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 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 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 [BB16]. 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 certification authority.

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

Input Authenticity. We first turn to the question of deciding authenticity of inputs. 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*. Moreover, 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 [BBS04,PS16,CV02] 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. [**Moumita:** change "credential"?] Unfortunately, all of the above benefits, which allow authentic data to be used securely in individual context are negated when computing securely over data from *multiple* data owners, if one adheres to the vanilla security guarantees of the multiparty protocols.

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+ [BBS04,ASM06] and PS [PS16] signature schemes which efficiently support minimal disclosure features as mentioned before.

Why naïve solutions are not satisfactory. One straightforward way to achieve this 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 they are 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.

Another approach is to have the certifying authority sign a commitment to the inputs, and then have the parties prove that their inputs are those contained inside the public commitment. Using Pedersen commitments, and customized zero-knowledge protocols, this approach can be more efficient that authenticating inside the MPC. However, this approach does not satisfy the property of *unlinkability*. Unlinkability – ensuring that (same) inputs used by a party across different protocols cannot be linked – is an essential privacy requirement. Our approach works over the shares of the input as opposed to identifying the input via commitments guranteeing unlinkability. Moreover, since our solution is essentially a distributed proof of knowledge of Pedersen commitment openings, this is as efficient (or more, with optimizations) as verifying signature on commitments.

Our goal is to lift existing MPC protocols into ones that also ensure 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 and (iii) incurring overhead in communication that is succinct in size of the inputs (which are large for our applications). This precludes approaches of prior works which require the authentication relation to be expressed as a circuit [BBC+19,HVW22].

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, we put forth a notion of distributed zero-knowledge protocols that are of independent interest.

Compressed Distributed Sigma protocols. 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 protocol 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 prover, and all other parties are verifiers. 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 from a random oracle (RO).

Our definition of distributed proof of knowledge 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. [AC20] to improve the communication complexity of our protocol from being linear in the size of the witness to logarithmic. Our techniques to construct compressed distributed zero-knowledge protocols are general and modular. We believe that sigma protocols for algebraic languages can be distributed using similar techniques, and our building blocks to be of independent interest in other applications.

The ideas outlined above will not prevent malicious provers from disrupting the protocol execution by using bad shares and causing abort. 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 this seeming error-correction over messages "in exponents", we leverage results from low degree testing (Lemma 2) used in constructions of efficient zkSNARKs like [AHIV17,BCR+19]. 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. The main technical challenge in achieving robust completeness for DPoK is to retain succinctness. 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 distributed sigma protocols as discussed above generalize to any *threshold linear secret sharing* (TLSS) scheme. In particular, for the case of robust distributed sigma protocols, we characterize the robustness threshold in terms of the minimum distance of the linear code associated to the TLSS scheme. The generalized protocols appear in Appendix A. Concrete bounds are obtained for Shamir Secret Sharing and Replicated Secret Sharing schemes.

Authenticated MPC. We consider a notion of input authenticity where the inputs possess a valid signature from a certification authority. This is a standard notion where applications know an entity who can certify that inputs satisfy certain properties by providing a signature on inputs[3]. Informally, we give a protocol that realizes the following authenticated MPC functionality.

- The parties send their inputs x_i and signature σ_i on x_i to $\mathcal{F}$ for $i \in [n]$.
- The functionality $\mathcal{F}$ checks that σ_i is a valid signature on x_i for all $i \in [n]$. If any of the signatures is invalid, for all invalid inputs x_j, it sends $(\texttt{abort}, P_j)$ to all the parties. Otherwise it computes $y = f(x_1, \ldots, x_n)$ and sends y to all parties.

In Section 4, we propose a generic compiler that transforms a protocol Π based on TLSS scheme to an *authenticated* protocol Π'. We describe our compiler for malicious protocols based on Shamir secret sharing, though it can be generalized to any TLSS based protocol, using the generalized distributed sigma protocols in Appendix A. For authentication, our techniques employ signature schemes that are algebraically compatible: these include Camenisch-Lysyanskaya (CL) signatures [CL01], Boneh-Boyen-Shaham (BBS) signatures (and variants) [BBS04,ASM06,CDL16], and Pointcheval-Sanders (PS) signatures [PS16]. These are signature schemes that support efficient zero-knowledge proofs of knowledge of a valid message-signature pair. We consider BBS+ signatures[4] to illustrate the building blocks of our compiler and implementation, and show the generality of our techniques by providing protocols for PS signatures as well in Section 4.1. We believe our techniques extend to other such structured algebraic signatures such as CL signatures [CL01]. The compiled protocol Π' inherits the security of

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

[4] There are standardization efforts for a version of BBS called BBS+ that has led to a recent RFC draft [LKWL22].

