1^\lambda, 1^\ell)$ outputs $(\mathbb{G}, \mathbf{g}, p)$ where p is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\mathbf{g} = (g_1, \ldots, g_\ell) \leftarrow_R \mathbb{G}^\ell$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\mathbf{g}^{\mathbf{s}} = z$. Let $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n) Shamir secret sharing over $\mathbb{F}_p$. Our protocol Π_{dlog} realizing $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ is as below. However, for ease of exposition, we first explain a simpler non-robust version of the protocol, before explaining the robust version. We use an instantiation of compressed sigma protocols (CSP) due to Attema et al. [3] as a black-box (see Section 2.4for details). We run CSP protocol instances over a broadcast channel, meaning that each worker $\mathcal{W}_i$ (playing the role of the prover of that instance) broadcasts its messages as part of the CSP protocol, and the verifier broadcasts all challenges as well.

Warm-up: Non-robust DPoK for DLOG. We begin by describing a simpler, non-robust version of Π_{dlog} outlined above, which we call $\Pi_{\mathsf{nr\text{-}dlog}}$. The protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ follows the steps of Π_{dlog} identically till Step (4a) [Commit to Shares], but skips all of the following steps between Step (4b) and Step (6), and directly executes an output step similar to Step (7). In this step, the protocol either produces an output, or aborts. In particular, $\Pi_{\mathsf{nr\text{-}dlog}}$ aborts if: (i) either any one of the proofs $\pi_{i1}, i \in [n]$ (proving validity of the committed shares) is invalid, or if: (ii) $\left(\prod_{j\in[n]} A_{ij}^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$. Note that the main difference from step (7) in Π_{dlog} is that the above check is performed over the shares of all of the parties, as opposed to over the subset of shares obtained from honest parties. We argue subsequently that this protocol achieves completeness and t-privacy.

Remark 4. The final step checks $(n - t)$ equations over exponents and not just the reconstruction equation. This is to ensure that we extract the witness consistent with honest party shares of the witness. This is crucial in the security proof of our compiler for honest majority protocols where honest party shares determine a unique consistent witness, and this ensures that corrupt parties use the same inputs in both the DPoK protocol and the associated MPC protocol.

Robust DPoK for DLOG. While $\Pi_{\mathsf{nr\text{-}dlog}}$ achieves completeness, it fails to identify the corrupt set of parties, and hence, fails to achieve robust completeness. This is achieved via the additional steps (4b) through (6) in Π_{dlog} outlined in the figure above. We subsequently present a formal proof that Π_{dlog} achieves d-robust completeness for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n - t)$ is the minimum distance of the Reed-Solomon code induced by (t, n)-SSS.

Protocol Π_{dlog}

1. **Public Parameters**: Let $(\mathbb{G}, \mathbf{g}, p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, 1^\ell)$. Let $\mathcal{R}^{\mathrm{DL}}$ denote the relation consisting of pairs $(z, \mathbf{s})$ such that $\mathbf{g}^{\mathbf{s}} = z$. Let $(h_1, h_2) \leftarrow_R \mathsf{Setup}(\mathcal{R}^{\mathrm{DL}})$ be two independent generators of $\mathbb{G}$.
2. **Input Phase**: The prover gets $(z, \mathbf{s})$ while workers $\mathcal{W}_i$, $i \in [n]$ are given $(z, \mathbf{s}_i)$ where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. [10]
3. **Pre-processing:** The prover sends r_i to $\mathcal{W}_i$ for $i \in [n]$ where $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ for $r \leftarrow_R \mathbb{F}_p$.
4. **Commit to Shares:** In the interactive phase, each worker $\mathcal{W}_i$ first commit to their respective shares by
   (a) broadcasting $A_i = \mathbf{g}^{\mathbf{s}_i}$ and running its associated proofs of knowledge $\mathsf{CSP}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$ over broadcast to obtain π_{i1}.
   (b) broadcasting $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$ and running its its associated proofs of knowledge $\mathsf{CSP}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$ over broadcast to obtain π_{i2}.
5. **Reveal Linear Form over Shares:** The verifier samples a challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$ and broacasts it. Thereafter, the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$. To ensure that corrupt workers use $\mathbf{s}_i, r_i$ consistent with earlier commitments A_i, B_i we additionally require them to run the following proof of knowledge CSP over broadcast to obtain π_{i3}:

$$\pi_{i3} = \mathsf{CSP}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\mathbf{s}_i, r_i, \omega_i)) : \mathbf{g}^{\mathbf{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i = v_i\}.$$

6. **Verifier Determines Honest Commitments:** Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the received values in the previous step by the workers, instead of $(v_1, \ldots, v_n)$. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, the

[9] Note that here the witness is $\mathbf{s} \in \mathbb{F}_p^\ell$, and we do not have any component $\mathbf{t}$ which is not being secret-shared.

verifier set $b_i = 0$ else it sets $b_i = 1$. Here we use $\mathbf{v} = (v_1, \ldots, v_n)$ defined by $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$ to denote the vector of honestly computed values. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$, $\mathcal{V}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Set $\mathsf{C} = \{i \in [n] : v_i \neq v'_i \vee b_i = 0\}$ and let $\mathbf{H}_Q = (h_{ij})$ denote the matrix guaranteed by Lemma 1 for $Q = [n] \backslash \mathsf{C} = \{i_1, \ldots, i_q\}$ for $q \in \mathbb{N}$.

7. **Output using Honest Messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\left(\prod_{j \in [q]} A_{i_j}^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, and $(0, \{\mathcal{P}\})$ otherwise.

Theorem 1. *Assuming that* CSP *satisfies completeness, knowledge-soundness and zero-knowledge with* $O(\log \ell)$*-communication overhead,* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *(as per definition 6) for relation generator* DlogGen *and* (t, n)*-*SSS *with the following properties:*

- ***Security****: t-private and d-robust, for* $d <$ dist$/2$*, where* dist $= (n - t)$ *is the minimum distance of the Reed-Solomon code induced by* (t, n)*-*SSS*.*
- ***Efficiency****:* $O(n)$ *communication over point-to-point channels and* $O(n \log \ell)$ *communication over broadcast channels.*

Proof. We provide the proof of security and efficiency below.

Proof of Security. In order to prove security, we prove robust completeness, knowledge-soundness and zero-knowledge.

Robust Completeness. We show that when the prover is honest, and has a correct witness $\mathbf{s}$, the verifier outputs 1 and identifies the corrupt workers with overwhelming probability. Let $\mathcal{A}$ be an adversary corrupting set C' of workers with $|\mathsf{C}'| = d < (n-t)/2$. Let $\mathbf{S}$ denote the matrix with i^{th} column as $(\mathbf{s}_i, r_i)$ for $i \in [n]$. Clearly $\mathbf{S} \in \mathcal{L}^m$ for $m = \ell + 1$. We construct a matrix $\mathbf{S}'$ as follows: for $i \in \mathsf{C}'$ where the adversary's proofs π_{i1}, π_{i2} and π_{i3} are valid, we extract $\mathbf{s}'_i$ and r_i from the proofs π_{i1} and π_{i2} respectively, and set $(\mathbf{s}'_i, r'_i)$ as the i^{th} column of $\mathbf{S}'$. For $i \in \mathsf{C}'$ where one of the proofs is not valid, we set i^{th} column of $\mathbf{S}'$ as $(\mathbf{s}'_i, r'_i)$ for $\mathbf{s}'_i, r'_i$ sampled uniformly. Finally for $i \notin \mathsf{C}'$, we set the i^{th} column of $\mathbf{S}'$ as $(\mathbf{s}_i, r_i)$ (i.e. it is identical to the corresponding column in $\mathbf{S}$). Intuitively, the matrix $\mathbf{S}'$ is the corrupted version of honest matrix $\mathbf{S}$ in which columns corresponding to corrupt provers consist of shares $(\mathbf{s}'_i, r'_i)$ the adversary had in its "head". Looking ahead, we force the adversary to reveal a linear combination over the shares in its "head", and if they are inconsistent with $\mathbf{S}$, the resulting message v'_i will differ from honestly computed v_i (Lemma 2), which will identify the corrupt messages. We now proceed with the formal proof. Let E denote the set of column indices where $\mathbf{S}$ and $\mathbf{S}'$ differ. Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the vector where v'_i is sent by $\mathcal{W}_i$ in Step (5). Clearly, as $\Delta(\mathbf{v}', \mathcal{L}) \leq |\mathsf{C}'| < (n-t)/2$, we can decode $\mathbf{v}'$ to vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathcal{L}$. By uniqueness of decoding, we must have $v'_i = v_i$ for $i \notin \mathsf{C}'$. We will prove that with overwhelming probability we must have $(\mathbf{s}'_i, r'_i) = (\mathbf{s}_i, r_i)$ for all $i \in Q$, which from Lemma 1 will imply that verifier outputs 1 (this is because verifier simply checks matrix relation in Lemma 1 over exponents). For sake of contradiction, assume that $(s'_i, r'_i) \neq (s_i, r_i)$ for $i \in \mathcal{H}$. We can assume that the proofs $\pi_{i1}, \ldots, \pi_{i3}$ were valid, for otherwise $b_i = 0$, which would imply $i \notin \mathcal{H}$, a contradiciton. Now from soundness of the proofs and binding property of the pedersen commitments, with overwhelming probability we must have $v'_i = \langle \boldsymbol{\gamma}, \mathbf{s}'_i \rangle + r'_i$. By assumption we have $i \in E$ and thus from Lemma 2, with overwhelming probability we have $v'_i \neq v_i$. Thus $i \notin \mathcal{H}$, which is again a contradiction. This proves that $\mathbf{s}'_i = \mathbf{s}_i$ for $i \in \mathcal{H}$, and thus the vector $(\mathbf{s}'_i)_{i \in \mathcal{H}}$ is $\mathcal{L}^m$-consistent. From Lemma 1, we conclude that the verifier outputs 1.

Knowledge-Soundness. To prove knowledge-soundness, we describe the extractor Ext which is provided the shares $\mathbf{s}_i, i \notin \mathsf{C}$ with C denoting the indices of workers corrupted by adversary $\mathcal{A}$. The extractor Ext runs the adversary $\mathcal{A}$. When $\mathcal{A}$ succeeds, for each $j \in [q]$ in Step (6) the extractor Ext sets $\mathbf{s}'_{i_j} = \mathbf{s}_{i_j}$ if $i_j \notin \mathsf{C}$; otherwise it invokes the extractor for CSP, which has oracle access to the worker $\mathcal{W}_{i_j}$ acting as the prover for the instantiation of $\mathsf{CSP}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$, to extract $\mathbf{s}'_{i_j}$ satisfying $\mathbf{g}^{\mathbf{s}'_{i_j}} = A_{i_j}$. The verification check in Step (7) implies that the tuple $\left(\mathbf{s}'_{i_j}\right)_{j \in [q]}$ is $\mathcal{L}^\ell$-consistent. The extractor outputs the witness $\mathbf{s}$, which is reconstructed from the columns of the unique matrix $\mathbf{S} \in \mathcal{L}^\ell$ determined by the tuple $\left(\mathbf{s}'_{i_j}\right)_{j \in [q]}$ This completes the proof of knowledge-soundness for Π_{dlog}.

Zero-Knowledge. For proving zero-knowledge, we assume WLOG that $\mathsf{C} = \{1, \ldots, \epsilon\}$ for $\epsilon \leq t$. The simulator Sim runs the adversary to obtain messages $\{A_i, B_i\}_{i \in \mathsf{C}}$. It then simulates messages of the honest parties as follows: Choose $A'_i \leftarrow_R \mathbb{G}$ for $1 \leq i \leq t$. Set $\mathbf{a} = (z, A'_1, \ldots, A'_t)$. Next, Sim sets $A'_{t+j} = \mathbf{a}^{\mathbf{t}_j}$ where $\mathbf{t}_j \in \mathbb{F}_p^{t+1}$ is the interpolation vector such that $f(t+j) = \langle (f(0), \ldots, f(t)), \mathbf{t}_j \rangle$ for all polynomials $f(x)$ of degree $\leq t$. The simulator picks B'_i, $i > \epsilon$ uniformly at random from $\mathbb{G}$. It simulates messages $\{A'_i, B'_i\}_{i > \epsilon}$ towards $\mathcal{A}$. The intuition behind simulation of A_j's is as follows: In the real protocol, the vector of shares for party j is of the form $(f_1(j), \ldots, f_\ell(j))$, where $f_i : i \in [\ell]$ are the polynomials used to share the values $s_i : i \in [\ell]$ respectively. Let $\mathbf{f} = (f_1, \ldots, f_\ell)$ denote the vector of sharing polynomials and let $\mathbf{f}(j)$ to denote the vector $(f_1(j), \ldots, f_\ell(j))$. Then for $j > \epsilon$ in the real protocol, $(A_j)_{j > \epsilon}$ are distributed as $(\mathbf{g}^{\mathbf{f}(j)})_{j > \epsilon}$, subject to constraint that $\mathbf{g}^{\mathbf{f}(0)} = z$. Sampling such a polynomials f_i, $i \in [\ell]$ corresponds to choosing $f_i(1), \ldots, f_i(t)$ uniformly and then determining $f_i(t+j) = \langle (f_i(0), \ldots, f_i(t)), \mathbf{t}_j \rangle$ using the interpolation vector $\mathbf{t}_j$. Thus $\mathbf{f}(t+j)$ is a $\mathbf{t}_j$-linear combination of $\mathbf{f}(0), \ldots, \mathbf{f}(t)$, which dictates simulator's computation of A_{t+j} from vector $\mathbf{a}$. Next, the simulator simulates the challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$. Then, on receiving $v_1, \ldots, v_\epsilon$ from $\mathcal{A}$, the simulator computes $(v'_1, \ldots, v'_n) \leftarrow_R \mathsf{Share}(v')$ for $v' \leftarrow_R \mathbb{F}_p$, computes simulated CSP proofs $\{\pi_{i1}, \pi_{i2}, \pi_{i3}\}_{i > \epsilon}$. Finally, the simulator simulates $(v'_i, \pi_{i1}, \pi_{i2}, \pi_{i3})_{i > \epsilon}$ towards $\mathcal{A}$. This completes the proof of zero-knowledge for Π_{dlog}. *Proof of Efficiency/Succinctness.* Assuming that CSP has $O(\log \ell)$-communication overhead [3], it follows by inspection that Π_{dlog} incurs $O(n)$ communication over point-to-point channels (where the prover distributes additional randomness to the workers) and $O(n \log \ell)$ communication over broadcast channels (for n instances of CSP). This completes the proof of efficiency/succinctness for Π_{dlog}, and hence the proof of Theorem 1. □

Corollary 1. *Setting $d = t < n/3$, Π_{dlog} is $n/3$-private and $n/3$-robust.*

Finally, the following corollary also follows immediately from the proof of Theorem 1, and formally captures the properties of the non-robust protocol $\Pi_{\mathsf{nr\text{-}dlog}}$.

Corollary 2. *Assuming that* CSP *satisfies completeness, knowledge-soundness and zero-knowledge with $O(\log \ell)$-communication overhead, $\Pi_{\mathsf{nr\text{-}dlog}}$ is a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for relation generator* $\mathsf{DlogGen}$ *and (t, n)-*SSS *that satisfies completeness and t-privacy, and incurs $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

Note that $\Pi_{\mathsf{nr\text{-}dlog}}$ retains all properties of its robust counterpart apart from d-robustness as stated in Theorem 1.

Generalization to Threshold Linear Secret Sharing. Finally, we can generalize the above protocol to work with any *threshold linear secret sharing scheme* (TLSS). The following results appear in Appendix A along with other relevant details.

