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 $t < n/4$, 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. We also illustrate the practicality of our approach by extending the well-known MP-SPDZ library with our compiler, thus yielding prototype authenticated MPC protocols.

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 "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. 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 present substantial progress to address the above shortcoming, by efficiently augmenting existing MPC protocols to additionally ensure that inputs have a valid attestation (in form of a digital signature) from the 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.

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.

Robust Distributed Sigma protocols. The ideas outlined above will not prevent malicious provers from disrupting the protocol execution by using bad shares and causing abort. In Section 3.3, we put forth a notion of robustness which additionally provides tolerance against abort in the presence of $n/4$ malicious provers. That is, when the shares indeed reconstruct a valid witness, the protocol will lead the verifier to accept even if up to $n/4$ provers deviate from the protocol. To achieve this seeming error-correction over messages "in exponents", we leverage results from low degree testing (Lemma 1) 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. We leave applicability of these techniques to other relations as an interesting future work.

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 3.4. 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 5, 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 3.4. 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 Appendix B. We believe our techniques extend to other such structured algebraic signatures such as CL signatures [CL01]. The compiled protocol Π' inherits the security of 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 $t < n/4$, 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. We provide the concrete communication overhead incurred by our compiler and compare it with related works in Tables 1 and 2. For further discussion on asymptotic comparison of efficiency, see Section B.3 and Table 3. A more detailed overview of our technical ideas is in Section 1.3.

[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].

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

	Succinct Signature	Succinct Communication	Robustness
BJ18 [BJ18]	No	No	No
ADEO21 [ADEO21]	Yes	Yes	No
$\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ (Sec. 4.1)	Yes	Yes	No
$\Pi_{\mathsf{ps\text{-}auth\text{-}opt}}$ (Appendix B.4)	Yes	Yes	No
$\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}$ (Appendix A.2)	Yes	Yes	Yes

Table 1: Comparison of features embodied by authenticated MPC protocols. Succinctness for signatures and communication refers to signature size and communication being sublinear with respect to message size and unauthenticated communication respectively.

Implementation. We implement our protocol to illustrate the practical viability of our approach. In Section 6, we plug our compiler to the versatile MP-SPDZ [Kel20] framework to additionally obtain input authentication for computations supported by MP-SPDZ. An attractive feature of our implementation, which leverages the modularity of the compiler is that existing computations for MP-SPDZ framework work essentially unchanged with our extension. Similar extensions are also possible for other MPC frameworks such as SCALE-MAMBA [NUH+22]. We run protocols for authenticating inputs assuming a *broadcast channel*. We report communication complexity separately in terms of broadcast bits and point-to-point bits. 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. Additional details on our implementation and results appear in Section 6.

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 such as identifiable abort for certain restricted corruption settings, which neither [BJ18] nor [ADEO21] achieves. We refer the reader to Table 1 for an overview of features provided by our shceme in comparison to prior work.

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 guaran-

tees that the protocol runs to completion even in the presence of malicious behavior, which was not considered in prior works.

Fully Linear PCPs. 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. While techniques of [BBC+19] can indeed be used to achieve our goals, our focus is on concrete efficiency (prover overhead, communication overhead on top of the underlying unauthenticated MPC). In [BBC+19], the relation to be proved is expressed as an arithmetic circuit and for the languages we consider (algebraic relations), expressing them as a circuit is prohibitively expensive (for instance, modular exponentiation has size that is roughly cubic in the bit size of the modulus). 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. We also consider the notion of *robustness* which is not considered in the context of fully linear PCPs.

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 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 (for instance, modular exponentiation has size that is roughly cubic in the bit size of the modulus). 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.
- Our approach allows additional efficiency gains (e.g., aggregation of multiple signature verifications into a single proof of knowledge – see our optimized protocol in Section 3.5) which is likely to be significantly harder (and less efficient) using the FLPCP framework due to the need to generate a PCP over a distributed witness.

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 (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.

Notional Differences. Finally, we would like to point out a subtle difference between our distributed notion and that of [BBC+19]. In [BBC+19], the witness is with a single party and the proof (oracle) generation is centralized. In our notion, the witness is shared and the proof is generated in a distributed way. In [BBC+19], the verification is distributed and the verifiers interact with each other. In our notion, the verification is public. In particular, broadcast suffices and the verifiers do not have to interact with each other. We believe that both notions of distributed zero-knowledge are complementary.

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.