Π. If Π guarantees security with abort for $t < n/3$, then the same holds for Π'; and if Π achieves guaranteed output delivery, then so does Π', when $t < n/3$, as long as the inputs are authentic (by definition, we abort if this is not the case)[5]. The latter crucially uses a *robustness* property of our distributed zero-knowledge protocol. Our compiler incurs negligible communication overhead over Π.

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 modelled as a proof of knowledge of the opening of a Pedersen commitment. We present specific instances of this general approach for signature schemes that are candidates for standardization (e.g., [BBS04,ASM06] is a candidate standard for verifiable credentials in Web 3.0). 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 $\ell >> n$ [BLZLN21,GP16]. In our application, we indeed expect ℓ to be $\Omega(\lambda n)$ where λ is a security parameter.

1.2 Related Work

Certified Inputs. The works of [KMW16,Bau16,ZBB17] achieve input validation for the special case of *two*-party computation using garbled circuit (GC) based techniques. The work of [BJ18] constructs MPC with certified inputs, albeit using techniques that are specific to certain MPC protocols [DN07,DKL$^+$13]. A recent work [ADEO21] develops techniques for computing bilinear pairings over secret shared data, thus enabling signature verification inside MPC for the Pointcheval-Sanders signature scheme [PS16]. Our proposed compiler uses efficient compressed distributed sigma protocol proofs for signature verification instead of verifying signatures inside the MPC protocol, and differs from both [BJ18] and [ADEO21] 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 [BJ18,ADEO21]. 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 [BJ18] nor [ADEO21] achieves.

Distributed Zero-knowledge. Various notions of distributed zero-knowledge have appeared in literature. The notion of distributed interactive proofs has appeared in [Ped91], in the context of relations describing the verification of signatures, where the signature is public and secret key is shared among the participants. The notion in [WZC$^+$18] considers a distributed prover in order to improve prover efficiency, but the witness is still held by one entity. In Feta [BJO$^+$22], 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 [CB17] 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 [SVdV16,OB21,DPP$^+$22], where the focus is on jointly computing a non-interactive publicly verifiable proof (with specific focus on Groth16 [Gro16], Plonk [GWC19] and Marlin [CHM$^+$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.

Fully Linear PCPs and Distributed Verification. A related notion of zero-knowledge proofs on distributed data is explored in [BBC$^+$19] that proposes the abstraction of a fully linear PCP (FLPCP) where each verifier only has access to a share of the statement. A similar notion based on MPC-in-the-head paradigm is presented in a concurrent work [HVW22]. We provide below a high level comparison of our work with aforementioned works in terms of definition, applications, and efficiency. Later in Section 3.2, we compare more concretely after having introduced our definitions.

Efficiency. While techniques of [BBC$^+$19] can indeed be used to achieve our goals, the focus of our work is on concrete efficiency (prover overhead, communication overhead on top of the underlying

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

unauthenticated MPC). In order to use [BBC+19], one has to express the relation as an arithmetic circuit; for the languages we consider (algebraic relations), expressing them as a circuit is prohibitively expensive. Instead, we take advantage of the algebraic nature of the relation to design concretely efficient distributed sigma protocols. In addition, [BBC+19] 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 [BBC+19] 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 workers running the proof on behalf of the prover (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 [BBC+19] 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 and admit efficient sigma protocols. Subsequent works [BGIN20] have used the FLPCP notion of distributed ZK on secret shared data to construct MPC protocols with full security. The concurrent work of [HVW22] also consider a setting that is subtly different from ours, where verification is distributed and relies on a single designated prover knowing the entire secret, and robustness holds with respect to dishonest verifiers. In contrast, our prover is distributed, the distributed phase of our protocol only requires the provers to possess a valid sharing of the witness (no prover needs to know the entire witness), and verification is public and only needs broadcast messages. Our robustness notion is against dishonest provers.

Distributed witness vs Distributed statement. In general, relations with shared witness and shared statement are equivalent via universal relations. However, our distributed witness model has several advantages over distributed statement model in prior works. 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), incuring overheads of writing the algebraic relation as a circuit. Modular exponentiation, for instance 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 a 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. 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[AC20], Bulletproofs[BBB+18] etc., that avoid circuit representation for several useful relations) to be amenable to be distributed under our definition.

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 $\Sigma = (\mathcal{P}^1, \mathcal{P}^2, \mathcal{V})$ 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. 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 $z_i (= \alpha_i + cw_i)$. The transcript is $\tau = (a_1, \ldots, a_n, c, z_1, \ldots, z_n)$, and the verifier accepts iff $g^{\sum_i z_i} = \prod_i a_i x^c$. This holds since $g^{\sum_i z_i} = g^{\sum_i (\alpha_i + cw_i)} = \prod_i a_i x^c$.