Theorem 2 (Robust Distributed Proof of Knowledge for Discrete Log for TLSS). *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, a generalization of the above protocol is a $\mathsf{DPoK}_{\mathsf{TLSS},\mathsf{DlogGen}}$ for relation generator* $\mathsf{DlogGen}$ *and (t, n, r)-*TLSS*, satisfies t-privacy and d-robustness, for $d < \mathsf{dist}/2$ (where* dist *is the minimum distance the linear code induced by* TLSS*), and incurs $O(rn)$ communication over point-to-point channels and $O(rn + \log \ell)$ communication over broadcast channels.*

Note that the exact corruption threshold depends on the exact distance of the linear code induced by the TLSS scheme. As an example, we provide concrete bounds for *Replicated Secret Sharing* in the corollary below:

Corollary 3 (Robust Distributed Proof of Knowledge for Discrete Log for Replicated Secret Sharing). *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, we obtain a $\mathsf{DPoK}_{\mathsf{RSS},\mathsf{DlogGen}}$ for relation generator* $\mathsf{DlogGen}$ *and $(t, n, \binom{n-1}{t})$-*RSS *scheme which satisfies t-privacy and d-robustness for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n - t)$ is the minimum distance of the linear code induced by the* TLSS *scheme.*

4 Round Efficient Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge (DPoK) in the random oracle model (ROM) which multiple provers, each having a share of the witness engage in an interactive

protocol with a verifier to convince it that their shares determine a valid witness. The provers do not directly interact with each other, and all the interaction with the verifier takes place over a public broadcast channel. We begin by recalling the standard definition of NIZK in the ROM, which we use in our definition and constructions of publicly verifiable DPoK.

4.1 NIZK in the ROM

The Fiat-Shamir heuristic [28] transforms a public-coin interactive proof into an non-interactive version in the random oracle model. Given a public coin proof system $\Pi = (\mathcal{P}, \mathcal{V})$ with r rounds such that Ch_i is the challenge space for the ith round, the corresponding non-interactive proof system $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ is defined as follows.

- $\mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda)$ The setup algorithm for $i \in [1, r]$ samples a function H_i uniformly from a set of all functions that map $\{0,1\}^*$ to Ch_i. Note that this is equivalent to instantiating H_i from a single random oracle via domain separation. We denote by H the set $\{\mathsf{H}_i\}_{i\in[1,r]}$.
- $\pi \leftarrow_R \mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}(x, w)$ The prover produces a proof string π on input statement x, and witness w. For each round $i \in [1, r]$, $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}$ invokes the next message function of the interactive prover $\mathcal{P}(x, w)$ on prior challenge c_{i-1} to get a_i, and obtains the ith round challenge by computing $c_i = \mathsf{H}_i(x, a_1, c_1, \ldots, a_{i-1}, c_{i-1}, a_i)$. Then $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}$ outputs $\pi = (a_1, c_1, \ldots, a_r, c_r, a_{r+1})$.
- $b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi)$ The verifier on input statement x, and proof string π, outputs a decision bit. $\mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}$ outputs $b = 1$, meaning the verifier accepts the proof, iff $\mathcal{V}(x, \pi) = 1$ *and* $c_i = \mathsf{H}_i(x, a_1, c_1, \ldots, c_{i-1}, a_i)$ for all $i \in [1, r]$.

Definition 7 (Knowledge soundness in the ROM). *Consider a non-interactive proof system $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ for relation $\mathcal{R}$. Π_{FS} is* extractable *with knowledge error $\kappa : \mathbb{N} \times \mathbb{N} \to [0, 1]$ in the random oracle model, if there exists an extractor* Ext *and some polynomial* poly*, such that for any* PPT *adversary $\mathcal{P}$ that makes at most q queries to* H*, it holds that*

$$\mathrm{ext}(\mathcal{P}, \mathsf{Ext}) \geq \frac{\mathrm{acc}(\mathcal{P}) - \kappa(\lambda, q)}{\mathsf{poly}(\lambda)}$$

and Ext *halts in an expected number of steps that is polynomial in λ and q, where the probabilities* acc *and* ext *are defined as follows.*

$$\mathrm{acc}(\mathcal{P}) = \Pr\left[\, b = 1 \;\middle|\; \begin{array}{r} \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda); \\ (x, \pi) \leftarrow_R \mathcal{P}^{\mathsf{H}}(\rho); \\ b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi) \end{array} \right]$$

$$\mathrm{ext}(\mathcal{P}, \mathsf{Ext}) = \Pr\left[\begin{array}{c} b = 1 \wedge \\ (x, w) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{r} \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda); \\ (x, \pi) \leftarrow_R \mathcal{P}^{\mathsf{H}}(\rho); \\ b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi); \\ w \leftarrow_R \mathsf{Ext}^{\mathcal{P}}(x, \pi, \rho, \mathcal{Q}_1) \end{array} \right]$$

where $\mathcal{Q}_1 = \{\mathcal{Q}_{1,i}\}_{i\in[1,r]}$ is the set consisting of pairs corresponding to queries to the random oracle H *with index i, and the response. In the experiment* ext*,* Ext *has oracle access to the next-message function of $\mathcal{P}$.*

Zero-knowledge for non-interactive proofs is defined in the explicitly programmable random oracle model where the simulator is allowed to program the random oracle. The zero-knowledge simulator $\mathcal{S}_{\mathsf{FS}}$ is modeled as a stateful algorithm that operates in two modes. In the first mode, $(c_i, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, x, i)$ handles random oracle calls to H_i on input x. In the second mode, $(\tilde{\pi}, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, x)$ simulates a valid proof string. We define stateful wrapper oracles.

- $\mathcal{S}_1(t, i)$ denotes the oracle that returns the first output of $\mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, t, i)$;
- $\mathcal{S}_2(x, w)$ returns the first output of $\mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, x)$ if $(\mathsf{pp}, x, w) \in \mathcal{R}$ and $\perp$ otherwise; (This is because ZK is defined only for true statements.)

Definition 8 (Non-interactive Zero Knowledge). *A NIZK* $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}, \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}})$ *for relation* $\mathcal{R}$ *is* non-interactive zero knowledge *in the random oracle model, if there exists a PPT simulator* $\mathcal{S}_{\mathsf{FS}} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for all PPT distinguisher* $\mathcal{D}$*, the following is negligible in* λ

$$\Big| \Pr\left[\mathcal{D}^{\mathsf{H},\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}}(1^\lambda) = 1 \ : \ \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda)\right] - \Pr\left[\mathcal{D}^{\mathcal{S}_1,\mathcal{S}_2}(1^\lambda) = 1 \ : \ \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda)\right] \Big|$$

where both $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}(x, w)$ *and* $\mathcal{S}_2$ *return* $\perp$ *if* $(x, w) \notin \mathcal{R}$*.*

Additionally, given a HVZK simulator $\mathcal{S}$ for Π, we can construct a NIZK simulator $\mathcal{S}_{\mathsf{FS}}$ for Π_{FS} as follows.

- On query (x, i) with mode 1, $\mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, x, i)$ lazily samples a lookup table $\mathcal{Q}_{1,i}$ maintained in state st. It checks whether $\mathcal{Q}_{1,i}[x]$ is already defined; if yes, it returns the previously assigned value; otherwise it returns and sets a fresh random value c_i sampled from Ch_i.
- On query x with mode 2, $\mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, x)$ calls the HVZK simulator $\mathcal{S}$ of Π to obtain a simulated transcript $\tilde{\pi} = (a_1, c_1, \ldots, a_r, c_r, a_{r+1})$. Then, it programs the tables such that $\mathcal{Q}_{1,1}[x, a_1] := c_1, \ldots, \mathcal{Q}_{1,r}[x, a_1, c_1, \ldots, a_r] := c_r$. If any of the table entries has been already defined $\mathcal{S}_{\mathsf{FS}}$ aborts, which happens with negligible probability under the assumption that a_1 has high min-entropy.

4.2 Round Efficient DPoK

We define a round efficient DPoK by building upon our original definition for DPoK from Section 3. Our definition is based on the Fiat-Shamir heuristic [28], using which we transform a DPoK (with number of rounds logarithmic in the size of the message) into a round efficient DPoK (having constant number of rounds).

Definition 9 (Round Efficient DPoK in the ROM). Let $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}, \Pi)$ be a DPoK as in Definition 6 for relation generator RGen and a secret-sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$, where Setup is a PPT algorithm, and Π is a k-round interactive protocol between PPT algorithms $\mathcal{P}$ (prover), $\mathcal{V}$ (interactive verifier) and $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (workers), such that all of the interaction with the verifier takes place over a public broadcast channel, and where in each round $j \in [k]$, the verifier $\mathcal{V}$ broadcasts a challenge sampled uniformly from the challenge set Ch_j. We define the corresponding round efficient DPoK for the same $(\mathsf{RGen}, \mathsf{SSS})$ pair as a tuple of the form $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}_{\mathsf{FS}}, \Pi_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$, where $\mathsf{Setup}_{\mathsf{FS}}$ is a PPT setup algorithm, Π_{FS} is an interactive protocol between PPT algorithms $\mathcal{P}_{\mathsf{FS}}$ (prover) and $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_1, \ldots, (\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_n$ (workers), and $\mathcal{V}_{\mathsf{FS}}$ is PPT verification algorithm. These are defined as follows:

- **Setup** $[(\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}, 1^\lambda)]$: The setup algorithm takes as input a relation $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$ and outputs a tuple of the form $(\mathsf{pp}, \mathsf{RO})$, where $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$, and $\mathsf{RO} = \{\mathsf{RO}_i\}_{i \in [1,r]}$, with each RO_i being a random function sampled uniformly from the set of all functions that maps $\{0,1\}^*$ to the challenge set Ch_i. As in our general definition of DPoK, the setup phase is required to be executed only once for a given relation $\mathcal{R}$. We again assume that $\mathcal{R}$ consists of pairs $(\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is parsed as $(\mathbf{s}, \mathbf{t})$) with $\mathbf{s} \in \mathbb{F}^m$; looking ahead, we partition the witness as $(\mathbf{s}, \mathbf{t})$ to explicitly specify which parts of the witness later needs to be shared. Also, note that sampling each RO_i independently is equivalent to instantiating RO_i from a single random oracle via domain separation.
- **Interactive Protocol** Π_{FS}: executed jointly by the prover $\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}$ and the workers $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_1, \ldots, (\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_n$ in the following phases:
  - **Input Phase**: The prover $\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}$ receives $(\mathsf{pp}, \mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$ as input, while each worker $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_i$, $i \in [n]$ receives $(\mathbf{x}, \mathbf{s}_i)$ as input, where $(\mathsf{pp}, \mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$.
  - **Preprocessing Phase**: This is (an optional) phase where the prover $\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}$ sends some auxiliary information aux_i to worker $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_i$ using secure private channels. This phase is identical to the preprocessing phase (if any) in the underlying DPoK scheme, with the prover $\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}$ invoking the prover $\mathcal{P}$ of DPoK to obtain its output in the preprocessing phase, and sending the same to the workers $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_1, \ldots, (\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_n$.

- **Interactive Phase**: In this phase, the prover and the workers interact using a public broadcast channel as follows, where all algorithm presented with FS subscript have access to the random oracle RO:
  * The prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ (resp. each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$) invokes the prover $\mathcal{P}$ (resp. the corresponding worker $\mathcal{W}_i$ of) of DPoK to produce the same round message as in the protocol Π.
  * Suppose that in round j of the protocol Π (for $j \in [k]$), the verifier $\mathcal{V}$ of the underlying DPoK outputs a challenge $\mathbf{c}_j \leftarrow_R \mathsf{Ch}_j$. In $\mathit{\Pi}_{\mathsf{FS}}$, each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ computes $\mathbf{c}_j$ locally as

$$\mathbf{c}_j = \mathsf{RO}_j\left(\mathbf{x}, \{\{\mathbf{m}_{i,\ell}\}_{i\in[n]}, \mathbf{c}_\ell\}_{\ell\in[j-1]}\right),$$

    where $\mathbf{m}_{i,\ell}$ is the prior message of $\mathcal{W}_i$ in round ℓ, and $\mathbf{c}_\ell$ is prior challenge in round ℓ.

  Let $\pi = \left(\mathbf{x}, \{\{\mathbf{m}_{i,\ell}\}_{i\in[n]}, \mathbf{c}_\ell\}_{\ell\in[k]}\right)$ denote the transcript of protocol $\mathit{\Pi}_{\mathsf{FS}}$ at the conclusion of k rounds.

– **Verification**: $[b \leftarrow_R \mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \mathbf{x}, \pi)]$: The verifier $\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}$ takes as input $(\mathsf{pp}, \mathbf{x}, \pi)$ and outputs a decision bit $b \in \{0,1\}$. It outputs 1 if and only if both of the following hold: (i) $\mathcal{V}(\mathsf{pp}, \mathbf{x}, \pi) = 1$ ($\mathcal{V}$ being the verifier of DPoK), and (ii) for each $j \in [k]$, $\mathbf{c}_j = \mathsf{RO}_j\left(\mathbf{x}, \{\{\mathbf{m}_{i,\ell}\}_{i\in[n]}, \mathbf{c}_\ell\}_{\ell\in[j-1]}\right)$. Otherwise, the verifier $\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}$ outputs 0.

A distributed proof of knowledge $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ as described above is said to be *t-private*, *ℓ-robust* if the following hold:

– **Completeness**: We say that completeness holds if for any $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, for $(\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}, 1^\lambda, 1^k)$, and for any $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$, if π denotes the transcript of an honest execution of the protocol $\mathit{\Pi}_{\mathsf{FS}}$, then we have

$$\Pr[\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \mathbf{x}, \pi) = 1] = 1$$

– **Knowledge Soundness**: We say that knowledge soundness holds if for any security parameter λ and any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that makes at most $Q = \mathsf{poly}(\lambda)$ queries to RO, where $\mathcal{A}_2$ corrupts the prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and subset of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$ for some $\mathsf{C} \subseteq [n]$, there exists an extractor Ext with oracle access to $\mathcal{A}_2$ (which controls $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and the set of corrupt $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$) such that for any $\mathcal{R} \leftarrow_R \mathsf{RGen}(\lambda)$, the following probability is negligible,

$$\Pr\left[\begin{array}{c} \mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \mathbf{x}, \pi) = 1 \wedge \\ ((\mathbf{x}, (\mathbf{s}, \mathbf{t})) \notin \mathcal{R} \vee \\ \mathsf{Consistent}(\{\mathbf{s}_i\}_{i\notin\mathsf{C}}, \mathbf{s}) = 0) \end{array} \middle| \begin{array}{r} (\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}) \\ (\mathbf{x}, \{\mathbf{s}_i, \mathsf{aux}_i\}_{i\notin\mathsf{C}}) \leftarrow_R \mathcal{A}_1(\mathsf{pp}) \\ \pi := \\ \mathit{\Pi}_{\mathsf{FS}}\big(\mathcal{A}_2(\rho), \{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i(\alpha_i)_{i\notin\mathsf{C}}\}\big) \\ (\mathbf{s}, \mathbf{t}) \leftarrow_R \\ \quad \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp}, \mathbf{x}, \{\mathbf{s}_i\}_{i\notin\mathsf{C}}, \pi, \mathcal{Q}) \end{array}\right]$$

where π denotes the transcript of an execution of the protocol $\mathit{\Pi}_{\mathsf{FS}}$ between the adversary $\mathcal{A}_2$ (which controls $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and the set of corrupt $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$), and the honest workers.