1.3 Technical Overview

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 $\alpha \in \mathbb{Z}_p$, 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

	Message Length				
Protocol	10^2	10^3	10^4	10^5	10^6
$\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$	0.5	0.9	1.2	1.6	1.9
BJ18 [BJ18]	1.5	17.5	168	1630	16300
ADEO21 [ADEO21]	0.9	0.9	0.9	0.9	0.9

Table 2: Communication overhead (in KB) for input authentication with abort for varying message lengths (reported per party for two party setting). Our overhead per party does not increase with the total number of parties, in contrast to prior work. Our overhead consists entirely of broadcast messages.

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

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.

Compression. Next, we 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 $\mathbf{w} \in \mathbb{Z}_p^m$ 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 $x' = x^c \prod A_i \prod B_i^{c^2}$; 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. Can we achieve a *robust* property where completeness holds as long as the shares reconstruct a valid witness, even if some provers are malicious? We show that this can be achieved 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 $\boldsymbol{c} \in \mathbb{Z}_p^m$, 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 1) for linear codes to claim that such a prover only succeeds with negligible probability. Unfortunately, the aforementioned "error preserving" property is provably known only for corruption bounded by a third of minimum distance ($d/3$), instead of the decoding radius of $d/2$. For the case of Shamir secret sharing, this downgrades our robustness threshold from $n/3$ corrupt provers to $n/4$. 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 (Section 2.3) and proof of knowledge of PS signatures (Appendix B), which leads to vastly more efficient distributed protocols.

2 Preliminaries

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

Let $\mathbb{G}$ be a group and $\mathbb{Z}_p$ 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 $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_p^n$, 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 $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_p^n$, $\mathbf{g}_L$ (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 3.4.

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 $q \geq t$, 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.*

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 $t-1$ 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 $[n, t, n-t+1]$ Reed Solomon code. 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 1. *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 $e < d/3$. 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.*

Variants of above Lemma are stated and proved in [AHIV17] for the bound $d/4$. It is also proved in [BCR+19], and independently in [DPP+22][Lemma A.5] for the bound $d/3$. Any improvement in the bound for the above Lemma implies higher tolerance for our robust protocols. For example, improving the bound to $d/2$ yields a robust protocol that tolerates upto $n/3$ corruptions, instead of $n/4$ claimed in this paper.

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 *proof of knowledge* 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.

2.3 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 4 (BBS+ Signature Scheme [BBS04,LKWL22]). *The BBS+ Signature Scheme to sign a message* $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ *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* $x \leftarrow_R \mathbb{Z}_p^*$*, 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* $\beta, s \leftarrow_R \mathbb{Z}_p$*, 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 $\mathbf{m} \in \mathbb{Z}_p^\ell$ 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 $r_1 \leftarrow_R \mathbb{Z}_p^*$ 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 $r_2 \leftarrow_R \mathbb{Z}_p$ 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.

2.4 Compressed Sigma Protocols

We recall the sigma protocol for vectors, for proving knowledge of discrete log $\mathbf{s} \in \mathbb{Z}_p^\ell$ 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 $\mathbf{r} \leftarrow_R \mathbb{Z}_p^\ell$, and sends $\alpha = \mathbf{g}^{\mathbf{r}}$ to the verifier $\mathcal{V}$. $\mathcal{V}$ then samples a challenge $c \leftarrow_R \mathbb{Z}_p$ 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 $\mathbf{x} \in \mathbb{Z}_p^\ell$ 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 :** $\mathbf{x} \in \mathbb{Z}_p^\ell$

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 $c \leftarrow_R \mathbb{Z}_p$ 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 $z' = A y^c B^{c^2}$.
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.

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 5 (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 $\mathcal{R}$.
- **Input Phase**: The prover $\mathcal{P}$ receives $(\mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$ as input, while the worker $\mathcal{W}_i$, $i \in [n]$ receives $(\mathbf{x}, \mathbf{s}_i)$ as input, where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. The verifier $\mathcal{V}$ receives $\mathbf{x}$ as input. Here we partition the witness as $(\mathbf{s}, \mathbf{t})$ to explicitly specify which parts of the witness needs to be shared.
- **Preprocessing Phase**: The prover $\mathcal{P}$ sends some auxiliary information aux_i to worker $\mathcal{W}_i$ 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 $(\mathsf{pp}, \mathbf{x}, \mathbf{s}, \mathbf{t})$ as $\mathcal{P}$'s input, $(\mathsf{pp}, \mathbf{x}, \mathbf{s}_i, \mathsf{aux}_i)$ as the input of $\mathcal{W}_i$ and $(\mathsf{pp}, \mathbf{x})$ as the input of $\mathcal{V}$. The verifier's message in each round $j \in [k]$ consists of a uniformly sampled challenge $\mathbf{c}_j \in \mathbb{F}^{\ell_j}$ for $\ell_j \in \mathbb{N}$. In each round $j \in [k]$, the worker $\mathcal{W}_i$ (resp. the prover $\mathcal{P}$) broadcasts a message $\mathbf{m}_{ij}$ (resp., $\mathbf{m}_i$) which depends on it's random coins and the messages received in prior rounds (including pre-processing phase).
- **Output Phase**: At the conclusion of k rounds, verifier outputs a bit $b \in \{0, 1\}$ indicating accept (1) or reject (0).