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.

Succinctness. This distributed Sigma protocol has linear communication complexity. To achieve succinctness, one could apply split-and-fold compression techniques to reduce the instance size by half based on a random challenge, and recurse, in order to make our distributed protocol *succinct*. To illustrate the idea, consider the distributed Schnorr described above adapted for vectors, that is for proving knowledge of $\mathbf{w} \in \mathbb{F}_p^m$ such that $x = \mathbf{g}^{\mathbf{w}}$, where $\mathbf{g}^{\mathbf{w}} = \prod_{i=1}^{m} g_i^{w_i}$. In this protocol, each $\mathcal{P}_i$ broadcasts a vector $\mathbf{z}_i$ as its third message, and this is the source of linear communication, since each prover's first message is still one group element, $a_i = \mathbf{g}^{\boldsymbol{\alpha}_i}$. We now outline the ideas to compress this communication. Let us denote component wise product by $\mathbf{g} \circ \mathbf{h} = (g_1 h_1, \ldots g_n h_n)$ for $\mathbf{g}$ and $\mathbf{h} \in \mathbb{G}^n$. Now, after receiving the verifier challenge c, each $\mathcal{P}_i$ uses c to compute a new instance (and corresponding witness), but of half the size, as follows: broadcast shares of the new instance $A_i = \mathbf{g}_R^{\mathbf{w}_{i,L}}, B_i = \mathbf{g}_L^{\mathbf{w}_{i,R}}$ where $\mathbf{g} = \mathbf{g}_L || \mathbf{g}_R$; set new reduced instance to be $\mathbf{g}' = \mathbf{g}_L^c \circ \mathbf{g}_R$, and $x' = x^c \prod_i A_i \prod_i B_i^{c^2}$; set new witness share to be $\mathbf{w}_i' = \mathbf{w}_{i,L} + c\mathbf{w}_{i,R}$. Recursing until the instance size is constant yields a protocol with logarithmic communication. Here again, we take advantage of the linearity of the secret sharing scheme in order to split and fold the shares in the exponent.

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 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 [BBS04], the PS signature scheme [PS16] and the Pedersen commitment protocol [Ped91] are some candidates for which our distributed protocol can be instantiated. Our compiler reuses the sharing that is already done as part of an 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 PS signatures (Section 4.1), 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 [BBC$^+$19] and previous works on authenticated MPC [BJ18,ADEO21,HVW22] 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 $x \leftarrow_R \mathcal{X}$ 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 $\{s_i\}_{i\in Q}$ for $Q \subseteq [n]$ and $|Q| > t$ 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 $\{\mathbf{s}_i\}_{i\in Q}$ for $Q \subseteq [n]$ with $|Q| \geq t+1$ 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 $i \in Q$. 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 $\{\mathbf{s}_i\}_{i\in Q}$.*

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 $Q \subseteq [n]$ with $q = |Q| \geq t+1$, there exists $q \times (n-t)$ matrix $\mathbf{H}_Q$ over $\mathbb{F}$ such that shares $\{\mathbf{s}_i\}_{i\in Q}$ are $\mathcal{L}^m$-consistent and determine the value $\mathbf{s} \in \mathbb{F}^m$ if and only if $\mathbf{X}\mathbf{H}_Q = (\mathbf{s}, \mathbf{0}^{n-t-1})$ where $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_q)$ is some canonical ordering of $\{\mathbf{s}_i\}_{i\in Q}$.*

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

[**Moumita:** Change the notation to concatenation?]

The following coding theoretic result is used to identify malicious behaviour in the distributed proof of knowledge protocol in Section 3.3. It has been previously used in construction of zero knowledge proofs in the interactive oracle setting (e.g [AHIV17,BCR+19]), to check that the oracle represents "low degree polynomials".

Lemma 2 ([BCI+20], 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*[6] 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* [BCC+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 [FS87] heuristic by deriving the verifier's messages as the output of a Random Oracle. In this work, we consider public-coin protocols.

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

2.3 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 [AC20], 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 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 [FS87], which we denote by NIPK.

2.4 BBS+ Signatures and PoK for BBS

In this section, we recall the BBS+ signature scheme from [BBS04,LKWL22], along with the associated proof of knowledge [CDL16].

The BBS+ Signature Scheme. We first recall the the BBS+ signature scheme from [BBS04,LKWL22].