– **Zero-Knowledge**: Zero-knowledge for publicly verifiable DPoKs is defined in the explicitly programmable random oracle model where the simulator is allowed to program the random oracle. The zero-knowledge simulator $\mathcal{S}_{\mathsf{FS}}$ is modeled as a stateful algorithm that operates in two modes. In the first mode, $(c_i, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, \mathbf{x}, i)$ handles random oracle calls to RO_i on input $\mathbf{x}$. In the second mode, $(\tilde{\pi}, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, \mathbf{x})$ simulates a valid proof string. We define stateful wrapper oracles.
  - $\mathcal{S}_1(t, i)$ denotes the oracle that returns the first output of $\mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, t, i)$;
  - $\mathcal{S}_2(x, w)$ returns the first output of $\mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, \mathbf{x})$ if $(\mathsf{pp}, \mathbf{x}, \mathbf{s}) \in \mathcal{R}$ and $\perp$ otherwise; (This is because ZK is defined only for true statements.)

  We say that a DPoK is zero-knowledge in the random oracle model if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq t$, there exists a PPT simulator $\mathcal{S}_{\mathsf{FS}}$ such that $\mathsf{View}_{\mathcal{A},\mathsf{RO},\mathit{\Pi}_{\mathsf{FS}}}(\mathsf{pp}, \mathbf{x})$ is indistinguishable from $\mathcal{S}_{\mathsf{FS}}(\mathsf{pp}, \mathbf{x})$ for $\mathsf{pp} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R})$. Here, the view is given by $\mathsf{View}_{\mathcal{A},\mathsf{RO},\mathit{\Pi}_{\mathsf{FS}}} = \{\mathbf{r}, (\mathbf{M}_i)_{i\in\mathsf{C}}\}$ where $\mathbf{r}$ denotes the internal randomness of $\mathcal{A}$ and $\mathbf{M}_i$ is the set of all messages received by $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ in $\mathit{\Pi}_{\mathsf{FS}}$.

– **Robust-Completeness**: We say that robust-completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq \ell$, $(\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}})_{\mathcal{A},\mathit{\Pi}_{\mathsf{FS}}}(\mathsf{pp}, \mathbf{x}, \mathit{\Pi}_{\mathsf{FS}}) = 1$ with overwhelming probability where $\mathsf{pp} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R})$.

Protocol $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$

1. **Public Parameters**: Let $(\mathbb{G}, \mathbf{g}, p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, 1^\ell)$. Let $\mathcal{R}^{\mathrm{DL}}$ denote the relation consisting of pairs $(z, \mathbf{s})$ such that $\mathbf{g}^{\mathbf{s}} = z$. Let $(h_1, h_2) \leftarrow_R \mathsf{Setup}(\mathcal{R}^{\mathrm{DL}})$ be two independent generators of $\mathbb{G}$.
2. **Input Phase**: The prover gets $(z, \mathbf{s})$ while workers $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$, $i \in [n]$ are given $(z, \mathbf{s}_i)$ where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. [11]
3. **Pre-processing:** The prover sends r_i to $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ for $i \in [n]$ where $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ for $r \leftarrow_R \mathbb{F}_p$.
4. **Commit to Shares:** In the interactive phase, the workers first commit to their respective shares by broadcasting
   (a) $A_i = \mathbf{g}^{\mathbf{s}_i}$ and its associated proofs of knowledge $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$.
   (b) $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$ and its associated proofs of knowledge $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
5. **Reveal Linear Form over Shares:** Each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ computes $\boldsymbol{\gamma}$ as $\boldsymbol{\gamma} = \mathsf{RO}(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n) \in \mathbb{F}_p^\ell$. Thereafter, the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$. To ensure that corrupt workers use $\mathbf{s}_i, r_i$ consistent with earlier commitments A_i, B_i we additionally require them to broadcast proof π_{i3} as:

$$\pi_{i3} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\mathbf{s}_i, r_i, \omega_i)) : \mathbf{g}^{\mathbf{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i = v_i\}.$$

6. **Verifier Determines Honest Commitments:** Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the received values in the previous step by the workers, instead of $(v_1, \ldots, v_n)$. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, the verifier set $b_i = 0$ else it sets $b_i = 1$. Here we use $\mathbf{v} = (v_1, \ldots, v_n)$ defined by $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$ to denote the vector of honestly computed values. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$, $\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Set $\mathsf{C} = \{i \in [n] : v_i \neq v'_i \vee b_i = 0\}$ and let $\mathbf{H}_Q = (h_{ij})$ denote the matrix guaranteed by Lemma 1 for $Q = [n] \backslash \mathsf{C} = \{i_1, \ldots, i_q\}$ for $q \in \mathbb{N}$.
7. **Output using Honest Messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\left(\prod_{j \in [q]} A_{ij}^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, and $(0, \{\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\})$ otherwise.

4.3 Robust Complete Round Efficient DPoK for Discrete Log

In this section, we provide a $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the discrete log relation based on Shamir Secret Sharing (SSS) [46]. Let DlogGen be a relation generator that on input $(1^\lambda, 1^\ell)$ outputs $(\mathbb{G}, \mathbf{g}, p)$ where p is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\mathbf{g} = (g_1, \ldots, g_\ell) \leftarrow_R \mathbb{G}^\ell$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\mathbf{g}^{\mathbf{s}} = z$. Let $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n) Shamir secret sharing over $\mathbb{F}_p$. Our protocol Π_{dlog} realizing $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ is as below. However, for ease of exposition, we first explain a simpler non-robust version of the protocol, before explaining the robust version.

We use the non-interactive proof of knowledge for the discrete logarithm relation, namely $\mathsf{NIPK}_{\mathsf{FS}} = (\mathsf{NIPK}.\mathsf{Setup}_{\mathsf{FS}}, \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}, \mathsf{NIPK}.\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}})$, obtained by applying the Fiat-Shamir heuristic (using random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}_p^\ell$) on the public-coin compressed sigma protocol [3] for proof of knowledge of the discrete logarithm relation. Additionally, we present the protocol Π_{dlog} using Fiat-Shamir heuristic [28] and a random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}_p^\ell$.

We now state and prove the following theorem for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$.

Theorem 3. *Assuming that* NIPK *satisfies completeness, knowledge-soundness and zero-knowledge with $O(\log \ell)$-communication overhead, $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ is a* $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *(as per definition 6) for relation generator* DlogGen *and (t, n)-*SSS *with the following properties:*

- ***Security**: t-private and d-robust, for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n - t)$ is the minimum distance of the Reed-Solomon code induced by (t, n)-*SSS*.*
- ***Efficiency**: $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

[11] Note that here the witness is $\mathbf{s} \in \mathbb{F}_p^\ell$, and we do not have any component $\mathbf{t}$ which is not being secret-shared.

Proof sketch. For knowledge-soundness, the intuition behind extraction of a valid witness are the fact that the shares (provided to the extractor via definition) held by the honest parties uniquely determines the output and the adversary succeeds in proving the statement in a protocol where these honest-party shares are used. For zero-knowledge, the key intuition behind the simulation is that the adversial messages can be 'ignored' for providing an accepting transcript as the protocol does 'error-correction' and removes the 'bad shares' from consideration.

Proof. Completeness and robust completeness of $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ follows similarly from the completeness and robust completeness of its respective counterpart Π_{dlog}.

Knowledge-Soundness. To prove knowledge-soundness, we describe the extractor Ext for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ as follows. Let C be the set of indices of workers corrupted by adversary $\mathcal{A}$. Additionally, we assume that there is an extractor Ext_1 for NIPK proof. The extractor Ext runs the adversary $\mathcal{A}$ as follows:

- Ext is provided $(\mathsf{pp}, z, \{\mathbf{s}_i\}_{i\notin\mathsf{C}}, \mathit{\Pi}_{\mathsf{FS}}, \mathcal{Q})$ as input at the onset, where $\{\mathbf{s}_i\}_{i\notin\mathsf{C}}$ are the honest-party shares and $\mathcal{Q}$ is the set of RO queries made by the adversary $\mathcal{A}$.
- Ext receives A_i, B_i from $\mathcal{A}$ along with the NIPK proofs $\{\pi_{i1}, \pi_{i2}\}$ for all $i \in \mathsf{C}$, such that $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
- Ext computes $\{A_i = \mathbf{g}^{\mathbf{s}_i}, B_i = h_1^{r_i} h_2^{\omega_i}\}_{i\notin\mathsf{C}}$ and sends $\{A_i, , \pi_{i1}, B_i, \pi_{i2}\}_{i\notin\mathsf{C}}$ to $\mathcal{A}$, where $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
- Ext computes $\boldsymbol{\gamma} = \mathsf{RO}\left(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n\right)$
- Ext receives $\{v_i, \pi_{i3}\}_{i\in\mathsf{C}}$ from $\mathcal{A}$
- Ext computes v_i, π_{i3} as $\{v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i\rangle + r_i\}_{i\notin\mathsf{C}}$ and $\pi_{i3} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{((A_i B_i, \boldsymbol{\gamma}\|\mathbf{1}\|\mathbf{0}, v_i), (\mathbf{s}_i, r_i, \omega_i)) : \mathbf{g}^{\mathbf{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \mathbf{s}_i\rangle + r_i = v_i\}$, and sends $\{v_i, \pi_{i3}\}_{i\notin\mathsf{C}}$
- Ext sets $\mathbf{s}'_i = \mathbf{s}_i$ for all $i \notin \mathsf{C}$ and for all $i \in \mathsf{C}$, it invokes the extractor Ext_1 for the Fiat-Shamir transformed proof π_{i1} to extract $\mathbf{s}'_i$ satisfying $\mathbf{g}^{\mathbf{s}'_i} = A_i$.
- Ext finally computes $\mathbf{s}'$ as $\mathbf{s}' = \mathsf{Reconstruct}(\{s_i\}_{i\notin\mathsf{C}})$ and outputs $\mathbf{s}'$.

Note that by using random oracle RO to obtain the challenge $\boldsymbol{\gamma}$ in Step (iii) described above, we ensure that a 'random linear combination' of the code is considered in Step (6) of the protocol. Now, considering that the adversary $\mathcal{A}$ succeeds, we now argue the correctness of the extracted witness. Since the adversary succeeds, the verification check in Step (7) of the protocol implies that the tuple $\left(\mathbf{s}'_i\right)_{i\notin\mathsf{C}}$ is $\mathcal{L}^\ell$-consistent and the reconstructed vector $\mathbf{s}'$ satisfies $\mathbf{s}' = \mathsf{Reconstruct}(\{s_i\}_{i\notin\mathsf{C}})$ along with $\left(\prod_{j\notin\mathsf{C}} A_j^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, where $A_j = \mathbf{g}^{\mathbf{s}_j}$ for all $j \notin \mathsf{C}$. Note that the extractor's output $\mathbf{s}'$ is reconstructed from the columns of the unique matrix $\mathbf{S} \in \mathcal{L}^\ell$ determined by the tuple $(\mathbf{s}'_i)_{j\notin\mathsf{C}}$. Hence, the extractor output is a valid witness for the given statement. This completes the proof of knowledge-soundness for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$.

Knowledge-error. Since there are three non-parallel instances of Fiat-Shamir transformed NIPK protocol from Attema et al. [3] being invoked, if the knowledge-error of the Fiat-Shamir transformed version is κ', then the knowledge-error of $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ is $\kappa \leq 3\kappa'$. And we know from [3] that the knowledge-error κ'' of NIPK protocol is negligible, and [4] ensures that the knowledge-error κ' of non-parallel Fiat-Shamir version of the multi-round protocol is still negligible and degrades only linearly with respect to the number of queries to the Random Oracle. Specifically, if Q is the upper-bound for the number of Random Oracle queries for NIPK protocol, then given that κ'' is the knowledge-error of the interactive NIPK protocol, from [4] we get that $\kappa' = (Q+1).\kappa''$.

Zero-Knowledge. For proving zero-knowledge, we describe the simulator as follows. Without loss of generaltiy, let us assume that $\mathsf{C} = \{1, \ldots, \epsilon\}$ for $\epsilon \leq t$. The simulator $\mathcal{S}_{\mathsf{FS}}$ runs the adversary as follows:

- $\mathcal{S}_{\mathsf{FS}}$ receives $\{A_i, B_i\}_{i\in\mathsf{C}}$ from the adversary.
- $\mathcal{S}_{\mathsf{FS}}$ simulates messages $\{A_i, B_i, \pi_{i1}, \pi_{i2}\}_{i\notin\mathsf{C}}$ of the honest parties as follows:
  - $\mathcal{S}_{\mathsf{FS}}$ chooses $A'_i \leftarrow_R \mathbb{G}$ for $1 \leq i \leq t$, and sets $\mathbf{a} = (z, A'_1, \ldots, A'_t)$.
  - $\mathcal{S}_{\mathsf{FS}}$ sets $A'_{t+j} = \mathbf{a}^{\mathbf{t}_j}$ where $\mathbf{t}_j \in \mathbb{F}_p^{t+1}$ is the interpolation vector such that $f(t+j) = \langle (f(0), \ldots, f(t)), \mathbf{t}_j\rangle$ for all polynomials $f(x)$ of degree $\leq t$, i.e. $\mathbf{t}_j = \{\lambda_0(t+j), \lambda_1(t+j), \ldots, \lambda_t(t+j)\}$ where $\lambda_0(x), \ldots, \lambda_t(x)$ are lagrange polynomials with respect to the set $\{0, \ldots, t\}$.
  - $\mathcal{S}_{\mathsf{FS}}$ picks B'_i, $i > \epsilon$ uniformly at random from $\mathbb{G}$.
  - $\mathcal{S}_{\mathsf{FS}}$ invokes the simulator for the NIPK to obtain $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.

  - Then $\mathcal{S}_{\mathsf{FS}}$ sends the messages $\{A'_i, B'_i, \pi_{i1}, \pi_{i2}\}_{i>\epsilon}$ to $\mathcal{A}$.
- $\mathcal{S}_{\mathsf{FS}}$ queries the random oracle RO to obtain the challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$.
- Thereafter, the simulator receives $\{v_i\}_{i<\epsilon}$ from $\mathcal{A}$, along with the proofs $\{\pi_{i3}\}_{i<\epsilon}$.
- $\mathcal{S}_{\mathsf{FS}}$ sets $v' \leftarrow_R \mathbb{F}_p$ and computes $(v'_1, \ldots, v'_n) \leftarrow_R \mathsf{Share}(v')$, computes simulated $\mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}$ proof $\{\pi_{i3} = \mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\{((A_iB_i, \boldsymbol{\gamma}\|\mathbf{1}\|\mathbf{0}, v_i), (\mathbf{s}_i, r_i, \omega_i)) : \mathbf{g}^{\mathbf{s}_i} h_1^{r_i} h_2^{\omega_i} = A_iB_i \wedge \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i = v_i\}$, and sends $\{v_i, \pi_{i3}\}_{i \notin \mathsf{C}}\}_{i>\epsilon}$.
- Finally, $\mathcal{S}_{\mathsf{FS}}$ sends $(v'_i, \pi_{i3})_{i>\epsilon}$ to the adversary $\mathcal{A}$.