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

- **Completeness**: We say that completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$ and $(\mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$, 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.

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

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. 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 $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$ 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 $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, $(\mathbf{x}, (\mathbf{s}, \mathbf{t})) \in \mathcal{R}$ 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 $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$.

We will report the communication efficiency of our protocols as $(C_{\text{pp}}, C_{\text{bc}})$ where C_{pp} denotes the bits sent via point to point secret channels while C_{bc} denotes the total bits sent via public broadcast channel.

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 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 as detailed in Section 3.2.

Remark 3. Looking forward, 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.

3.2 Distributed Proof of Knowledge for Discrete Log

We now provide a construction of distributed proof of knowledge for the discrete log relation and Shamir Secret Sharing [Sha79]. Let $\mathsf{DlogGen}$ be a relation generator that on input $(1^\lambda, \ell)$ 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 $\mathcal{R}^{\text{DL}}$ is defined by $(z, \mathbf{s}) \in \mathcal{R}^{\text{DL}}$ if $\mathbf{g}^{\mathbf{s}} = z$. Let $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n) Shamir secret sharing over $\mathbb{Z}_p$.

We first present a distributed protocol for $\mathsf{DlogGen}$, namely $\Pi_{\mathsf{d\text{-}pok}}$, that satisfies t-privacy, 0-robustness and $(O(n), O(\ell n))$-efficiency. Further, we assume that $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = \sum_{i \in [n]} k_i \mathbf{s}_i$ for some $(k_1, \ldots, k_n) \in \mathbb{F}^n$. The only variation, apart from the workers running the classical Sigma protocol for the proof of knowledge over their respective shares in parallel, is the additional randomization of the first message of each prover using the additive share ρ_i of 0. This makes our simulator particularly simple in this case.

Protocol $\Pi_{\mathsf{d\text{-}pok}}$

– **Public Parameters**: $(\mathbb{G}, (\mathbf{g}, h), p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, \ell)$ defining relation $\mathcal{R}^{\text{DL}}$.
– **Input Phase**: $\mathcal{P}$ receives $\mathbf{s}$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\text{DL}}$. Each worker $\mathcal{W}_i$ receives $(z, \mathbf{s}_i)$ respectively, where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. The verifier receives z.
– **Preprocessing Phase**: The prover samples $(\rho_1, \ldots, \rho_n) \leftarrow_R \mathbb{F}^n$ uniformly subject to $\sum_{i \in [n]} \rho_i = 0$. It sends ρ_i to $\mathcal{W}_i$ over private secure channel.
– **Interactive Phase**:
  1. $\mathcal{W}_i$ $(i \in [n])$ samples $\mathbf{t}_i \leftarrow_R \mathbb{Z}_p^\ell$ and broadcasts $\alpha_i = \mathbf{g}^{k_i \mathbf{t}_i} h^{\rho_i}$.
  2. $\mathcal{V}$ chooses $c \leftarrow_R \mathbb{Z}_p$ and broadcasts c.
  3. $\mathcal{W}_i$ $(i \in [n])$ computes $\mathbf{x}_i = c\mathbf{s}_i + \mathbf{t}_i$ and broadcasts $\mathbf{x}_i$.
– **Output Phase**: $\mathcal{V}$ outputs 1 if $\mathbf{g}^{k_1 \mathbf{x}_i + \ldots + k_n \mathbf{x}_n} = z^c \cdot \alpha_1 \cdots \alpha_n$, 0 otherwise.