Definition 5 (BBS+ Signature Scheme [BBS04,LKWL22]). *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 :*

- $\mathsf{Setup}(1^\lambda)$ *: 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. We now recall the proof of knowledge for BBS+ signatures, which was originally proposed in [CDL16].

- **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 $\bar{A} = (A')^{-\beta} \cdot b^{r_1} (= (A')^x)$, where $b = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)$.
  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 discrete-logarithm relation $\{(A')^{-\beta} h_0^{r_2} = \frac{\bar{A}}{d} \quad \wedge \quad d^{-r_3} h_0^{s'} \prod_{i=1}^{\ell} h_i^{m_i} = g_1^{-1}\}$, 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.

2.5 PS Signatures and PoK for PS

We recall the Pointcheval Sanders (PS) signature scheme from [PS16], along with the associated proof of knowledge.

Definition 6 (PS Signature Scheme [PS16]). *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* $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: 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.

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 $\mathbf{m}$. [**Sikhar:** It might be good to motivate briefly that this property is useful to achieve unlinkability (or something similar).] 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. $\mathcal{P} \to \mathcal{V}$: α.
  3. $\mathcal{V} \to \mathcal{P}$: $c \leftarrow_R \mathbb{F}_p$.
  4. $\mathcal{P} \to \mathcal{V}$: $\mathbf{s}' = c\mathbf{s} + \mathbf{r}$.
  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 [PS16]), that leads to much more efficient distributed prover protocols. This variant is same as the one described in Definition 6, 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 7 (PS Signature: B [PS16]). *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* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *as described in Definition 6, except* KeyGen *which is described below:*

- $\mathsf{KeyGen}(\mathsf{pp})$*: 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* $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$*.*
- $\mathsf{Sign}(\mathsf{sk}, (m_1, \ldots, m_\ell))$*: 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.*

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 $\big(\sigma_1 = \sigma_1', \sigma_2 = \sigma_2'(\sigma_1')^{-t}(Y_{\ell+1})^{-s}\big)$. 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 $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.

3 Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge (DPoK in short) 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 8 (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 $(\mathbf{x}, \mathbf{s})$ with $\mathbf{s} \in \mathbb{F}^m$.
- **Input Phase**: The prover $\mathcal{P}$ receives $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}$ 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 the parties receive $\mathbf{x}$ as input.
- **Preprocessing Phase**: This is an optional phase where prover $\mathcal{P}$ communicates with workers and verifier using secure private channels.
- **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 $(\mathsf{pp}, \mathbf{x}, \mathbf{s})$ as $\mathcal{P}$'s input, $(\mathsf{pp}, \mathbf{x}, \mathbf{s}_i, \mathsf{aux}_i)$ as the input of $\mathcal{W}_i$ and $(\mathsf{pp}, \mathbf{x}, \mathsf{aux}_V)$ as the input of $\mathcal{V}$. Here aux_* denotes the messages received by the parties over private communication. 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.

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

In the above, $\mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp}, \mathbf{x})$ denotes verifier's output in the protocol Π with its input as $(\mathsf{pp}, \mathbf{x})$ and $\mathcal{A}$ being the adversary, while $\mathcal{L}$ denotes the linear code associated with the secret sharing scheme. We remark that 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 the shares of corrupt parties which reconstruct a valid witness.

- **Zero-Knowledge**: We say that a DPoK is 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.
- **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. We introduce the notion of *robust completeness* – a stronger notion of completeness for DPoKs that is *robust* to the presence of some corrupt parties. Note that the standard notion of completeness only holds if all the workers follow the protocol. This is sometimes undesirable; given that the shares of the honest parties are sufficient to determine the secret, an honest prover should expect to be able to "ride over" a few deviating workers – a property that is guaranteed by robust completeness. Additionally, using a robust complete DPoK to design MPC over authenticated inputs ensures that input authentication does not abort in the presence of malicious behavior, i.e., if the remainder of the protocol has resilience against malicious behavior, input authentication preserves it.

Remark 2. We assume an honest verifier $\mathcal{V}$ in the above definition for ease of exposition. However, our eventual goal is to have a publicly verifiable transcript.

Remark 3. Looking ahead, we define $\mathsf{PV\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ as the publicly verifiable version of $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ in the Random Oracle Model, where the verifier's challenge is computed using the Fiat-Shamir heuristic [FS87]. We note that although the public-coin nature of our DPoK allows us to achieve public verifiability, the distributed nature of our protocol may not allow us to achieve complete NIZK proofs, as we may need all the workers messages in the prior round to use the Fiat-Shamir heuristic, however it allows us to compress rounds and provide a publicly verifiable version of our DPoKs. [**Moumita:** I hope this remark is enough for us to use the $\mathsf{PV\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ notation for our theorem statements.]