We argue correctness of simulation of honest-party first messages $\{A_j\}_{i \notin \mathsf{C}}$ as follows: in the real protocol, the vector of shares for party j is of the form $(f_1(j), \ldots, f_\ell(j))$, where $f_i : i \in [\ell]$ are the polynomials used to share the values $s_i : i \in [\ell]$ respectively. Let $\mathbf{f} = (f_1, \ldots, f_\ell)$ denote the vector of sharing polynomials and let $\mathbf{f}(j)$ to denote the vector $(f_1(j), \ldots, f_\ell(j))$. Then for $j > \epsilon$ in the real protocol, $(A_j)_{j>\epsilon}$ are distributed as $(\mathbf{g}^{\mathbf{f}(j)})_{j>\epsilon}$, subject to constraint that $\mathbf{g}^{\mathbf{f}(0)} = z$. Sampling such a polynomials f_i, $i \in [\ell]$ corresponds to choosing $f_i(1), \ldots, f_i(t)$ uniformly and then determining $f_i(t+j) = \langle (f_i(0), \ldots, f_i(t)), \mathbf{t}_j \rangle$ using the interpolation vector $\mathbf{t}_j$. Thus $\mathbf{f}(t+j)$ is a $\mathbf{t}_j$-linear combination of $\mathbf{f}(0), \ldots, \mathbf{f}(t)$, which dictates simulator's computation of A_{t+j} from vector $\mathbf{a}$. The simulated transcript is an accepting transcript as $\mathbf{g}^{\mathbf{f}(0)} = z$ and $\mathbf{g}^{\mathbf{f}(i)} = A_i$ for all $i \notin \mathsf{C}$, and the verification check is satisfied since a known linear combination of $\{\mathbf{f}(i)\}_{i \notin \mathsf{C}}$ in the exponent yields the desired value $\mathbf{f}(0)$ in the exponent. Additionally, since $\{\mathbf{f}(i)\}_{i \notin \mathsf{C}}$ are implicitly set as the honest-party shares, it is identical to the correct distribution of secret shares. This completes the proof of zero-knowledge for $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$.

We note that knowledge soundness ensures simulation extractability in the random oracle model [31,32], and hence, our Fiat-Shamir transformed round efficient DPoK is simulation-extractable. The following corollary of Theorem 3 follows immediately and yields the concrete bounds on the corruption threshold tolerated by $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$.

Corollary 4. *Setting $d = t < n/3$, $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$ is $n/3$-private and $n/3$-robust.*

5 DPoK for BBS+ Signatures over Secret-Shared Inputs

In this section, we build upon our (publicly verifiable) DPoK for the discrete log relation to design a protocol that allows a prover $\mathcal{P}$ to prove knowledge of a BBS+ (or PS) signature on a secret-shared input. Concretely, suppose that the prover $\mathcal{P}$ holds a BBS+ (or PS) signature σ on a message $\mathbf{m}$ under a public key pk, where $\mathbf{m}$ is secret-shared across n parties $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (i.e. each worker $\mathcal{W}_i$ holds a share $\mathbf{m}_i$). The goal of the protocol is to allow the prover $\mathcal{P}$ to convince a designated verifier $\mathcal{V}$ that σ is a valid signature on $\mathbf{m}$ under pk, *without* revealing σ in the clear (this helps realize the desired property of signature unlinkability, as explained subsequently). We also present similar PoK protocols for PS signatures [43] over secret-shared inputs in Appendix C. Looking ahead, we use these protocols as building blocks to design our compiler for upgrading any secret-sharing based MPC protocol into an authenticated version of the same protocol, where the (secret-shared) inputs are authenticated using BBS+(or PS) signatures as above.

We start by defining the relation for BBS+ signature verification.

Definition 10 (BBS+ Relation). *Let BBSGen denote the relation generator, such that $\mathsf{BBSGen}(1^\lambda, \ell)$ outputs a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R BBS.\mathsf{Setup}(1^\lambda)$. The corresponding relation $\mathcal{R}^{\text{bbs}}$ is defined by $(\mathbf{x}, (\mathbf{m}, \mathbf{t})) \in \mathcal{R}^{\text{bbs}}$ for*
$\mathbf{x} = \mathsf{pk} = (g_1, w, h_0, \ldots, h_\ell) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1^\ell$, $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ and $\mathbf{t} = \sigma = (A, \beta, s) \in \mathbb{G}_1 \times \mathbb{F}_p^2$ if $e(A, wg_2^\beta) = e(g_1 h_0^s \prod_{i=1}^\ell h_i^{m_i}, g_2)$.

Protocol $\Pi_{\mathsf{bbs+}}$

- **Public Key** $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$
- **$\mathcal{P}$'s inputs**: Message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\mathbf{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^\ell h_i^{m_i}\right)^{\frac{1}{\beta+x}}$, such that $(\mathsf{pk}, (\mathbf{m}, \sigma)) \in \mathcal{R}^{\text{bbs}}$
- **$\mathcal{W}_i$'s inputs :** $\mathcal{W}_i$ possesses the i^{th} share $\mathbf{m}_i$ of the message vector $\mathbf{m}$, such that $\mathsf{Reconstruct}(\mathbf{m}_1, \ldots, \mathbf{m}_n) = \mathbf{m}$

– **Pre-processing** : $\mathcal{P}$ samples $u \leftarrow_R \mathbb{F}_p^*, r \leftarrow_R \mathbb{F}_p, \eta \leftarrow_R \mathbb{F}_p$, and computes $d = b^u \cdot h_0^{-r}$ and $t = s - r \cdot v$ where $v = u^{-1}, b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$. $\mathcal{P}$ computes $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$, $(v_1, \ldots, v_n) \leftarrow_R \mathsf{Share}(v)$, $(\beta_1, \ldots, \beta_n) \leftarrow_R \mathsf{Share}(\beta)$, $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$, $(\eta_1, \ldots, \eta_n) \leftarrow_R \mathsf{Share}(\eta)$. $\mathcal{P}$ sends the shares $(r_i, v_i, \beta_i, t_i, \eta_i)$ to $\mathcal{W}_i$, for all $i \in [n]$.
In other words, each $\mathcal{W}_i$ locally holds the i-th share $\mathbf{s}_i = (\mathbf{m}_i, r_i, v_i, \beta_i, t_i, \eta_i)$ such that

$$\mathbf{s} = (\mathbf{m}, r, v, \beta, t) = \mathsf{Reconstruct}\left(\{\mathbf{s}_i\}_{i \in [n]}\right).$$

– **Interactive Protocol:**
1. $\mathcal{P}$ computes $A' = A^u$, $\bar{A} = (A')^{-\beta} \cdot b^u (= (A')^x)$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$ and $d = b^u \cdot h_0^{-r}$. $\mathcal{P}$ sets $C = d^{-v} h_0^{t-\eta}$, $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$, and broadcasts $(A', \bar{A}, d, C, D)$ to each $\mathcal{W}_i$ and $\mathcal{V}$.
2. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the DPoK Π_{dlog} for the relation $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$, where $(\eta, m_1, \ldots, m_\ell)$ are secret-shared across the workers; and $\mathbf{g} = (h_0, \ldots, h_\ell)$, $z = D$ is available to all parties.
3. Simultaneously, the workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the DPoK Π_{dlog} for the relation $C = d^{-v} h_0^{t-\eta} \wedge \frac{\bar{A}}{d} = (A')^{-\beta} h_0^r$, where (v, η) and (β, r) are secret-shared; and $\mathbf{g} = ((d, h_0), (A', h_0))$, $z = (C, \frac{\bar{A}}{d})$ is available to all parties.
4. $\mathcal{V}$ accepts if $C \cdot D = g_1^{-1}$, *and* $e(A', w) = e\left(\bar{A}, g_2\right)$, *and both* instances of Π_{dlog} accept.

5.1 Our DPoK Protocol $\Pi_{\mathsf{bbs+}}$

We build upon the robust complete DPoK Π_{dlog} for discrete log to propose a DPoK achieving robust completeness for BBS+ signatures, which allows a designated prover $\mathcal{P}$, to show knowledge of a BBS+ signature (A, β, s) over the message $\mathbf{m} \in \mathbb{F}_p^{\ell}$ that is secret-shared amongst the workers $\mathcal{W}_1, \ldots, \mathcal{W}_n$. Recall that this PoK involved the following steps: (i) the prover randomly chooses some auxiliary inputs, and combines them with the signature to output a randomized first message (this randomization ensures unlinkability), and then (ii) the prover shows knowledge of these auxiliary inputs and components of the signature satisfying discrete-log relations determined by the first message. Our BBS+ DPoK over secret-shared inputs follows a similar blueprint, where the prover similarly randomizes the first message using certain auxiliary inputs. In our case, the prover: (i) secret-shares the auxiliary inputs to the workers using point-to-point channels (this step is unique to our protocol and is designed to facilitate distributed proving in the subsequent steps), and (ii) broadcasts the first message to the workers *and* the verifier (this step uses broadcast channels and is conceptually similar to the PoK over non-distributed inputs). At this point, the problem reduces to a DPoK for the discrete log relation. We handle this using our robust complete DPoK Π_{dlog} for discrete log.

We prove the $\Pi_{\mathsf{bbs+}}$ to be a DPoK for the relation generator BBSGen in the following theorem.

Theorem 4. *Assuming that* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* $\mathsf{DlogGen}$ *and* (t, n)-SSS, $\Pi_{\mathsf{bbs+}}$ *is a* DPoK *for the relation generator* BBSGen *and* (t, n)-SSS *with:*

– ***Security****: t-private and d-robust, for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n - t)$ is the minimum distance of the Reed-Solomon code induced by (t, n)-SSS.*
– ***Efficiency****: $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

Proof. We provide the proof of security and efficiency below. In order to prove security, we prove robust completeness, soundness, and zero-knowledge.

Robust Completeness. Robust completeness follows from direct calculation using the robust completeness of the underlying subprotocols DPoK Π_{dlog} for $\mathsf{DlogGen}$, used in step (3) and (4).

Knowledge Soundness. Consider an adversary that corrupts a t-sized subset of the workers in $\Pi_{\mathsf{bbs+}}$. By inspection, for $t < n/3$, an honest verifier detects the corrupt subset of workers, since the underlying protocol Π_{dlog} satisfies d-robust completeness for $d < n/3$.

Consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which corrupts $\mathcal{P}$ and $\mathcal{W}_i$, $i \in \mathsf{C}$. We show that, given an extractor Ext for Π_{dlog}, it is possible to design an extraction algorithm Ext' that given $\{\mathbf{m}_i\}_{i \notin \mathsf{C}}$, where $\mathbf{m}_i$ is the share of $\mathbf{m}$ provided to $\mathcal{W}_i$, extracts a signature σ on $\mathbf{m}$. First Ext runs the adversary

$\mathcal{A}$ to obtain the messages $(r_i, v_i, \beta_i, t_i, \eta_i)$ for $i \notin \mathsf{C}$. The extractor Ext' also obtains the message $(A', \bar{A}, d, C, D)$ from $\mathcal{A}$. Next it sets $\mathbf{s}'_i = (\eta_i, \mathbf{m}_i)$ and $\mathbf{s}''_i = (v_i, y_i, \beta_i, r_i)$ for $i \notin \mathsf{C}$ where $y_i = t_i - \eta_i$ for $i \notin \mathsf{C}$. It then invokes the extractor Ext for DPoK sub-protocol Π_{dlog} in steps (2) and (3) respectively and computes the extracted witness as follows:

$$
\begin{aligned}
&(\mathbf{s}')_{i\in\mathsf{C}} = (\eta, \mathbf{m})_{i\in\mathsf{C}} \leftarrow_R \mathsf{Ext}^{\mathcal{A}}(\{\mathbf{s}'_i\}_{i\notin\mathsf{C}}) \\
&(\mathbf{s}'')_{i\in\mathsf{C}} = (v, y, \beta, r)_{i\in\mathsf{C}} \leftarrow_R \mathsf{Ext}^{\mathcal{A}}(\{\mathbf{s}''_i\}_{i\notin\mathsf{C}}) \\
&\text{where} \\
&\eta = \mathsf{Reconstruct}(\eta_1, \ldots, \eta_n), \quad \mathbf{m} = \mathsf{Reconstruct}(\mathbf{m}_1, \ldots, \mathbf{m}_n) \\
&v = \mathsf{Reconstruct}(v_1, \ldots, v_n), \quad y = \mathsf{Reconstruct}(y_1, \ldots, y_n) \\
&\beta = \mathsf{Reconstruct}(\beta_1, \ldots, \beta_n), \quad r = \mathsf{Reconstruct}(r_1, \ldots, r_n)
\end{aligned}
$$

Using the message $(A', \bar{A}, d, C, D)$ obtained from the adversary $\mathcal{A}$ and the outputs $\eta, \mathbf{m}, v, y, \beta, r$ obtained from the extractor Ext for DPoK sub-protocol Π_{dlog}, extracted witness is computed as $(\mathbf{m}, \mathbf{t})$, where $\mathbf{t} = (A'^{v}, \beta, y + \eta + vr)$.

Here, we parse the extracted witness $\mathbf{m}$ as $\mathbf{m} = (m_1, \ldots, m_\ell)$. From knowledge-soundness of the DPoK sub-protocol Π_{dlog} and verifier's checks, with overwhelming probability we have: $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$, $C = d^{-v} h_0^{y}$, $(A')^{-\beta} h_0^{r} = \bar{A}/d$, $C \cdot D = g_1^{-1}$ and $\bar{A} = (A')^{x}$. We first note that $v \neq 0$, otherwise substituting C, D in the relation $C \cdot D = g_1^{-1}$ yields a non-trivial discrete-log relation between the generators $g_1, h_0, \ldots, h_\ell$. From the preceding equations, we can derive:

$$
(A'^{v})^{\beta + x} = g_1 h_0^{y+\eta+vr} \prod_{i=1}^{\ell} h_i^{m_i}
$$

which shows that $(A'^{v}, \beta, y + \eta + vr)$ is a valid signature on $\mathbf{m}$. Hence, the extractor Ext' has computed a valid witness for the BBSGen relation. This completes the proof of knowledge soundness for $\Pi_{\mathsf{bbs+}}$.

Honest Verifier Zero-Knowledge. Finally, consider an adversary $\mathcal{A}$ that corrupts workers $\mathcal{W}_i, i \in \mathsf{C}$ where $|\mathsf{C}| \leq t$. We show that, given a ZK-simulator $\mathsf{Sim}_1^{\mathrm{zk}}$ for Π_{dlog} and a ZK-simulator $\mathsf{Sim}_2^{\mathrm{zk}}$ for the single-prover proof of knowledge for BBS+ signatures from [17] (recalled in Section 2.3), we construct a simulation algorithm Sim' that output a simulated view of an honest verifier in the protocol $\Pi_{\mathsf{bbs+}}$ without the knowledge of the witness $(\mathbf{m}, \sigma)$. Using the simulator $\mathsf{Sim}_2^{\mathrm{zk}}$, the simulator Sim' generates the message $(A', \bar{A}, d, C, D)$. As the statements for the DPoKs in steps (2) and (3) depend entirely on the public parameters and the preceding message, the simulation follows by invoking simulator $\mathsf{Sim}_1^{\mathrm{zk}}$ to simulate the transcript for respective DPoKs on the statements derived from the simulated first message. Looking ahead, in the formal proof of security for our compiled MPC protocol, we use this simulation algorithm Sim' to simulate proofs of knowledge of BBS+ signatures on the inputs of the honest parties. This completes the proof of zero-knowledge soundness for $\Pi_{\mathsf{bbs+}}$.