Theorem 1. *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, protocol $\Pi_{\mathsf{d\text{-}pok}}$ is a t-private $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the relation generator $\mathsf{DlogGen}$ and Shamir Secret Sharing Scheme achieving communication efficiency of $(O(n), O(\ell n))$.*

Proof. **Completeness.** We have $\mathbf{x}_i = c\mathbf{s}_i + \mathbf{t}_i$ and $\mathbf{s} = \sum_{i=1}^{n} k_i \mathbf{s}_i$. Thus, $\sum_{i=1}^{n} k_i \mathbf{x}_i = c\sum_{i=1}^{n} k_i \mathbf{s}_i + \sum_{i=1}^{n} k_i \mathbf{t}_i = c\mathbf{s} + \sum_{i=1}^{n} k_i \mathbf{t}_i$. Hence if $\mathbf{s}$ satisfies $\mathbf{g}^{\mathbf{s}} = z$, then $\mathcal{V}$ outputs 1 with probability 1.

Knowledge-Soundness. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be the adversary that results in that results in $\mathcal{V}$ accepting in the protocol. We describe the extractor Ext which extracts a valid witness, i.e. $\mathbf{s}$ such that $\mathbf{g}^{\mathbf{s}} = z$ as: The extractor runs the adversary $\mathcal{A}_2$ on different challenges till it obtains two accepting transcripts $\left(\{\alpha_i\}_{i\in[n]}, c, \{\mathbf{x}_i\}_{i\in[n]}\right)$ and $\left(\{\alpha_i\}_{i\in[n]}, c', \{\mathbf{x}'_i\}_{i\in[n]}\right)$ for two distinct challenges c and c', $c \neq c'$. Here, extractor generates the messages for the honest players using the shares $\mathbf{s}_i, i \notin \mathsf{C}$ provided to it. Moreover, Forking Lemma [PS96,BN06] ensures that Ext runs in polynomial time assuming computing discrete log is hard in $\mathbb{G}$. Then we have:

$$\mathbf{g}^{k_1\mathbf{x}_1+\ldots+k_n\mathbf{x}_n} = z^c \cdot \alpha_1 \cdots \alpha_n, \quad \mathbf{g}^{k_1\mathbf{x}'_1+\ldots+k_n\mathbf{x}'_n} = z^{c'} \cdot \alpha_1 \cdots \alpha_n$$

Defining $\mathbf{s}_i = (\mathbf{x}_i - \mathbf{x}'_i)/(c - c')$, we have $\mathbf{g}^{\mathbf{s}} = z$ for $\mathbf{s} = \mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n)$, as desired[6].

Zero Knowledge. We construct a simulator Sim that produces a transcript indistinguishable from the transcript of the protocol. WLOG, we assume $n \notin \mathsf{C} \subsetneq [n]$. Sim samples $c \leftarrow_R \mathbb{Z}_p$, $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$ and $\alpha_i \leftarrow_R \mathbb{G}$ for parties controlled by $\mathcal{A}$, i.e. for all $i \in \mathsf{C}$ such that $|\mathsf{C}| \leq t$. Sim also samples $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$, $\forall\ i \in [n], i \notin \mathsf{C}$, samples $\alpha_i \leftarrow_R \mathbb{G}$, $\forall\ i \in [n-1], i \notin \mathsf{C}$, and sets

$$\alpha_n = \mathbf{g}^{k_1\mathbf{x}_i+\ldots+k_n\mathbf{x}_n} / \{z^c \cdot \alpha_1 \cdots \alpha_{n-1}\}$$

The transcript output by the simulator is uniform subject to the verification constraint, and is thus distributed identically to the transcript of the protocol.

Efficiency. It is easily seen that the protocol incurs $O(n)$ communication over private channels, and $O(\ell n)$ communication over the broadcast channel. The workers also incur $O(\ell)$ exponentiations.

Public Verifiability Looking ahead, for our eventual compiler transforming any MPC protocol into a corresponding authenticated MPC protocol, we use (extensions of) a publicly verifiable version of $\Pi_{\mathsf{d\text{-}pok}}$. Note that $\Pi_{\mathsf{d\text{-}pok}}$ is public-coin, with multiple first round messages from different provers. As a result, we cannot make this protocol completely non-interactive using the standard Fiat-Shamir transformation [FS87] of Sigma protocols into NIZK proofs. However, we can compress $\Pi_{\mathsf{d\text{-}pok}}$ by one round while relying on the Fiat-Shamir heuristic to achieve a 2-round publicly verifiable version $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ as follows:

- In the first round, each prover $\mathcal{W}_i$ samples $\mathbf{t}_i \leftarrow_R \mathbb{Z}_p^\ell$ and broadcasts its first message $\alpha_i = \mathbf{g}^{k_i\mathbf{t}_i} h^{\rho_i}$.
- In the second round, each prover $\mathcal{W}_i$ computes $c = \mathsf{RO}\,(z \| \alpha_1 \| \cdots \| \alpha_n)$ and broadcasts its second round message $\mathbf{x}_i = c\mathbf{s}_i + \mathbf{t}_i$.