3.2 Comparison with Related Work on Distributed Proofs

We compare our notion of n-party distributed proofs of knowledge with recently formulated notions of distributed zero knowledge proofs, particularly in [BBC+19] and in a concurrent work [HVW22]. Similar to our case, both of these works also present interactive protocols for verifying an NP relation in a distributed manner using both private channels and broadcast, however there are substantial differences in the notion of distributed relation, approach and the guarantees of the respective protocols.

Distributed Relation: The formulation of distributed relation in above works is subtly different from ours: In the aforementioned works, the relation $R(x, w)$ is described by an arithmetic circuit $C(x, w)$ and the statement x is shared as vector $(x_1, \ldots, x_n)$ among the parties with i^{th} party knowing the piece x_i. In contrast, we consider statement to be public, and witness to be distributed among the provers. For example, 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 s and w denotes the witness required to check the discrete-log relation in an arithmetic circuit. This introduces obvious inefficiencies. Since, our focus is on relations described algebraically, we formulate our notion as distributed witness, with multiple provers sharing the witness.

Approach. Our approach used to verify the distribued relation also differs significantly. In [BBC+19], the designated prover computes a fully linear PCP proof for $C(x, w) = 1$ which it shares with other (verifying) parties. The verifiers then verify the proof generating a random query, querying the local shares of the proof and then combining them (using linearity) to learn the output bit. In [HVW22], the designated prover with entire statement and witness runs a threshold n-party MPC protocol in its head, and shares the respective views with other parties (acting as verifiers). The verifiers then communicate over broadcast channel to determine if the output bit follows from correct execution

of the MPC protocol with sufficient honest parties. In contrast, in our approach requires distributed provers to execute a sigma protocol over their shares, similar to folklore sigma protocols, with novel error-correction involving messages in the exponents to ensure only honest messages are used by the verifier. We detail our approach in the next section.

Guarantees and Efficiency. The distributed proofs in [BBC+19,HVW22] are constant round protocols secure against unbounded adversaries. The security is achieved in the standard model assuming helper functionalities like broadcast and ideal coin tossing. Both the related works achieve soundness assuming honest majority. While the FLPCP based approach in [BBC+19] achieves completeness assuming fully honest setting, the MPC in the head based approach of [HVW22] achieves *strong completeness*, ensuring completeness holds against a small $(n/6)$ number of corrupt parties. Both the methods incur an initial $O(|C|)$ communication from designated prover to distributed verifiers over private channels. In contrast, we only assume efficient adversaries (indeed, our relations of interest are based on computational hardness assumptions) and achieve security with constant number rounds in the random oracle model. Our soundness holds for arbitrary corruption threshold, while completeness holds for a corruption threshold of $n/3$. Our communication is $O(\log \ell)$ over both the private and the broadcast channels.

3.3 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) [Sha79]. 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.

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 $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$.
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, the workers first commit to their respective shares by boradcasting $A_i = \mathbf{g}^{\mathbf{s}_i}$ and $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$. The workers also broadcast associated proofs of knowledge π_{i1} and π_{i2} as:

$$\pi_{i1} = \mathsf{NIPK}\left\{(\mathbf{s}_i) : \mathbf{g}^{\mathbf{s}_i} = A_i\right\}, \quad \pi_{i2} = \mathsf{NIPK}\left\{(r_i, \omega_i) : h_1^{r_i} h_2^{\omega_i} = B_i\right\}$$

5. **Reveal Linear Form over Shares:** The verifier sends a challenge vector $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$, and 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:

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

   The NIPKs used in above steps can be instantiated with $O(\log \ell)$ communication complexity using compressed sigma protocols (CSPs) of Attema et al. [AC20], made non-interactive using Fiat-Shamir transformation.
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, 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 $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$, $\mathcal{V}$ 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* NIPK *satisfies completeness, knowledge-soundness and zero-knowledge with $O(\log \ell)$-communication overhead, Π_{dlog} is a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ 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.*

Proof. In order to prove security, we prove robust completeness, soundness and zero-knowledge. [**Moumita:** check the definition for consistency.]

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 $i \in Q$. We can assume that the proofs $\pi_{i1}, \ldots, \pi_{i3}$ were valid, for otherwise $b_i = 0$, which would imply $i \notin Q$, 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 $i \notin Q$, which is again a contradiction. This proves that $\mathbf{s}'_i = \mathbf{s}_i$ for $i \in Q$, and thus the vector $(\mathbf{s}'_i)_{i \in Q}$ 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 NIPK proof $\pi_{i_j 1}$ 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 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 NIPK 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 NIPK has $O(\log \ell)$-communication overhead (such an instantiation of NIPK follows from the Fiat-Shamir transformed CSPs of [AC20]), 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 NIPK). This completes the proof of efficiency/succinctness for Π_{dlog}, and hence the proof of Theorem 1. □

The following corollary of Theorem 1 follows immediately and yields the concrete bounds on the corruption threshold tolerated by Π_{dlog}.