Proof of Efficiency/Succinctness. Recall that Π_{dlog} has $O(n)$ communication over point-to-point channels and $O(n \log \ell)$-communication overhead over broadcast channel. It follows by inspection that $\Pi_{\mathsf{bbs+}}$ also inherit the same communication overheads from Π_{dlog}. This completes the proof of efficiency for $\Pi_{\mathsf{bbs+}}$, and hence the proof of Theorem 4. □

5.2 Extensions of $\Pi_{\mathsf{bbs+}}$

In this section, we present two extensions of the $\Pi_{\mathsf{bbs+}}$ protocol. The first of these batches several parallel instances of $\Pi_{\mathsf{bbs+}}$ to achieve optimized verification, but only achieves non-robust completeness. The second extension is a round efficient version of $\Pi_{\mathsf{bbs+}}$ which achieves the same security guarantees as $\Pi_{\mathsf{bbs+}}$, albeit in the random oracle model.

Efficiently Batching BBS+ PoKs. We now present the protocol $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ which efficiently batches n parallel instances of the protocol $\Pi_{\mathsf{bbs+}}$ with the party $\mathcal{P}_i$ acting as the prover in the i^{th} instance of the protocol. The optimization exploits the fact that each party needs to prove a linear (in exponents) relation over large part of its witness (the message vector), which can be reduced via a random challenge to proving a linear relation over the linearly combined messages. However we lose robustness: we can no longer identify the corrupt parties or a corrupt prover using error-correction as in $\Pi_{\mathsf{bbs+}}$, as the combined witness cannot be attributed to a specific party. Thus, we simply abort if one of the checks in the underlying protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ fails.

Protocol $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$

- **Public Parameters**: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R \mathsf{BBSGen}(1^\lambda)$ defining BBS+ relation $\mathcal{R}^{\text{bbs}}$. Let $\mathsf{pk} = (g_1, w = g_2^x, h_0, \ldots, h_\ell)$ be a known public key for secret key $\mathsf{sk} = x \leftarrow_R \mathbb{F}_p$.
- P_i**'s inputs**:
  - Message $\mathbf{m}_i \in \mathbb{F}_p^\ell$ and signature $\sigma_i = (A_i, \beta_i, s_i)$ on $\mathbf{m}_i$ under pk.
  - i^{th} share of the message $\mathbf{m}_j$ of P_j.
- **Pre-processing**: $\mathcal{P}_i$ samples $u_i \leftarrow_R \mathbb{F}_p^*, r_i \leftarrow_R \mathbb{F}_p, \eta_i \leftarrow_R \mathbb{F}_p$, and computes $d_i = b_i^{u_i} \cdot h_0^{-r_i}$ and $t_i = s_i - r_i \cdot v_i$ where $v_i = u_i^{-1}, b_i = g_1 h_0^{s_i} \prod_{i=1}^{\ell} h_i^{m_i}$. and secret shares $r_i, v_i, t_i, \eta_i, \beta_i$ among $P_1, \ldots, P_n$. All parties set $\mathbf{g} = (h_0, \ldots, h_\ell)$.
- **Interactive Protocol**
  1. $\mathcal{P}_i$, $i \in [n]$ computes $A_i' = A_i^{u_i}$, $\bar{A}_i = (A_i')^{-\beta} \cdot b_i^u (= (A_i')^x)$. $\mathcal{P}$ sets $C_i = d_i^{-v_i} h_0^{t_i - \eta_i}$, $D_i = \mathbf{g}^{\eta_i, \mathbf{m}_i}$, and broadcasts $(A_i', \bar{A}_i, d_i, C_i, D_i)$.
  2. The verifier samples a challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$ and broacasts it. Each P_i then computes $\mathbf{y}_i = \sum_{j \in [n]} \gamma^j (\eta_{ij}, \mathbf{m}_{ij})$, where $\eta_{ij}, \mathbf{m}_{ij}$ denotes $\mathcal{P}_i$'s share of $\mathcal{P}_j$'s inputs $\mathbf{m}_j, \eta_{ij}$.
  3. All parties compute $D = \prod_{j \in [n]} D_j^{\gamma^j}$.

     Parties hold shares $\mathbf{y}_i$ of $\mathbf{y}$ satisfying $\mathbf{g}^{\mathbf{y}} = D$

  4. Parties run the interactive phase of the protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ on statement D with $\mathbf{g}$ as the generator. They run the interactive phase of the protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ on statements $C_i = d_i^{-v_i} h_0^{t_i - \eta_i} \wedge \frac{\bar{A}_i}{d_i} = (A_i')^{-\beta_i} h_0^{r_i}$, for each $i \in [n]$ with generators (d_i, h_0) and (A_i', h_0) respectively.
  5. Parties also check that $e\left(\prod_{i=1}^n A_i', w\right) = e\left(\prod_{i=1}^n \bar{A}_i, g_2\right)$ holds.
- **Output**: P_j outputs $b_j = 1$ if all the above protocols lead to accept.

Round Efficient DPoK for BBS+ Signatures. Finally, note that by replacing Π_{dlog} with its round efficient version $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$ (obtained using the Fiat-Shamir heuristic, presented in Section 4.2) in steps (2) and (3) of the Interactive Phase, we obtain a round efficient version of the protocol, which we call $\Pi_{\mathsf{bbs+}}^{\mathsf{FS}}$. Observe that $\Pi_{\mathsf{bbs+}}^{\mathsf{FS}}$ requires constant rounds of interaction, as compared to logarithmic (in the size of the message) rounds of interaction for $\Pi_{\mathsf{bbs+}}$, and satisfies the same robust completeness, knowledge soundness and zero-knowledge properties as $\Pi_{\mathsf{bbs+}}$, albeit in the random oracle model.

6 Compiler for Authenticated MPC

In this section we present our compiler for MPC with input authentication that outputs an MPC protocol where each input is authenticated using a BBS+ signature under a common (public) verification key. In Appendix C, we outline a similar compiler based on PS signatures.

Class of MPC Protocols. Our compiler takes advantage of the observation that a large class of secret-sharing based MPC protocols share the following template. (i) There is an input sharing phase where parties secret-share their inputs, and (ii) when using secret sharing schemes with certain thresholds ($t_{\mathsf{sh}} < |H|$), the input of parties is completely determined at the end of the input sharing phase. This means that using inputs inconsistent with this sharing is considered deviating, against which the protocol is secure. This is precisely where our compiler performs well: verification of authenticity (or any other predicate) on the inputs can be done fully outside the MPC by running a DPoK on the shares. (iii) For an MPC protocol of this template, there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$, where $\mathsf{Sim}_{\mathsf{sh}}$ deterministically extracts the inputs of corrupt parties, and $\mathsf{Sim}_{\mathsf{on}}$ simulates the protocol view.

Features of Our Compiler. Our compiler allows identification of all (malicious) parties with non-authenticated inputs (this is a consequence of the robust completeness property of Π_{dlog} used inside $\Pi_{\mathsf{bbs+}}$). We further note that our robust protocol Π_{dlog} tolerates a maximum corruption threshold of $t < n/3$ (assuming that the secret-sharing used is Shamir's secret sharing). Hence, our compiled MPC protocol also tolerates a maximum corruption threshold of $t < n/3$. Using the non-robust version will result in a non-robust compiler that retains the $t < n/2$ threshold of the underlying MPC.

The Desired Ideal Functionality. We define below the desired ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{authid}}$ for MPC with input authentication.

Functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$

Inputs

The ideal functionality receives from each party P_i an input-signature pair of the form $(\mathbf{x}_i, \sigma_i)$ under the public verification key pk.

Verify Authenticity

1. If $\mathsf{Ver}(\mathsf{pk}, x_i, \sigma_i) \neq 1$ for some party P_i, then output a set of corrupted parties C and abort.
2. Otherwise, proceed to computation.

Computation

Invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ for Π_{mpc} on inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

6.1 Our Compiler

We now present a formal description of our compiler. Let $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$ be a secret-sharing based MPC protocol that guarantees security with abort against malicious corruptions of a dishonest majority of the parties $\{P_1, \ldots, P_n\}$, where:

- Π_{sh} denotes the secret-sharing phase of Π_{mpc} and consists of the steps used by each party P_i for $i \in [n]$ to secret-share its input $\mathbf{x}_i \in \mathbb{F}_p^\ell$ to all of the other parties (throughout, we assume that this sharing is done using a linear secret-sharing scheme ($\mathsf{Share}, \mathsf{Reconstruct}$).
- Π_{on} denotes the remaining steps of the protocol Π_{mpc} where the parties interact to compute $y = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$

- **Inputs:** All parties hold public parameters and the verification key pk of a BBS+ signature scheme. Party P_i has input $\mathbf{x}_i \in \mathbb{F}_p^\ell$, together with a signature σ_i, such that $(\mathsf{pk}, (\mathbf{x}_i, \sigma_i)) \in \mathcal{R}^{\mathrm{bbs}}$.
- $\overline{\Pi}_{\mathsf{sh}}$: This phase is identical to Π_{sh}, i.e., each party P_i shares its input $\mathbf{x}_i$ to all other parties exactly as in Π_{sh}.
- $\overline{\Pi}_{\mathsf{on}}$: In this phase, the parties do the following:
  - For each $j = 1, \ldots, n$, the parties execute an instance of $\Pi_{\mathsf{bbs+}}$ for $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \in \mathcal{R}^{\mathrm{bbs}}$ with $\mathcal{P}_j$ acting as the Prover, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ constituting the workers and $\mathcal{P}_i$, $i \neq j$ acting as verifiers, . If any party outputs 0 at the end of this phase, the protocol aborts.
  - Otherwise, the parties jointly execute Π_{on}.

In the description of our compiler, we assume that each party P_i holds a BBS+ signature σ_i on its input $\mathbf{x}_i$ with respect to a common public verification key pk. The compiler runs n instances of $\Pi_{\mathsf{bbs+}}$, where for instance i, party P_i acts as the prover and all other parties P_j for $j \neq i$ act as verifiers. Given $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$, our robust compiler outputs an authenticated MPC protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$. The compiler Π_{ampc} is described above.

Theorem 5 (Security of Π_{ampc}). *Assuming that: (a) the MPC protocol Π_{mpc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, and (b) Π_{dlog} is a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for relation generator $\mathsf{DlogGen}$ and (t, n)-SSS our compiled MPC protocol with input authentication Π_{ampc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ for the same corruption threshold of $t < n/3$.*

Proof. We construct a simulator for the Π_{ampc} protocol, and prove indistinguishability of the simulation from a real-world execution of Π_{ampc}. The underlying MPC protocol Π_{mpc} secure emulates $\mathcal{F}_{\mathsf{MPC}}$, and let $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$ be the corresponding simulator.

Simulator for Π_{ampc}. We now describe the simulator $\overline{\mathsf{Sim}}$ for the authenticated MPC protocol $\Pi_{\mathsf{ampc}} = \big(\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}}\big)$. Let $\mathcal{H} \subseteq [n]$ and $\mathcal{C} \subset [n]$ denote the set of honest and corrupt parties, respectively. The simulator $\overline{\mathsf{Sim}}$ proceeds as follows:

1. Simulate the sharing phase $\overline{\Pi}_{\mathsf{sh}}$ of the underlying MPC Π_{mpc} by invoking $\mathsf{Sim}_{\mathsf{sh}}$ (note that $\mathsf{Sim}_{\mathsf{sh}}$ does not expect any inputs). $\overline{\mathsf{Sim}}$ receives the ith share $\{\mathbf{s}_i^j\}_{i\in\mathcal{H}}$ from the adversary (invoked by $\mathsf{Sim}_{\mathsf{sh}}$) corresponding to the input $\mathbf{s}^j$ of each corrupt party $P_j, j \in \mathcal{C}$.
2. For each P_j s.t. $j \in \mathcal{C}$, let $(\Pi_{\mathsf{bbs+}})_j$ denote the instance of the protocol $\Pi_{\mathsf{bbs+}}$ used by the parties where P_j acts as the prover, and all of the remaining parties acting as both workers and verifiers. The simulation of the online phase proceeds as follows.
   (a) First, the simulator of the online phase invokes the simulator of the underlying DPoK $\Pi_{\mathsf{bbs+}}$ to simulate the proofs of knowledge of BBS+ signatures on the inputs of the honest parties.
   (b) For each instance $\Pi_{\mathsf{bbs+}}$, where a corrupt party P_j, $j \in \mathcal{C}$ is acting as the prover, invoke the extractor Ext' of the DPoK $\Pi_{\mathsf{bbs+}}$ on the shares of the honest parties $(\mathbf{s}_i^j)_{i\in\mathcal{H}}$ corresponding to the corrupt party P_j's input to extract the witness $(\mathbf{x}_j, \sigma_j)$ from P_j. Note that since we assume honest-majority, the shares $\{\mathbf{s}_i^j\}_{i\in\mathcal{H}}$ given as input to the extractor Ext' completely determines the respective inputs of each corrupt party $P_j, j \in \mathcal{C}$. Hence, the compiler aborts if $\mathsf{Consistent}(\mathbf{x}_j, \{\mathbf{s}_i^j\}_{i\in\mathcal{H}}) = 0$.
   (c) Invoke $\mathsf{Sim}_{\mathsf{on}}$ to simulate the online phase of the underlying MPC Π_{mpc}.
3. Send $\{(\mathbf{x}_j, \sigma_j)\}_{j\in\mathcal{C}}$ to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$. If $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ aborts by identifying some subset of corrupt parties, abort while identifying the same subset of corrupt parties; otherwise output whatever $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ outputs.

Completing the Security Proof. We now prove the security of Π_{ampc} by using a sequence of hybrids described as follows (for simplicity of exposition, we assume w.l.o.g. that parties $P_1, \ldots, P_{|\mathcal{C}|}$ are corrupt and parties $P_{|\mathcal{C}|+1}, \ldots, P_n$ are honest):

- Hyb_0: This hybrid is identical to the real-world execution of Π_{ampc}.
- Hyb_1: This hybrid is identical to Hyb_0 except that we simulate the sharing phase $\overline{\Pi}_{\mathsf{sh}}$ of the underlying Π_{mpc} protocol by invoking $\mathsf{Sim}_{\mathsf{sh}}$. Receive from $\mathsf{Sim}_{\mathsf{sh}}$ the set of shares $\{\mathbf{s}_i^j\}_{i\in\mathcal{H}}$ corresponding to the input $\mathbf{s}^j$ of each corrupt party $P_j, j \in \mathcal{C}$.
- $\{\mathsf{Hyb}_{2,j}\}_{j\in[0,n-|\mathcal{C}|]}$: Hybrid $\mathsf{Hyb}_{2,0}$ is identical to hybrid Hyb_1, and for each $j \in [1, n-|\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{2,j}$ is identical to $\mathsf{Hyb}_{2,(j-1)}$ except that proof of knowledge corresponding to the input of honest party $P_{|\mathcal{C}|+j}$ is simulated using Sim' as described in Step 2(a) of the simulator. More concretely, for each honest party $P_{|\mathcal{C}|+j}$, instead of using the real input $\mathbf{x}_{|\mathcal{C}|+j}$ and the real BBS+ signature $\sigma_{|\mathcal{C}|+j}$, proof of knowledge of a BBS+ signature is simulated instead of running an instance of the protocol $\Pi_{\mathsf{bbs+}}$ where party $P_{|\mathcal{C}|+j}$ is the prover.
- $\{\mathsf{Hyb}_{3,j}\}_{j\in[0,|\mathcal{C}|]}$: The first of these hybrids, i.e., Hybrid $\mathsf{Hyb}_{3,0}$ is identical to hybrid $\mathsf{Hyb}_{2,n-|\mathcal{C}|}$. Next, for each $j \in [1, |\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{3,j}$ is identical to $\mathsf{Hyb}_{3,(j-1)}$ except that we abort if the following bad event occurs: for the instance of $\Pi_{\mathsf{bbs+}}$ where the corrupt party P_j is the prover, invoke the extractor Ext' (as mentioned in Step 2(b) of the simulator and described in the proof overview) on the shares of the honest parties $(\mathbf{s}_i^j)_{i\in\mathcal{H}}$ corresponding to the corrupt party P_j's input to extract the witness $(\mathbf{x}_j, \sigma_j)$ from P_j. If $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \notin \mathcal{R}^{\mathrm{bbs}}$ or $\mathsf{Consistent}(\mathbf{x}_j, \{\mathbf{s}_i^j\}_{i\in\mathcal{H}}) = 0$, then abort.
- Hyb_4: This hybrid is identical to $\mathsf{Hyb}_{3,|\mathcal{C}|}$ except for the following: invoke $\mathsf{Sim}_{\mathsf{on}}$ of the underlying Π_{mpc} protocol to simulate the online phase $\overline{\Pi}_{\mathsf{on}}$, and output whatever $\mathsf{Sim}_{\mathsf{on}}$ outputs.
- Hyb_5: This hybrid is identical to Hyb_4 except that after invoking $\mathsf{Sim}_{\mathsf{on}}$ to simulate $\overline{\Pi}_{\mathsf{on}}$, we query $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ with the extracted inputs $\{(\mathbf{x}_j, \sigma_j)\}_{j\in\mathcal{C}}$.