Verification proceeds exactly as in $\Pi_{\mathsf{d\text{-}pok}}$, and $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ retains the same communication complexity as $\Pi_{\mathsf{d\text{-}pok}}$. We now state the following theorem:

Theorem 2. *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, protocol $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ is a t-private $\mathsf{PV\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the relation generator $\mathsf{DlogGen}$ and Shamir Secret Sharing Scheme achieving communication efficiency of $(O(n), O(\ell n))$.*

Proof. Perfect completeness of $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ follows from perfect completeness of $\Pi_{\mathsf{d\text{-}pok}}$. For knowledge-soundness of $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$, we use the forking lemma [PS96][BN06] to extract two accepting transcripts of $\Pi_{\mathsf{d\text{-}pok}}$ from the adversary $\mathcal{A}$ controlling the set of workers $\{\mathcal{W}_i : i \in \mathsf{C}\}$. Note that we fork the adversary $\mathcal{A}$ on behalf of all the workers $\mathcal{W}_i, i \in \mathsf{C}$, as we need all the workers to respond with their respective valid last message on different challenges. We obtain two accepting transcripts from an adversary with probability close to ϵ^2/Q, where Q is total number of random oracle queries by the adversary controlling all the workers, and ϵ is the success probability of the adversary for providing an

[6] We note that, if a component of $\mathbf{s}_i$ is non-zero, say $(\mathbf{s}_i)_j \neq 0$ for some $j \in [n]$, then from $\mathbf{s}_i = (\mathbf{x}_i - \mathbf{x}'_i)/(c - c')$ we ensure that the two last message vectors in the transcript satisfy $(\mathbf{x}_i)_j \neq (\mathbf{x}'_i)_j$ for that j.

accepting transcript. The extraction of the witness thereafter follows from the knowledge-soundness and the forking lemma of the underlying protocol $\Pi_{\mathsf{d\text{-}pok}}$.

Finally, to argue that $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ is zero-knowledge, we construct the following simulator Sim to provide an accepting transcript, using only the knowledge of the statement (this simulator is essentially identical to the simulator for $\Pi_{\mathsf{d\text{-}pok}}$, except for the added advantage of RO oracle provided to the $\mathcal{A}$):

1. Sim simulates the random oracle RO as follows: it maintains a local table consisting of tuples of the form (x, y). On receiving a query x from the adversary $\mathcal{A}$, it looks up this table to check if an entry of the from (x, y) exists. If yes, it responds with y. Otherwise, it responds with a uniformly sampled y, and programs the random oracle as $\mathsf{RO}(x) := y$ by adding the entry (x, y) to the table.
2. Sim samples $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$ and $\alpha_i \leftarrow_R \mathbb{G}$ for parties controlled by $\mathcal{A}$, i.e. for all $i \in \mathsf{C}$ such that $|\mathsf{C}| \leq t$. Sim also samples $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$, $\forall\ i \in [n], i \notin \mathsf{C}$, samples $\alpha_i \leftarrow_R \mathbb{G}$, $\forall\ i \in [n-1], i \notin \mathsf{C}$, and sets $\alpha_n = \mathbf{g}^{k_1\mathbf{x}_i + \ldots + k_n\mathbf{x}_n} / \{z^c \cdot \alpha_1 \cdots \alpha_{n-1}\}$
3. Sim aborts if $\mathcal{A}$ has issued a query on $(z \| \alpha_1 \| \cdots \| \alpha_n)$. Otherwise, Sim programs $\mathsf{RO}(z \| \alpha_1 \| \cdots \| \alpha_n) := c$ and outputs $(\{\alpha_i\}_{i\in[n]}, c, \{\mathbf{x}_i\}_{i\in[n]})$.

We note that Sim runs in polynomial time, while maintaining a uniform distribution subject to the verification constraint for the transcript it outputs. Additionally, Sim aborts with only negligible probability, since the adversary $\mathcal{A}$ guesses each of $\alpha_1, \ldots, \alpha_n$ with at most negligible probability. This completes the proof of Theorem 2.