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

Publicly Verifiable Version of Protocol Π_{dlog}. Looking ahead, we use a publicly verifiable version of Π_{dlog} to design more advanced protocols. This publicly verifiable version which we call $\Pi^{\mathsf{pv}}_{\mathsf{dlog}}$ uses the well-known Fiat-Shamir heuristic [FS87] and a random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}_p^\ell$. We briefly outline the changes from Π_{dlog} for completeness. In Step 5 of Π_{dlog} (revealing linear form over shares), instead of receiving the challenge $\boldsymbol{\gamma}$ from the verifier, each worker $\mathcal{W}_i$ computes it as

$$\boldsymbol{\gamma} = \mathsf{RO}\left(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n\right) \in \mathbb{F}_p^\ell.$$

The verification is also modified accordingly. Note that the above transformation does not affect robust completeness and succinctness. It also does not affect soundness because the proof of soundness relies purely on invoking the extractor for NIPK, and hence works identically for Π_{dlog} and $\Pi^{\mathsf{pv}}_{\mathsf{dlog}}$. Finally, we argue zero-knowledge for $\Pi^{\mathsf{pv}}_{\mathsf{dlog}}$ by allowing the simulator to program the random oracle to the challenge vector $\boldsymbol{\gamma}$ (the rest of the simulation is as described earlier for Π_{dlog}).

Generalization to Threshold Linear Secret Sharing Scheme. 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 $\mathbb{G}$, 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.*

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 2 (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 the linear code induced by the TLSS scheme.*

4 Compiler for MPC with Input Authentication

In this section, we present our compiler for MPC with input authentication based on algebraic signatures. We start with providing our building blocks, and then we proceed with the compiler description and its proof of security and efficiency.

4.1 PoK for Algebraic 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).

Looking ahead, we use these protocol as a building block 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 use the variant of BBS+ signature scheme from [CDL16], which was adapted from earlier schemes in [BBS04,ASM06], followed by a construction using PS signatures [PS16].

PoK for BBS+ Signatures over Secret-Shared Inputs. We now present a PoK for BBS+ signatures for secret-shared inputs. We refer the reader to Section 2.4 for the description of the BBS+ signature scheme and its proof of knowledge (in the non-distributed setting) from [BBS04,ASM06].

Our Protocol $\Pi_{\mathsf{bbs+}}$. Our protocol $\Pi_{\mathsf{bbs+}}$ is described below, to be invoked from our compiler with input authentication based on BBS+ signatures for modularity. It builds upon the known BBS+ PoK [BBS04,ASM06] in the non-distributed setting. 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+ PoK 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, with the workers holding the shares of the witness (message + auxiliary inputs) and the verifier holding the public statement (public key pk + first round message). We handle this using our robust complete DPoK Π_{dlog} for discrete log.

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 $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
- $\mathcal{W}_i$**'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 $C = d^{-v} h_0^{t-\eta} \wedge (A')^{-\beta} h_0^r = \frac{\bar{A}}{d}$, 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* $e(A', w) = e(\bar{A}, g_2)$, *and both* instances of Π_{dlog} accept.

PoK for PS Signatures over Secret-Shared Inputs. We now present a PoK for PS signatures for secret-shared inputs. We refer the reader to Section 2.5 for the description of the PS signature scheme and its proof of knowledge (in the non-distributed setting) from [PS16].

Our Protocol Π_{ps}. Our protocol Π_{ps} 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 [PS16] 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 any instantiation of Π_{ps} ensures 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.

Remark 4 (Unlinkability). Note that, in both the protocols $\Pi_{\mathsf{bbs+}}$ and Π_{ps}, the signatures are part of the secret witness. In particular, the protocols not reveal any additional information about σ to the verifier $\mathcal{V}$ beyond its validity. This property ensures unlinkability across multiple proofs using the same signature. We note that the original PoK for BBS+ [BBS04,ASM06] and PS [PS16] satisfies unlinkability, albeit in the non-distributed setting. We ensure that our PoK for BBS+ and PS signature over distributed inputs also satisfies this property.