$\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$. This follows from the security of the underlying Π_{mpc} protocol. Suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_0 and Hyb_1. It is easy to use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ that can distinguish between a real and simulated execution of Π_{sh}, thus breaking security of the underlying Π_{mpc} protocol.

$\mathsf{Hyb}_{2,j-1} \approx_c \mathsf{Hyb}_{2,j}$. This follows from the ZK property of Π_{dlog} and the PoK for single-prover version of BBS+ signatures. In particular, suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{2,(j-1)}$ and $\mathsf{Hyb}_{2,j}$ for some $j \in [1, n-|\mathcal{C}|]$. Then $\mathcal{A}$ can be used to construct one of the following algorithms: (a) either an adversary $\mathcal{A}'$ that breaks the ZK property of the Π_{dlog} protocol, or (b) an adversary $\mathcal{A}''$ that breaks the ZK property of the PoK for single-prover version of BBS+ signatures.

$\mathsf{Hyb}_{3,j-1} \approx_c \mathsf{Hyb}_{3,j}$. This follows from knowledge soundness of Π_{dlog}. The two hybrids differ only when the bad event occurs, i.e., the extractor Ext' in Step 2(b) of the simulator fails to output a valid witness $(\mathbf{m}, \sigma)$ where $\mathbf{m}$ is consistent with the honest party shares. However, as described in the proof overview, assuming the knowledge-soundness of Π_{dlog}, the extractor Ext' outputs a valid witness. Hence, assuming knowledge-soundness of Π_{dlog}, the probability of the bad event occurring must be negligible.

$\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_{3,|\mathcal{C}|}$. This follows from the security of the underlying Π_{mpc} protocol. At the end of Π_{sh}, if abort did not occur, then for each $i \in [n]$, all honest parties hold shares $\langle \mathbf{x}'_j \rangle_{j \in \mathcal{H}}$ of some $\mathbf{x}'_i \in \mathbb{F}^\ell$. In $\mathsf{Hyb}_{3,|\mathcal{C}|}$, the extractor succeeds in outputting a valid witness $\mathbf{x}_i$, and this is the unique $\mathbf{x}'_i$ determined at the end of Π_{sh}. Suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_4 and $\mathsf{Hyb}_{3,|\mathcal{C}|}$. It is easy to use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ that can distinguish between a real and simulated execution of Π_{on}, thus breaking the security of the underlying Π_{mpc} protocol.

$\mathsf{Hyb}_5 \equiv \mathsf{Hyb}_4$. Hyb_5 and Hyb_4 are identical. In Hyb_4, the output of is given by the output of $\mathsf{Sim}_{\mathsf{on}}$ and in Hyb_5, the output is given by the output of $\mathsf{Sim}_{\mathsf{sh}}$, which are idential by the security of the underlying Π_{mpc}. We also note that Hyb_5 is identical to $\overline{\mathsf{Sim}}$.

This completes the proof of Theorem 5. □

Round Efficient Compiler for Authenticated MPC. Finally, it is easy to see that invoking the round efficient DPoK $\Pi^{\mathsf{FS}}_{\mathsf{bbs+}}$ protocol instead of the DPoK $\Pi_{\mathsf{bbs+}}$ protocol enables us to obtain a round efficient version of our compiler. The round efficient version achieves the same security guarantees as the compiler presented above, albeit in the random oracle model.

6.2 Resistance to Known Vulnerabilities

We conclude this section with a discussion on why our proposed DPoK protocols and our compiler for authenticated MPC resist some known attacks and insecurities of ZKP protocols in practice.

Resistance to ROS Attacks. In [10], the authors presented an algorithm for solving ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) mod p in polynomial time for $\ell > \log p$ dimensions, which leads to the ROS attack on certain advanced families of digital signatures which involve computations over secret shares. However, the ROS attack does not apply to our proposed DPoK protocols. In particular, note that the ROS attack only works when: (i) there are more than $\log p$ parallel sessions for the same shares, (ii) the adversary chooses its first message after seeing all of the other first messages from the honest parties, (iii) the adversary chooses the challenge.

The ROS attack is not applicable for our protocols as: (i) there are no parallel sessions in our protocols, (ii) each protocol is instantiated using the output of (the randomized) Share algorithm of the underlying secret sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$, thereby ensuring that we do not reuse the shares across sessions, and in the round-efficient versions of our proposed protocols: (iii) the parties send non-interactive proofs instead of sending the first-messages separately (see $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ in Section 4.3), and finally (iv) the challenge is not chosen by the adversary (verifier); it is determined by performing a hash of the available public transcript.

Resistance to OSNARK-related Vulnerabilities. In [29], the authors provide a study of when SNARKs are insecure in the presence of certain oracles (in particular, the knowledge soundness guarantees do not hold in such settings since the extraction fails). As defined in [29], an OSNARK is a SNARK that guarantees extraction even in presence of an oracle for the prover. We note here that the negative result for the existence of OSNARKs, as outlined in [29], does not provide a general impossibility result, since it only applies either to SNARKs where the prover has access to oracles with secret states (such that the extractor does not have access to these states), and for standard-model SNARKs. We note that the attack does not apply: (i) to SNARKs in the ROM, and (ii) when the extractor is black-box in the adversary. Fiat-Shamir transformed Sigma protocols are also known to satisfy black-box *simulation-extractability*, i.e., knowledge soundness holds even in the presence of proof oracles [31,32]. Analogously, our Fiat-Shamir transformed round-efficient proofs of knowledge are simulation-extractable in the random oracle model, as we establish through formal proofs of security. In particular, there are no other oracles with secret states in our setting. We emphasize that signatures are already independently obtained by the parties on their inputs, and signing or signature-oracles are not included as part of our authenticated MPC protocols.

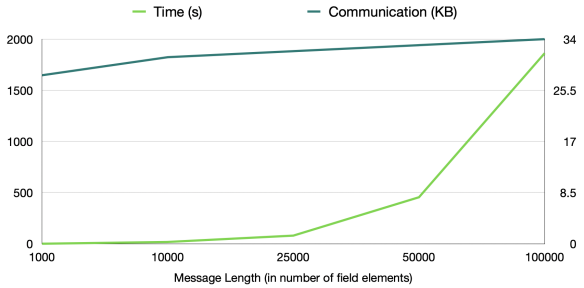

Fig. 1: Evaluation of communication and computational overheads incurred by our implementation of $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ for 5 parties.

# Parties	# Rows	Vanilla MPC		DPoK Overhead	
		Comm(MB)	Time (s)	Comm.(KB)	Time (s)
3	100	1733	6.67	13	0.519
	1000	16754	64	15	18
	2000	33398	129	15.3	65
	4000	66502	260	15.8	246
5	100	8838	26	28	0.643
	1000	87747	265	31	20
	2000	175671	521	32	76
	4000	350658	958	33	312

Table 2: Benchmarks for the secure KPI application with 3 and 5 parties. The second column titled "Rows" indicates the number of rows in each party's dataset (the number of columns is fixed to 10).

7 Implementation and Evaluation

In this section, we present a prototype implementation of our compiler using $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ for BBS+ signatures. We test and benchmark our implementation on a 16GB system with Intel Core i5-9400 CPU clocked at 2.9GHz and running Ubuntu Linux 20.04. All the benchmarks use single execution thread. We use the implementation of `BN128` elliptic curve from the library `libff` [45] to implement $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ with $(t, 2t+1)$-Shamir secret sharing[12]. We then integrate our implementation of $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ with a maliciously secure implementation of Shamir-secret sharing-based MPC from the well-known `MP-SPDZ` library [38] to obtain an implementation of authenticated MPC[13].

Our implementation exploits the modularity of our compiler in the following manner. We first use the MP-SPDZ to share the parties inputs and export the shares held by each party locally. MP-SPDZ provides a convenient interface `write-to-file()` which allows exporting the shares of the secret object held by the party. Next, we use the exported shares to run an instance of $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$. If the $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ fails, we abort the protocol. If the above succeeds, we continue the online phase of the protocol. In MP-SPDZ, the call `read-from-file()` allows one to start the online phase of the protocol using the shares of the object exported in the first step. This seamless integration with existing MP-SPDZ programs illustrates the generality of our approach.

Evaluation and Discussion. We benchmark both $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ (in a standalone manner) and the final authenticated MPC protocol (obtained by integrating $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ with MP-SPDZ [38] as specified in our compiler) in the setting of the industry KPI application outlined in the introduction. In Figure 1, we illustrate the overheads incurred by a standalone implementation of $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ for varying input sizes with 5 parties. We then consider two instances of the KPI application, with 3 and 5 parties, where each party's dataset has 10 columns and variable number of rows (betwern 100 and 4000). We summarize the overheads for vanilla unauthenticated computation using MP-SPDZ, as well as the additional overheads incurred by the compiled authenticated MPC, in Table 2. It is readily apparent that the communication overhead of input authentication over vanilla MPC are minimal. The computational overhead grows with input size, which is unavoidable to an extent, as BBS+ signature verification involves algebraic

[12] We do not implement broadcast functionality cryptographically. To obtain the benchmarks we implement a server acting as a broadcast hub. Efficient broadcast can be implemented for our setting based on [33].

[13] We have submitted an anonymized version of our code repository under "Additional materials" on the submission website.

operations that grow with the size of the input. The major contributor to the computational overheads are the instances of NIPK, which may be parallelized for large input sizes. We leave such optimized implementations as interesting future work.

References

1. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.
2. Diego F. Aranha, Anders P. K. Dalskov, Daniel Escudero, and Claudio Orlandi. Improved threshold signatures, proactive secret sharing, and input certification from LSS isomorphisms. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912, pages 382–404, 2021.
3. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.
4. Thomas Attema, Serge Fehr, and Michael Klooß. Fiat–shamir transformation of multi-round interactive proofs (extended version). *J. Cryptol.*, 36(4), aug 2023.
5. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 111–125. Springer, Heidelberg, September 2006.
6. Carsten Baum. On garbling schemes with and without privacy. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 468–485. Springer, Heidelberg, August / September 2016.
7. Carsten Baum, Robin Jadoul, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. Feta: Efficient threshold designated-verifier zero-knowledge proofs. Cryptology ePrint Archive, Paper 2022/082, 2022. `https://eprint.iacr.org/2022/082`.
8. Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st FOCS*, pages 900–909. IEEE Computer Society Press, November 2020.
9. 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, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.
10. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021.
11. Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. Efficient adaptively-secure byzantine agreement for long messages. Cryptology ePrint Archive, Paper 2021/1403, 2021. `https://eprint.iacr.org/2021/1403`.
12. Marina Blanton and Fattaneh Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PoPETs*, 2016(4):144–164, October 2016.
13. Marina Blanton and Myoungin Jeong. Improved signature schemes for secure multi-party computation with certified inputs. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 438–460. Springer, Heidelberg, September 2018.
14. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.
15. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97. Springer, Heidelberg, August 2019.
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, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.
17. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *TRUST 2016*, volume 9824, pages 1–20. Springer, 2016.
18. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.
19. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002.
20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020.

21. Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI 2017*, pages 259–282. USENIX Association, 2017.
22. Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.
23. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.
24. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO*, pages 572–590, 2007.
25. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.
26. Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any NP statement jointly? efficient distributed-prover zero-knowledge protocols. *Proc. Priv. Enhancing Technol.*, 2022(2):517–556, 2022.
27. Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Report 2022/062, 2022. `https://eprint.iacr.org/2022/062`.
28. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.
29. Dario Fiore and Anca Nitulescu. On the insecurity of snarks in the presence of oracles. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985*, page 108–138, Berlin, Heidelberg, 2016. Springer-Verlag.
30. 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`.
31. Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat–shamir zksnarks (updatable srs) simulation extractable? Cryptology ePrint Archive, Paper 2021/511, 2021. `https://eprint.iacr.org/2021/511`.
32. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the random oracle model). Cryptology ePrint Archive, Paper 2023/147, 2023. `https://eprint.iacr.org/2023/147`.
33. Chaya Ganesh and Arpita Patra. Broadcast extensions with optimal communication and round complexity. In George Giakkoupis, editor, *35th ACM PODC*, pages 371–380. ACM, July 2016.
34. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.
35. Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. Your reputation's safe with me: Framing-free distributed zero-knowledge proofs. Cryptology ePrint Archive, Paper 2022/1523, 2022. `https://eprint.iacr.org/2022/1523`.
36. Jonathan Katz, Alex J. Malozemoff, and Xiao Wang. Efficiently enforcing input validity in secure two-party computation. Cryptology ePrint Archive, Report 2016/184, 2016. `https://ia.cr/2016/184`.
37. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1575–1590. ACM Press, November 2020.
38. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.
39. Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 549–560. ACM Press, November 2013.
40. Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The bbs signature scheme. Internet Engineering Task Force, 2022. `https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html`.
41. Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Report 2021/1530, 2021. `https://eprint.iacr.org/2021/1530`.
42. Torben Pryds Pedersen. Distributed provers with applications to undeniable signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 221–242. Springer, Heidelberg, April 1991.
43. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.
44. 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*, volume 9696 of *LNCS*, pages 346–366. Springer, Heidelberg, June 2016.

45. MIT SCIPR Lab. libff: C++ library for finite fields and elliptic curves. https://github.com/scipr-lab/libff, 2023. `https://github.com/scipr-lab/libff`.
46. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
47. Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018.
48. Yihua Zhang, Marina Blanton, and Fattaneh Bayatbabolghani. Enforcing input correctness via certification in garbled circuit evaluation. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 552–569. Springer, Heidelberg, September 2017.

A Generalization to Threshold Linear Secret Sharing Scheme

In this section, we provide generalization of our technique shown for Shamir Secret Sharing [46] to any Threshold Linear Secret Sharing Scheme. Here we present the definition of Threshold Linear Secret Sharing (TLSS) Scheme, which is a restriction of the definition of Linear Secret Sharing Scheme provided in [22, Chapter 6] to the case when each party receives same number of shares.