Succinct Distributed Protocol. The basic protocol presented previously incurred $O(\ell)$ communication per worker, due to each worker sending a vector of size ℓ as the final message. To reduce this communication, we use a distributed version of *split and fold* technique used earlier in [BBB$^+$18] and [AC20] to compress classical Sigma protocols for proof of knowledge. Instead of sending vectors $\mathbf{x}_i$ in the final message, the workers instead prove knowledge of vectors $\mathbf{x}_i$, whose reconstruction $\mathbf{x}$ opens the commitment on the right hand side of verification constraint. The aforementioned (distributed) proof of knowledge is reduced in each round to a (distributed) proof of knowledge over smaller vectors until the vectors are succinct enough to be revealed in full (typically when they are of size 2). Our protocol $\Pi_{\mathsf{d\text{-}csp}}$ is detailed below. Note that this protocol is not required to be zero knowledge, as revealing $\mathbf{x}_i$ in the original protocol leaks no information.

Protocol $\Pi_{\mathsf{d\text{-}csp}}$

- **Public Parameters**: $(\mathbb{G}, \mathbf{g}, p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, \ell)$ defining relation $\mathcal{R}^{\mathrm{DL}}$.
- $\mathcal{P}$**'s inputs**: $\mathbf{s}$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$.
- $\mathcal{W}_i$**'s inputs :** $(z, \mathbf{s}_i)$, where $\mathbf{s}_i$ is the i^{th} share of the message vector $\mathbf{s}$, such that $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = \mathbf{s}$.
- **Interactive Phase**:
  1. $\mathcal{W}_i$ divides $\mathbf{g} = (\mathbf{g}_L, \mathbf{g}_R)$ where $\mathbf{g}_L$ and $\mathbf{g}_R$ denote the first and second halves of the vector $\mathbf{g}$ respectively. Similarly, it obtains vectors $\mathbf{s}_{i,L}$ and $\mathbf{s}_{i,R}$ from the vector $\mathbf{s}_i$. It broadcasts commitments $A_i = \mathbf{g}_R^{k_i \mathbf{s}_{i,L}}$ and $B_i = \mathbf{g}_L^{k_i \mathbf{s}_{i,R}}$.
  2. $\mathcal{V}$ chooses $c \leftarrow_R \mathbb{Z}_p$ and broadcasts c.
  3. $\mathcal{W}_i$ computes $\mathbf{s}'_i = \mathbf{s}_{i,L} + c \cdot \mathbf{s}_{i,R}$.
  4. All the parties compute new generators $\mathbf{g}' = \mathbf{g}_L^c \circ \mathbf{g}_R$ and new commitment $z' = (A_1 \cdots A_n) \cdot z^c \cdot (B_1 \cdots B_n)^{c^2}$.
  5. If size$(\mathbf{g}') = 2$: $\mathcal{W}_i$, $i \in [n]$ send $\mathbf{s}'_i$ to $\mathcal{V}$, else the parties repeat the steps from Step 1 with $\mathbf{s}_i = \mathbf{s}'_i$, $z' = z$ and $\mathbf{g} = \mathbf{g}'$, effectively running the interactive phase on the reduced statement $(\mathbf{g}', z')$.
- **Output Phase**: $\mathcal{V}$ outputs 1 if $\mathbf{g}'^{k_1\mathbf{s}'_1 + \cdots + k_n\mathbf{s}'_n} = z'$, else it outputs 0.

Theorem 3. *The protocol $\Pi_{\mathsf{d\text{-}csp}}$ satisfies completeness and knowledge-soundness for relation generator* $\mathsf{DlogGen}$ *and Shamir Secret Sharing Scheme while incurring $O(n \log \ell)$ bits of communication over broadcast channel. The combined protocol $\Pi_{\mathsf{cd\text{-}pok}}$ obtained by composing $\Pi_{\mathsf{d\text{-}pok}}$ and $\Pi_{\mathsf{d\text{-}csp}}$ is a* DPoK *with t-privacy for* $\mathsf{DlogGen}$ *and (t, n)-Shamir Secret Sharing with communication efficiency of $(O(n), O(n \log \ell))$.*

Proof. **Completeness.** The completeness follows from the following calculation which shows that if the workers have shares $(\mathbf{s}_1, \ldots, \mathbf{s}_n)$ of the correct witness $\mathbf{s}$, then the workers end up with shares $(\mathbf{s}'_1, \ldots, \mathbf{s}'_n)$ whose reconstruction $\mathbf{s}'$ satisfies the reduced statement $\mathbf{g}'^{\mathbf{s}'} = z'$.