Remark 5 (Public Verifiability). The protocol $\Pi_{\mathsf{bbs+}}$ (resp., Π_{ps}) was presented and analyzed assuming an honest designated verifier for simplicity. By replacing Π_{dlog} with its publicly verifiable version $\Pi_{\mathsf{dlog}}^{\mathsf{pv}}$ in steps (2) and (3) of the Interactive Phase, we obtain a publicly verifiable version of the protocol, which we call $\Pi_{\mathsf{bbs+}}^{\mathsf{pv}}$ (resp., $\Pi_{\mathsf{ps}}^{\mathsf{pv}}$). Observe that $\Pi_{\mathsf{bbs+}}^{\mathsf{pv}}$ (resp., $\Pi_{\mathsf{ps}}^{\mathsf{pv}}$) requires one less round of interaction, as compared to $\Pi_{\mathsf{bbs+}}$ (resp., Π_{ps}).

Remark 6. We also remark that while the protocols $\Pi_{\mathsf{bbs+}}$ and Π_{ps} (and their publicly verifiable versions $\Pi_{\mathsf{bbs+}}^{\mathsf{pv}}$ and $\Pi_{\mathsf{ps}}^{\mathsf{pv}}$ respectively) resemble DPoK, we do not formally model them as DPoKs. Observe that

these protocols require the prover to send to the workers a part of the witness in a non-secret-shared form (concretely, the prover sends randomized versions of the signature, which is part of the witness instead of secret-sharing it to the workers). This departs from our formal definitions of DPoK. Hence, we do not explicitly prove properties, such as robust completeness, knowledge-soundness and zero-knowledge for these protocols. We primarily use them as building blocks in the description of our compiler for modularity.

Remark 7. 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 both $\Pi_{\mathsf{bbs+}}$ and Π_{ps} also inherit the same communication overheads from Π_{dlog}.

4.2 Compiler for MPC with Input Authentication using BBS+ signatures

In this section we present our compiler for MPC with input authentication that builds upon our PoK for BBS+ signatures over secret-shared inputs. In particular, the compiler outputs an MPC protocol where each input is authenticated using a BBS+ signature under a common (public) verification key. Note that similar techniques can be used to provide a compiler for MPC with input authentication based on PS signatures.

As described subsequently, 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 $\Pi_{\mathsf{bbs+}}$). The compiled protocol could either abort after identifying malicious parties with non-authenticated inputs (thus preserving the id-abort security guarantees of the underlying MPC protocol), or substitute some default authenticated inputs for the identified malicious parties (thus preserving the full/GOD security guarantees of the underlying MPC protocol). For simplicity of exposition, we present the id-abort secure version of our compiler.

We further note that our 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 $t < n/3$.

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 $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

We now present a formal description of our compiler.

Notations. 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 $y = f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

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 as below:

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 $(\mathsf{pk}, (\mathbf{x}_i, \sigma_i)) \in \mathcal{R}^{\text{bbs}}$.
- $\overline{\Pi}_{\mathsf{sh}}$: 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 $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \in \mathcal{R}^{\text{bbs}}$ 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}.

Theorem 3 (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 Overview. We first provide an informal overview of our proof. Our proof uses certain properties of $\Pi_{\mathsf{bbs+}}$ informally outlined below. These properties hold equivalently for its publicly verifiable version $\Pi_{\mathsf{bbs+}}^{\mathsf{pv}}$.

- 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 signature as:

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

  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 $\mathbf{m}$. Looking ahead, in the formal proof of security for our compiled MPC protocol, we use this extraction algorithm Ext' to extract the signatures on the inputs of the corrupt parties.

– 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 $\mathsf{Sim}_1^{\mathsf{zk}}$ for Π_{dlog} and a ZK-simulator $\mathsf{Sim}_2^{\mathsf{zk}}$ for the single-prover proof of knowledge for BBS+ signatures from [CDL16] (recalled in Section 2.4), 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 $\mathsf{Sim}_2^{\mathsf{zk}}$, 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 $\mathsf{Sim}_1^{\mathsf{zk}}$ 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.