Definition 11 (Threshold Linear Secret Sharing Scheme). *A (t, n, r) threshold linear secret-sharing* (TLSS) *scheme over a finite field $\mathbb{F}$ consists of algorithms* (Share, Reconstruct) *as described below:*

- Share *is a randomized algorithm that is defined by a $m \times (t+1)$ matrix M (for some $m \geq n$) and a labeling function $\phi : [m] \to [n]$ such that $|\phi^{-1}(i)| = r$ for all $i \in [n]$. On input $s \in \mathbb{F}$,* Share *samples $r_1, \ldots, r_t \leftarrow_R \mathbb{F}$ uniformly and independently and sets $\mathbf{r}_s = (s, r_1, \ldots, r_t)$. It sets $\mathbf{s}_i = \{(M\mathbf{r}_s)_j \ : \ \phi(j) = i\}$ as the i^{th} share for all $i \in [n]$. We denote the output as $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(s)$, where $\mathbf{s}_i \in \mathbb{F}^r$ is the share sent to i^{th} party.*
- Reconstruct *is a deterministic algorithm that takes a set $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| > t$, a vector of shares $(\mathbf{s}_1, \ldots, \mathbf{s}_{|\mathcal{I}|})$ and outputs $s = \mathsf{Reconstruct}((\mathbf{s}_1, \ldots, \mathbf{s}_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$. Specifically, for all sets $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| > t$, there exists a vector $\mathbf{k}_{\mathcal{I}} = (k_{11}, \ldots, k_{nr}) \in \mathbb{F}^{nr}$ such that $\mathbf{s} = \sum_{i=1}^{n} \sum_{j=1}^{r} k_{ij} s_{ij}$. Here $\mathbf{s}_i = (s_{i1}, \ldots, s_{ir})$ for $i \in [n]$.*

A TLSS *scheme satisfies the following properties:*

- **Correctness***: For every $s \in \mathbb{F}$, any $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q > t$, we have $\mathsf{Reconstruct}((\mathbf{s}_{i_1}, \ldots, \mathbf{s}_{i_q}), \mathcal{I}) = s$.*
- **Privacy***: For every $s \in \mathbb{F}$, any $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q \leq t$, the tuple $(\mathbf{s}_{i_1}, \ldots, \mathbf{s}_{i_q})$ is information-theoretically independent of s.*

Remark 5. We focus on Threshold Linear Secret Sharing schemes in this section, and we denote it as TLSS. As before we can extend a TLSS scheme to secret-share vectors $\mathbf{s} \in \mathbb{F}^\ell$ by applying Share, Reconstruct algorithms component-wise.

A.1 Robust DPoK for Discrete Log for TLSS

In this section we generalize the construction of robust complete protocol for discrete-log relation presented in Section 3.2 to the case when (Share, Reconstruct) can be an arbitrary TLSS scheme. We also characterize the robustness threshold for the same in terms of minimum distance of linear code associated with the TLSS scheme. The proof of robust completeness now depends on Lemma 3 (below), which generalizes Lemma 2 to the case when linear code is over an extension field $\mathbb{F}_{p^r} \cong \mathbb{F}_p^r$ of the field $\mathbb{F} = \mathbb{F}_p$.

Let DlogGen be a relation generator that on input $(1^\lambda, m)$ outputs $(\mathbb{G}, \mathbf{g}, p)$ where p is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\mathbf{g} = (g_1, \ldots, g_m) \leftarrow_R \mathbb{G}^m$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\mathbf{g}^{\mathbf{s}} = z$. Let TLSS = (Share, Reconstruct) denote (t, n, r) threshold linear secret sharing over finite field of order p $\mathbb{F} = \mathbb{F}_p$. We follow the framework presented for DlogGen; namely Π_{dlog} (Figure 3.2), that is t-private, d-robust and incurs $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels. We present our generalized protocol with the similar guarantees.

Additional Preliminaries and Notation. We setup some useful notation and preliminaries specific to this section to ease the presentation. For $s \in \mathbb{F}$, we will view the output $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ to consist of n-shares each over $\mathbb{F}_{p^r}$, i.e. we view $s_i \in \mathbb{F}^r$ as an element of $\mathbb{F}_{p^r}$. Applying the sharing component-wise, for a vector $\mathbf{s} \in \mathbb{F}^\ell$, we view the output $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$ to consist of n-shares, each in $(\mathbb{F}_{p^r})^\ell$, i.e an ℓ-length vector over $\mathbb{F}_{p^r}$. We also veiw a vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in (\mathbb{F}_{p^r})^\ell$ as $\ell \times r$ matrix over $\mathbb{F}$, where i^{th} row of the matrix corresponds to $s_i \in \mathbb{F}_{p^r}$ viewed as a vector in $\mathbb{F}^r$. We also introduce the linear code $\mathcal{L}_{\mathsf{TLSS}}$, which is induced by the sharings under the TLSS scheme.

Definition 12 (TLSS induced code). *For an (n, t, r)-TLSS scheme over $\mathbb{F}$ given by algorithms $(\mathsf{Share}, \mathsf{Reconstruct})$, we define linear code $\mathcal{L}_{\mathsf{TLSS}}$ over the field $\mathbb{F}_{p^r}$ as*

$$\mathcal{L}_{\mathsf{TLSS}} = \{(s_1, \ldots, s_n) : \Pr\left[(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s), s \leftarrow_R \mathbb{F}\right] > 0\},$$

consisting of all possible sharings output by the Share algorithm.

We now state the generalization of Lemma 2 to fields of the form $\mathbb{F}_{p^r}$. The lemma is proved in [26][Lemma A.5]. We recall that for an $[n, k, *]$ linear code $\mathcal{L}$ over $\mathbb{F}$, $\mathcal{L}^m$ denotes the set of $m \times n$ matrices over $\mathbb{F}$ whose rows are codewords in $\mathcal{L}$.

Lemma 3. *Let $\mathcal{L}$ be an $[n, k, d]$-linear code over finite field $\mathbb{F}_{p^k}$ and let $\mathbf{S}$ be an $m \times n$ matrix over $\mathbb{F}_{p^k}$. Let $e = \Delta(\mathbf{S}, \mathcal{L}^m)$ be such that $e < d/3$. Then for any codeword $\mathbf{r} \in \mathcal{L}$, and $\boldsymbol{\gamma}$ sampled uniformly from $\mathbb{F}^m$, we have $\Delta(\mathbf{r} + \boldsymbol{\gamma}^T\mathbf{S}, \mathcal{L}) = e$ with probability at least $1 - d/|\mathbb{F}|$. Furthermore, if E denotes the column indices where $\mathbf{S}$ differs from the nearest matrix $\mathbf{Q}$ in $\mathcal{L}^m$, with probability $1 - d/|\mathbb{F}|$ over choice of $\boldsymbol{\gamma}$, the vector $\mathbf{r} + \boldsymbol{\gamma}^T\mathbf{S}$ differs from the closest codeword $\mathbf{v} \in \mathcal{L}$ at precisely the positions in E.*

We now proceed with the description of the generalised protocol, where we highlight key differences from the protocol Π_{dlog} for the case of Shamir Secret Sharing.

1. *Public Parameters*: The public parameters, as before consists of $(\mathbb{G}, \mathbf{g}, p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, \ell)$. Additionally we have $h_1, h_2 \leftarrow_R \mathbb{G}$. The relation $\mathcal{R}^{\mathrm{DL}}$ consists of $(z, \mathbf{s})$ satisfying $\mathbf{g}^{\mathbf{s}} = z$.
2. *Input Phase*: The prover gets $(z, \mathbf{s})$ while workers $\mathcal{W}_i$,$i \in [n]$ are given $(z, \mathbf{s}_i)$ where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$.
3. *Pre-processing:* The prover sends δ_i to $\mathcal{W}_i$ for $i \in [n]$ where $(\delta_1, \ldots, \delta_n) \leftarrow_R \mathsf{Share}(\delta)$ for $\delta \leftarrow_R \mathbb{F}_{p^r}$.
4. *Commit to Shares:* In the interactive phase, the worker $\mathcal{W}_i$ proceeds as follows: The worker veiws the share $\mathbf{s}_i$ as $\ell \times r$ matrix M_i over $\mathbb{F}$. Then for each $j \in [r]$, it computes $A_{ij} = \mathbf{g}^{M_i[j]}$, where $M_i[j]$ denotes the j^{th} column of the matrix. Similarly it views the input δ_i as vector $(\delta_{i1}, \ldots, \delta_{ir})$ over $\mathbb{F}$. It then computes commitments B_{ij} for $j \in [r]$ as $B_{ij} = h_1^{\delta_{ij}} h_2^{\omega_j}$ for $\omega_j \leftarrow_R \mathbb{F}$. Finally $\mathcal{W}_i$ broadcasts $\mathbf{A}_i = (A_{i1}, \ldots, A_{ir})$ and $\mathbf{B}_i = (B_{i1}, \ldots, B_{ir})$.
5. *Reveal Linear Form over Shares:* The verifier sends a challenge vector $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}^\ell$, and the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + \delta_i$. In the preceding inner-product, we consider $\mathbf{s}_i$ as a vector over $\mathbb{F}_{p^r}$ and v_i, δ_i are considered as elements in the field $\mathbb{F}_{p^r}$. To ensure that corrupt workers use $\mathbf{s}_i, \delta_i$ consistent with earlier commitments $\mathbf{A}_i, \mathbf{B}_i$ we additionally require them to provide proofs by running the proof of knowledge CSP for the following relations (viewing $\mathbf{s}_i$ as $\ell \times r$ matrix M_i over $\mathbb{F}$):

$$\begin{aligned}
\pi_{i1} &= \mathsf{CSP}(M_i) : \mathbf{g}^{M_i[j]} = A_{ij} \, \forall \, j \in [r], \\
\pi_{i2} &= \mathsf{CSP}(\delta_i, \omega_1, \ldots, \omega_r) : h_1^{\delta_{ij}} h_2^{\omega_j} = B_{ij} \, \forall \, j \in [r], \\
\pi_{i3} &= \mathsf{CSP}\big\{(M_i, \delta_i, \omega_1, \ldots, \omega_r) : \\
&\quad \mathbf{g}^{M_i[j]} h_1^{\delta_{ij}} h_2^{\omega_j} = A_{ij} B_{ij} \wedge \langle \boldsymbol{\gamma}, M_i[j] \rangle + \delta_{ij} = v_{ij} \, \forall \, j \in [r]\big\}.
\end{aligned}$$

   The NIPK used above can be instantiated with $O(\log \ell)$ communication complexity using compressed sigma protocols (CSPs) of Attema et al. [3], made non-interactive using Fiat-Shamir transformation. We observe that each proof asserts r constraints, which can be reduced to one constraint each using a random challenge. We skip the details here.
6. *Verifier Determines Honest Commitments:* Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the purported values of $(v_1, \ldots, v_n)$ received in the previous step. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, he verifier sets $v'_i \leftarrow_R \mathbb{F}_{p^r}$ (randomly). Here we use $\mathbf{v} = (v_1, \ldots, v_n)$ defined by $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$ to denote the vector of honestly computed values. We recall that we consider $\mathbf{v}$ to be a vector over $\mathbb{F}_{p^r}^n$. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < \mathsf{dist}/2$, with dist being the minimum distance of the code induced by the

TLSS, $\mathcal{V}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by using error correction. Let C denote indices of corrupt workers (who actually deviate from the protocol). From Lemma 3 we conclude $\mathsf{C} = \{i \in [n] : v_i \neq v'_i\}$ with overwhelming probability. Let $k'_1, \ldots, k'_q$ denote the reconstruction coefficients for the set $[n] \backslash \mathsf{C}$ where each $k'_i = (k'_{i1}, \ldots, k'_{ir}) \in \mathbb{F}^r$ for each i.

7. *Output using honest messages:* $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\prod_{j \in [q], t \in [r]} A_{i_j,t}^{k'_{jt}} = z$, and $(0, \{\mathcal{P}\})$ otherwise.

Theorem 6 (Robust Distributed Proof of Knowledge for Discrete Log for TLSS). *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, the above protocol is a $\mathsf{DPoK}_{\mathsf{TLSS},\mathsf{DlogGen}}$ for relation generator $\mathsf{DlogGen}$ and (t, n, r)-TLSS scheme which satisfies t-privacy and d-robustness, for $d < \mathsf{dist}/3$, where dist is the minimum distance the linear code induced by the TLSS scheme. Moreover the protocol incurs $O(rn)$ communication over point-to-point channels and $O(rn + \log \ell)$ communication over broadcast channels.*

The proof of the above theorem is similar to that for the protocol Π_{dlog}, except that we use Lemma 3 instead of Lemma 2 to identify corrupt messages, and appropriately omit them from the verification check. We now discuss implications of the above theorem for specific threshold secret sharing schemes.

A.2 (Corollary) Distributed Proof of Knowledge using Replicated Secret Sharing

Our earlier results obtained for Shamir Secret Sharing [46] in Theorem 1 can be seen as special case of Theorem 6 for $r = 1$ and $\mathsf{dist} = (n - t)$. Here we additionally specialise Theorem 6 to the case of *replicated secret sharing*. We recall the definition of Replicated Secret Sharing (RSS) Scheme provided in [27].

Definition 13 (Replicated Secret Sharing Scheme). *A $(t, n, \binom{n-1}{t})$ replicated linear secret-sharing (RSS) scheme over a finite field $\mathbb{F}$ consists of algorithms ($\mathsf{Share}, \mathsf{Reconstruct}$) as described below:*

- Share *is a randomized algorithm that on input $s \in \mathbb{F}$, samples $s_A \in \mathbb{F}$ for all $A \in [n], |A| = t$, such that $\sum_A s_A = s$, and sets $s_i = \{s_A : i \notin A\}$. We denote the output as $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(s)$, where $\mathbf{s}_j \in \mathbb{F}^{\binom{n-1}{t}}$ is the share sent to party P_j.*
- $\mathsf{Reconstruct}$ *is a deterministic algorithm that takes a set $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| \geq t$, a vector $(s_1, \ldots, s_{|\mathcal{I}|})$ and outputs $s = \mathsf{Reconstruct}((s_1, \ldots, s_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$.*

A RSS scheme satisfies the following properties:

- **Correctness**: *For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q \geq t$, we have $\mathsf{Reconstruct}((s_{i_1}, \ldots, s_{i_q}), \mathcal{I}) = s$.*
- **Privacy**: *For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q < t$, the tuple $(s_{i_1}, \ldots, s_{i_q})$ is information-theoretically independent of s.*

Remark 6. We note that RSS scheme is a specific instance of TLSS scheme discussed in the prior section.

Let $\mathsf{DlogGen}$ be a relation generator that on input $(1^\lambda, m)$ outputs $(\mathbb{G}, \mathbf{g}, p)$ where p is a λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\mathbf{g} = (g_1, \ldots, g_m) \leftarrow_R \mathbb{G}^m$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\mathbf{g}^{\mathbf{s}} = z$. Let $\mathsf{RSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote $(t, n, \binom{n-1}{t})$ replicated secret sharing over $\mathbb{F}_p$. In this section, we state the theorems and the threshold bounds for RSS as a specific case of TLSS (Theorem 6).

Theorem 7 (Robust Distributed Proof of Knowledge for Discrete Log for Replicated Secret Sharing). *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, protocol $\Pi_{\mathsf{rob\text{-}rss}}$ is a $\mathsf{DPoK}_{\mathsf{RSS},\mathsf{DlogGen}}$ for relation generator $\mathsf{DlogGen}$ and $(t, n, \binom{n-1}{t})$-RSS scheme which satisfies t-privacy and d-robustness, for $d = t < \mathsf{dist}/3$, where $\mathsf{dist} = (n - t)$ is the minimum distance of two valid codewords of the linear code induced by the TLSS scheme.*

Remark 7. We note that the corruption threshold of $t < n/3$ attainable for Shamir Secret Sharing (SSS) Scheme and Replicated Secret Sharing (RSS) Scheme follows from the fact that the underlying linear code defined by both sharing schemes attain a minimum distance of $\mathsf{dist} = n - t$ between any two valid codewords. We note that the linear codes considered for SSS scheme lies in $\mathbb{F}_p$ (Reed-Solomon Codes), whereas the linear codes considered for RSS lies in $\mathbb{F}_{p^k}$.