$$\begin{aligned}\mathbf{g}'^{k_1\mathbf{s}'_1+\ldots+k_n\mathbf{s}'_n} &= (\mathbf{g}_L^c \circ \mathbf{g}_R)^{\sum_i k_i\mathbf{s}'_i} = (\mathbf{g}_L^c \circ \mathbf{g}_R)^{\sum_i k_i(\mathbf{s}_{i,L}+c\mathbf{s}_{i,R})} \\ &= \prod_i \left(\mathbf{g}_R^{\mathbf{s}_{i,L}}\right)^{k_i} \cdot \left(\mathbf{g}_L^{c\mathbf{s}_{i,L}}\mathbf{g}_R^{c\mathbf{s}_{i,R}}\right)^{k_i} \cdot \left(\mathbf{g}_L^{c^2\mathbf{s}_{i,R}}\right)^{k_i} \\ &= \prod_i A_i \cdot (\mathbf{g}^{k_i\mathbf{s}_i})^c \cdot B_i^{c^2} = \prod_i A_i \cdot z^c \cdot \prod_i B_i^{c^2} = z'.\end{aligned}$$

Knowledge-Soundness. We prove knowledge-soundness of (a single execution of the recursive relation in the) protocol along the lines of the same proof for the single prover version in [AC20]. We again obtain several accepting transcripts by running the adversary, and using the shares of honest parties to simulate the messages from them as in the proof of Theorem 1. Let $(\{A_i, B_i\}_{i\in[n]}, c_1, \{s_i'^1\}_{i\in[n]})$, $(\{A_i, B_i\}_{i\in[n]}, c_2, \{s_i'^2\}_{i\in[n]})$, and $(\{A_i, B_i\}_{i\in[n]}, c_3, \{s_i'^3\}_{i\in[n]})$ be three accepting transcripts. Let a_1, a_2 and a_3 be such that $a_1 + a_2 + a_3 = 0$, $c_1a_1 + c_2a_2 + c_3a_3 = 1$ and $c_1^2a_1 + c_2^2a_2 + c_3^2a_3 = 0$. Note that such (a_1, a_2, a_3) can be computed as the associated coefficient matrix is invertible provided c_1, c_2, c_3 are distinct, which happens with overwhelming probability. Define $\mathbf{s}_i = \sum_j a_j(c_j\mathbf{s}_i'^j, \mathbf{s}_i'^j)$ and consider $\mathbf{s} = k_1\mathbf{s}_1 + \cdots + k_n\mathbf{s}_n$. For $j \in [3]$, we have:

$$\mathbf{g}'_j = \mathbf{g}_L^{c_j} \circ \mathbf{g}_R, \quad \mathbf{g}_j'^{k_1\mathbf{s}'_1+\cdots+k_n\mathbf{s}'_n} = (A_1 \cdots A_n) \cdot z^{c_i} \cdot (B_1 \cdots B_n)^{c_i^2}$$

Raising the respective equations to power a_i and then multiplying, we get:

$$\begin{aligned}z = \prod_{j=1}^{3} \mathbf{g}'^{a_j(k_1\mathbf{s}'_1+\cdots+k_n\mathbf{s}'_n)} &= \mathbf{g}_L^{\sum_j a_jc_j(k_1\mathbf{s}'_1+\cdots+k_n\mathbf{s}'_n)}\mathbf{g}_R^{\sum_j a_j(k_1\mathbf{s}'_1+\cdots+k_n\mathbf{s}'_n)} \\ = \mathbf{g}^{k_1\mathbf{s}_1+\cdots+k_n\mathbf{s}_n} &= \mathbf{g}^{\mathbf{s}}.\end{aligned}$$

Thus, the vector $\mathbf{s}$ as constructed is a witness for the original relation.

Efficiency. The workers and verifier perform $O(\ell)$ operations over $\mathbb{G}$ (exponentiations). The communication (broadcast) incurred by each worker is $O(\log \ell)$ bits.