We now use the above ideas to formally prove Theorem 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 $\Pi_{\mathsf{ampc}} = \left(\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}}\right)$. 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 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) The first step in the simulation of the online phase is to simulate the proofs of knowledge of BBS+ signatures on the inputs of the honest parties. The simulator proceeds as the algorithm Sim' that was outlined informally in the proof overview. Using the simulator for the single-prover proof of knowledge for BBS+ signatures from [CDL16], the simulator 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 simulators for the respective DPoKs on the statements derived from the simulated first message.
   (b) For each instance $\Pi_{\mathsf{bbs+}}$, where a corrupt party P_i is acting as the prover, invoke the extraction algorithm Ext' described in the proof overview on $(\mathbf{s}_j^i)_{j \in \mathcal{H}}$ to extract the witness $(\mathbf{x}_i, \sigma_i)$ from P_i.
   (c) Invoke $\mathsf{Sim}_{\mathsf{on}}$ to simulate the online phase of the underlying MPC Π_{mpc}.
3. Send $\{(\mathbf{x}_i, \sigma_i)\}_{i \in \mathcal{C}}$ to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$. If $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$ aborts, abort; otherwise output whatever $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$ 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 $\mathsf{Sim}_{\mathsf{sh}}$.
– $\{\mathsf{Hyb}_{2,j}\}_{j \in [0, n-|\mathcal{C}|]}$: 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.

- $\{\mathsf{Hyb}_{3,j}\}_{j\in[0,|\mathcal{C}|]}$: Hybrid $\mathsf{Hyb}_{3,0}$ is identical to hybrid $\mathsf{Hyb}_{2,n-|\mathcal{C}|}$, while 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 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 $(\mathbf{s}_i^j)_{i\in\mathcal{H}}$ to extract the witness $(\mathbf{x}_j, \sigma_j)$. If $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \notin \mathcal{R}^{\text{bbs}}$, 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 $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$ with the extracted inputs $\{(\mathbf{x}_i, \sigma_i)\}_{i\in\mathcal{C}}$.

$\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$. 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 $(\mathbf{m}, \sigma)$. 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 is given by the output of $\mathsf{Sim}_{\mathsf{on}}$, which by the security of Π_{mpc} is $f(\mathbf{x}'_1, \ldots, \mathbf{x}'_n)$ where $\mathbf{x}'_i$ is the input determined at the end of $\mathsf{Sim}_{\mathsf{sh}}$. In Hyb_5, the output is given by $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$ which is $f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{x}_i$, given by the knowledge extractor is the unique $\mathbf{x}'_i$ determined at the end of Π_{sh}. We also note that Hyb_5 is identical to $\overline{\mathsf{Sim}}$. This completes the proof of Theorem 3. □

References

AC20. 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.

ADEO21. 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.

AHIV17. 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.

ASM06. 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.

Bau16. 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.

BB16. 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.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBC+19. 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.

BBS04. 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.

BCC+16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

BCI+20. 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.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

BGIN20. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 244–276. Springer, Heidelberg, December 2020.

BJ18. 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.

BJO+22. 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`.

BLZLN21. 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`.

CB17. Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI 2017*, pages 259–282. USENIX Association, 2017.

CDL16. 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.

CDN15. Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

CHM+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.

CL01. 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.

CV02. 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.

DKL+13. 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.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO*, pages 572–590, 2007.

DPP+22. Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any NP statement jointly? efficient distributed-prover zero-knowledge protocols. *Proc. Priv. Enhancing Technol.*, 2022(2):517–556, 2022.

Esc22. Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Report 2022/062, 2022. `https://eprint.iacr.org/2022/062`.

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

GP16. 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.

Gro16. 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.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

HVW22. 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`.

KMW16. 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`.

LKWL22. 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`.

OB21. 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`.

Ped91. 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.

PS16. 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.

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

SVdV16. Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 346–366. Springer, Heidelberg, June 2016.

WZC+18. 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.

ZBB17. 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 [Sha79] 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 [CDN15, Chapter 6] to the case when each party receives same number of shares.

Definition 9 (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 $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R$ 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 $s =$ 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 8. 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 3.3 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 3.3), 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 10 (TLSS induced code). *For an (n, t, r)-TLSS scheme over $\mathbb{F}$ given by algorithms (Share, Reconstruct), we define linear code $\mathcal{L}_{\mathsf{TLSS}}$ over the field $\mathbb{F}_{p^r}$ as*

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

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 [DPP+22][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 for the following relations (viewing $\mathbf{s}_i$ as $\ell \times r$ matrix M_i over $\mathbb{F}$):

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

   The NIPK used above can be instantiated with $O(\log \ell)$ communication complexity using compressed sigma protocols (CSPs) of Attema et al. [AC20], 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 4 (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 [Sha79] in Theorem 1 can be seen as special case of Theorem 4 for $r = 1$ and $\mathsf{dist} = (n - t)$. Here we additionally specialise Theorem 4 to the case of *replicated secret sharing*. We recall the definition of Replicated Secret Sharing (RSS) Scheme provided in [Esc22].

Definition 11 (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 9. We note that RSS scheme is a specific instance of TLSS scheme discussed in the prior section.

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 $\mathsf{RSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ 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 4).

Theorem 5 (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* 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 10. 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 $\mathbb{F}_{p^k}$.