B PS Signatures and PoK for PS

In this section we show the generality of techniques shown above by providing distributed protocols for another pairing-based signature scheme, whose proof of knowledge of signature also reduces to discrete logarithm relation.

We begin by recalling the Pointcheval Sanders (PS) signature scheme from [43], along with the associated proof of knowledge.

Definition 14 (PS Signature Scheme [43]). *The PS Signature Scheme to sign a message* $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *described as follows :*

- $\mathsf{Setup}(1^\lambda)$ *: For security parameter* λ*, this algorithm outputs groups* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *of prime order* p*, with an efficient bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$*, as part of the public parameters* pp*. Note that the bilinear groups are of type 3, which ensures that there are no homomorphisms between* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *that are efficiently computable.*
- $\mathsf{KeyGen}(\mathsf{pp})$ *: This algorithm samples* $\tilde{g} \leftarrow_R \mathbb{G}_2$ *and* $(x, y_1, \ldots, y_\ell) \leftarrow_R \mathbb{F}_p^{n+1}$*, computes* $(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell) = (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_\ell})$*, and outputs* $(\mathsf{sk}, \mathsf{pk})$*, where* $\mathsf{sk} = (x, y_1, \ldots, y_\ell)$ *and* $D\ \mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: This algorithm samples* $h \leftarrow_R \mathbb{G}_1 \setminus \{0\}$*, and outputs* $\sigma = (h, h^{x + \sum_j y_j m_j})$*.*
- $\mathsf{Verify}(\mathsf{pk}, (m_1, \ldots, m_\ell), \sigma)$ *: This algorithm parses* σ *as* (σ_1, σ_2)*, and first checks if* $\sigma_1 \neq \mathsf{e}_1$*. It then proceeds to check if*

$$e\left(\sigma_1, \tilde{X} \cdot \prod_j \tilde{Y}_j^{m_j}\right) = e(\sigma_2, \tilde{g}).$$

  If yes, it outputs 1, and outputs 0 otherwise.

Note that given $\sigma = (\sigma_1, \sigma_2)$, $\sigma' = (\sigma_1^r, \sigma_2^r)$ is also a valid signature if σ is a valid signature. However, it can be seen that the distribution of σ is not independent of the message $\mathbf{m}$ in the above scheme.

B.1 Proof of Knowledge

PS signatures support an efficient zero-knowledge proof of knowledge (ZKPoK) wherein a prover holding a valid PS signature σ on a message vector $\mathbf{m}$ can efficiently prove knowledge of the signature. A prover $\mathcal{P}$ who owns a PS signature $\sigma = (\sigma_1, \sigma_2)$ on a message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ can prove knowledge of such a signature using a slight modification of the signature scheme as described above. At a high level, $\mathcal{P}$ generates a signature on a a pair $(\mathbf{m}, t)$ for uniformly sampled $t \leftarrow_R \mathbb{F}_p$ based on the original signature σ; the usage of a random t makes the resulting signature independent of $\mathbf{m}$. The complete protocol is as below:

- **Public Key** $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\mathbf{m}$
  1. $\mathcal{P}$ samples $r, t \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r)$.
  2. $\mathcal{P}$ sends the computed value $\sigma' = (\sigma_1', \sigma_2')$ to $\mathcal{V}$.
  3. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK of $(\mathbf{m}, t)$ for the relation:

$$e(\sigma_1', \tilde{X}) \cdot \prod_j e(\sigma_1', \tilde{Y}_j)^{m_j} \cdot e(\sigma_1', \tilde{g})^t = e(\sigma_2', \tilde{g}).$$

  4. $\mathcal{V}$ accepts if the ZKPoK is valid.

The proof of knowledge protocol used in Step (3) is a special case of "proof of opening", wherein we can use a protocol for proving the knowledge of $\mathbf{s} \in \mathbb{F}_p^\ell$ which opens the commitment $z = \mathbf{g}^{\mathbf{s}}$ where $\mathbf{g} = (g_1, \ldots, g_\ell)$ and $g_1, \ldots, g_\ell$ are public generators of a group $\mathbb{G}$ (of order p), where the discrete log problem is hard. We describe the protocol concretely below.

- $\mathcal{P}$ **and** $\mathcal{V}$**'s common inputs**: $z \in \mathbb{G}$.
- $\mathcal{P}$**'s private inputs**: $\mathbf{s} \in \mathbb{F}_p^\ell$.

1. $\mathcal{P}$ samples $\mathbf{r} \leftarrow_R \mathbb{F}_p^\ell$ and computes $\alpha = g^{\mathbf{r}}$.
2. $\mathcal{P} \rightarrow \mathcal{V}$: α.
3. $\mathcal{V} \rightarrow \mathcal{P}$: $c \leftarrow_R \mathbb{F}_p$.
4. $\mathcal{P} \rightarrow \mathcal{V}$: $\mathbf{s}' = c\mathbf{s} + \mathbf{r}$.
5. $\mathcal{V}$ checks: $g^{\mathbf{s}'} = \alpha z^c$.

We also describe another variant of PS Signature Scheme, based on a stronger assumption (Assumption 1 in [43]), that leads to much more efficient distributed prover protocols. This variant is same as the one described in Definition 14, with the exception of KeyGen algorithm which includes additional elements in the public key (hence stronger assumption). The modified KeyGen algorithm is described below:

Definition 15 (PS Signature: B [43]). *The PS Signature Scheme to sign a message* $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *consists of a tuple of PPT algorithms* (Setup, KeyGen, Sign, Verify) *as described in Definition 14, except* KeyGen *which is described below:*

- KeyGen(pp)*: The algorithm samples* $g \leftarrow_R \mathbb{G}_1$, $\tilde{g} \leftarrow_R \mathbb{G}_2$, $(x, y_1, \ldots, y_{\ell+1}) \leftarrow_R \mathbb{F}_p^{\ell+1}$ *and computes* $(X, Y_1, \ldots, Y_{\ell+1}) = (g^x, g^{y_1}, \ldots, g^{y_{\ell+1}})$, $(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1}) = (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_{\ell+1}})$. *It then outputs* $(\mathsf{sk}, \mathsf{pk})$ *where* $\mathsf{sk} = (x, y_1, \ldots, y_{\ell+1})$ *and* $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$.
- Sign$(\mathsf{sk}, (m_1, \ldots, m_\ell))$*: Choose* $h \leftarrow_R \mathbb{G}_1 \backslash \{0\}$ *and output* $(h, h^{x + \sum_{i=1}^{\ell} y_i \cdot m_i})$. *Note that* Sign *still works on the* ℓ*-length message.*

B.2 Alternate Proof of Knowledge

We describe a protocol for showing knowledge of a PS signature (σ_1, σ_2) on a message $\mathbf{m} \in \mathbb{F}_p^\ell$ while simultaneously revealing a dynamically sampled commitment C of $\mathbf{m}$. The proof of knowledge reduces to the knowledge of opening of C and a short pairing check as described below:

- **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\mathbf{m}$
  1. $\mathcal{P}$ samples $r, t, s \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r \cdot Y_{\ell+1}^s)$, $C = \tilde{g}^t \prod_{i=1}^{l} \tilde{Y}_i^{m_i} \in \mathbb{G}_2$.
  2. $\mathcal{P}$ sends the computed value $\sigma' = (\sigma_1', \sigma_2')$ and C to $\mathcal{V}$.
  3. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK showing knowledge of $(m_1, \ldots, m_\ell, t)$ such that $C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}$ and a ZKPoK showing knowledge of s such that $e(Y_{\ell+1}, \tilde{g})^s = e(\sigma_2', \tilde{g}) e(\sigma_1', \tilde{X})^{-1} e(\sigma_1', C)^{-1}$.
  4. $\mathcal{V}$ accepts if the ZKPoKs are valid.

Proof. For completeness, notice that $\sigma_2 = \sigma_1^{x + \sum_{i=1}^{\ell} y_i m_i}$ and thus we have $\sigma_1' = \sigma_1^r$, $\sigma_2' = Y_{\ell+1}^s \cdot \sigma_1^{r(x + \sum_{i=1}^{\ell} y_i m_i + t)}$ and $C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}$. Thus we have:

$$\begin{aligned} e(\sigma_2', \tilde{g}) &= e(\sigma_1^r, \tilde{g}^{x + \sum_{i=1}^{\ell} y_i m_i + t}) \cdot e(Y_{\ell+1}, \tilde{g})^s \\ &= e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', C) \cdot e(Y_{\ell+1}, \tilde{g})^s \end{aligned}$$

The above is equivalent to the verification relation. Zero knowledge follows from the fact that σ_1', σ_2' and C are distributed uniformly in their respective domains, and from the zero knowledge property of the ZKPoKs. To show knowledge soundness, we show an extractor $\mathcal{E}$ which extracts a valid signature on a message in $\mathbb{F}_p^\ell$. Using the extractors for the ZKPoKs, $\mathcal{E}$ obtains $(m_1, \ldots, m_\ell, t, s)$ such that

$$C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}, \quad e(\sigma_2', \tilde{g}) = e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', C) \cdot e(Y_{\ell+1}, \tilde{g})^s$$

The extractor $\mathcal{E}$ computes $\left(\sigma_1 = \sigma_1', \sigma_2 = \sigma_2' (\sigma_1')^{-t} (Y_{\ell+1})^{-s}\right)$. To see that (σ_1, σ_2) is a valid signature we verify:

$$\begin{aligned} e(\sigma_2, \tilde{g}) &= e(\sigma_2', \tilde{g}) \cdot e(\sigma_1', \tilde{g})^{-t} \cdot e(Y_{\ell+1}, \tilde{g})^{-s} \\ &= e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', C) \cdot e(\sigma_1', \tilde{g})^{-t} \\ &= e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \\ &= e(\sigma_1, \tilde{X} \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \end{aligned}$$

The above shows (σ_1, σ_2) is a valid signature for the block $(m_1, \ldots, m_\ell)$ for the public key $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.

C DPoK for PS Signatures over Secret-Shared Inputs

We now present a DPoK for PS signatures for secret-shared inputs. We refer the reader to Section B for the description of the PS signature scheme and its proof of knowledge (in the non-distributed setting) from [43]. We start by defining a relation relevant to PS signature verification.

Definition 16 (PS Relation). *Let* PSGen *denote the relation generator, such that* $\mathsf{PSGen}(1^\lambda, \ell)$ *outputs a bilinear group* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R PS.\mathsf{Setup}(1^\lambda)$. *The corresponding relation* $\mathcal{R}^{\text{ps}}$ *is defined by* $(\mathbf{x}, (\mathbf{m}, \mathbf{u})) \in \mathcal{R}^{\text{ps}}$ *for* $\mathbf{x} = \mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1}) \in \mathbb{G}_1^{\ell+2} \times \mathbb{G}_2^{\ell+3}$, $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *and* $\mathbf{u} = (\sigma, t) = ((\sigma_1, \sigma_2), t) \in \mathbb{G}_1^2 \times \mathbb{F}_p$ *if*

$$e(\sigma_1', \tilde{X}) \cdot \prod_j e(\sigma_1', \tilde{Y}_j)^{m_j} \cdot e(\sigma_1', \tilde{g})^t = e(\sigma_2', \tilde{g}).$$

Our Protocol Π_{ps}. Our DPoK protocol Π_{ps} for relation PSGen is described below, which can be invoked from our compiler with input authentication based on PS signatures (instead of BBS+). It builds upon the known PS PoK [43] in the non-distributed setting. The PoK involved the following steps: (i) the prover randomizes the signature using some auxiliary inputs and broadcasts the randomized signature to all other parties (this randomization ensures unlinkability), and then (ii) the prover shows knowledge of these auxiliary inputs and secret-shares of the message satisfying discrete-log relations determined by the first message.

Our PS PoK over secret-shared inputs follows the same blueprint, where the prover similarly randomizes the first message using certain auxiliary inputs. In our case, the problem reduces to a DPoK for the discrete log relation, with the workers holding the shares of the witness (message) and the verifier holding the public statement (public key pk + the randomized signature). We handle this using our robust complete DPoK Π_{dlog} for discrete log.

Protocol Π_{ps}

- **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\mathbf{m}$
- $\mathcal{W}_i$**'s inputs :** $\mathcal{W}_i$ possesses the i^{th} share $\mathbf{m}_i$ of the message vector $\mathbf{m}$, such that $\mathsf{Reconstruct}(\mathbf{m}_1, \ldots, \mathbf{m}_n) = \mathbf{m}$
- **Pre-processing** : $\mathcal{P}$ samples $t \leftarrow_R \mathbb{F}_p$, computes $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$. $\mathcal{P}$ sends the shares t_i to $\mathcal{W}_i$, for all $i \in [n]$.
- **Interactive Protocol**
  1. $\mathcal{P}$ samples $r, v \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r \cdot Y_{\ell+1}^v)$, $C = \tilde{g}^t \prod_{i=1}^\ell \tilde{Y}_i^{m_i}$. P also generates a NIPK π showing knowledge of v such that $e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', C) \cdot e(Y_{\ell+1}, \tilde{g})^v = e(\sigma_2', \tilde{g})$.
  2. $\mathcal{P}$ broadcasts the computed value $\sigma' = (\sigma_1', \sigma_2')$, C and π to $\mathcal{V}$.
  3. Each $\mathcal{W}_i$ and $\mathcal{V}$ locally set $\mathbf{g} = (\tilde{g}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.
  4. Each $\mathcal{W}_i$ locally holds the i-th share $\mathbf{s}_i = (\mathbf{m}_i, t_i)$ such that $\mathbf{s} = (\mathbf{m}, t) = \mathsf{Reconstruct}\left(\{\mathbf{s}_i\}_{i \in [n]}\right)$.
  5. All $\mathcal{W}_i$ for $i \in [n]$ and $\mathcal{V}$ run DPoK protocol Π_{dlog} for the relation $\mathbf{g}^{\mathbf{s}} = C$
  6. $\mathcal{V}$ accepts if π is valid and Π_{dlog} accepts.

We note that DPoK protocol Π_{ps} achieves robust completeness, knowledge-soundness and zero-knowledge. The proof is straightforward from the existing proof of knowledge of PS signatures and robust completeness, knowledge-soundness and zero-knowledge properties of our DPoK protocol Π_{dlog} for discrete log.

Theorem 8. *Assuming that* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* DlogGen *and* (t, n)-SSS, Π_{ps} *is a* DPoK *for the relation generator* PSGen *and* (t, n)-SSS *with the following properties:*

- ***Security****: t-private and d-robust, for $d <$ dist$/2$, where* $\mathsf{dist} = (n - t)$ *is the minimum distance of the Reed-Solomon code induced by* (t, n)-SSS.

– ***Efficiency***: *$O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

Remark 8 (Public Verifiability). The protocol Π_{ps} was presented and analyzed assuming an honest designated verifier for simplicity. By replacing Π_{dlog} with its publicly verifiable version $\Pi^{\mathsf{pv}}_{\mathsf{dlog}}$ in steps (5) of the Interactive Phase, we obtain a publicly verifiable version of the protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps}}$. Observe that $\Pi^{\mathsf{pv}}_{\mathsf{ps}}$ requires one less round of interaction, as compared to Π_{ps}, while it retains the properties of robust completeness, knowledge soundness and honest verifier zero-knowledge holds identically for the Π_{ps}.