Final Compressed DPoK for Discrete Log. The final compressed distributed protocol $\Pi_{\mathsf{cd\text{-}pok}}$ for the relation $\mathcal{R}^{\mathrm{DL}}$ is obtained by composition of protocols $\Pi_{\mathsf{d\text{-}pok}}$ and $\Pi_{\mathsf{d\text{-}csp}}$ in the following way: once the workers compute the vectors $\mathbf{x}_i$ in Step 3 of the protocol $\Pi_{\mathsf{d\text{-}pok}}$ (3.2), instead of broadcasting $\mathbf{x}_i$, they run the interactive protocol of the protocol $\Pi_{\mathsf{d\text{-}csp}}$ using $\mathbf{s}_i = \mathbf{x}_i$ as their shares. The output of $\Pi_{\mathsf{d\text{-}csp}}$ on the these shares is considered the output of $\Pi_{\mathsf{cd\text{-}pok}}$. Finally, we can analogously achieve a publicly verifiable version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{cd\text{-}pok}}$, by using $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ as the base protocol and compressing it using the compressed Sigma protocol $\Pi_{\mathsf{d\text{-}csp}}$ described above.

3.3 Robust Complete DPoK for Discrete Log

In this section, we build upon the protocols presented so far to simultaneously achieve both succinctness and *robust completeness*. In particular, we present a protocol Π_{rob} which is a DPoK for relation generator $\mathsf{DlogGen}$ and a (t, n)-Shamir Secret Sharing while satisfying d-robustness for $d \leq (n-t)/3$ and achieves $(O(n), O(n \log \ell))$-efficiency. We detail the protocol below.

Protocol Π_{rob}

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 r_i to $\mathcal{W}_i$ for $i \in [n]$ where $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ for $r \leftarrow_R \mathbb{Z}_p$.
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 $\omega_i \leftarrow_R \mathbb{Z}_p$. 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 $\boldsymbol{\gamma} \leftarrow_R \mathbb{Z}_p^\ell$, 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 protocls (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 $v'_i \leftarrow_R \mathbb{Z}_p$ (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. Since $\Delta(\mathbf{v}', \mathbf{v}) \le d < (n-t)/2$, $\mathcal{V}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Let C denote indices of corrupt workers (who actually deviate from the protocol). From Lemma 1 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 $[n] \backslash \mathsf{C}$.
7. **Output using honest messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\prod_{j \in [q]} A_{i_j}^{k'_j} = z$, and $(0, \{\mathcal{P}\})$ otherwise.

Theorem 4. *The protocol described above is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* DlogGen *and* (t, n)*-Shamir Secret Sharing scheme which satisfies* t*-privacy and* d*-robustness, for* $d < \mathsf{dist}/3$*, where* $\mathsf{dist} = (n - t)$ *is the minimum distance of the Reed-Solomon code induced by the* SSS *scheme.*

Proof. We note that standard (non-robust) completeness is straightforward to verify, while succinctness follows immediately by instantiating NIPK using the Fiat-Shamir transformed CSPs from [AC20]. 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 $i_j \in \mathsf{C}$; 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 $\mathbf{s}' = \sum_{j \in [q]} k'_{i_j} \mathbf{s}'_{i_j}$ satisfies $\mathbf{g}^{\mathbf{s}'} = z$. The extractor outputs $\mathbf{s}'$. For proving zero-knowledge, we assume WLOG that $\mathsf{C} = \{1, \ldots, \epsilon\}$ for $\epsilon \le t$. 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 $1 \le i \le t$. Set $\mathbf{a} = (z, A'_1, \ldots, A'_t)$. Next, Sim sets $A'_{t+j} = \mathbf{a}^{\mathbf{t}_j}$ where $\mathbf{t}_j \in \mathbb{Z}_p^{t+1}$ is such that $f(t+j) = \langle (f(0), \ldots, f(t)), \mathbf{t}_j \rangle$ for all polynomials $f(x)$ of degree $\le t$. 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 for $j > \epsilon$ is that in real protocol $(A_j)_{j > \epsilon}$ are distributed as $(\mathbf{g}^{\mathbf{f}(j)})_{j > \epsilon}$, where $\mathbf{f} = (f_1, \ldots, f_\ell)$ denotes a vector of polynomials each of degree at most t chosen uniformly subject to $\mathbf{g}^{\mathbf{f}(0)} = z$. We use the notation $\mathbf{f}(j)$ to denote the vector $(f_1(j), \ldots, f_\ell(j))$. Sampling such a polynomial 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 suitable 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 $\boldsymbol{\gamma} \leftarrow_R \mathbb{Z}_p^\ell$. 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 $v' \leftarrow_R \mathbb{Z}_p$, 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 $\mathcal{A}$. The completeness relies on the following observation from Lemma 1. If a corrupt worker $\mathcal{W}_i$ deviates from the protocol by outputting v'_i which is inconsistent with the shares $\mathbf{s}_i, r_i$ it received from $\mathcal{P}$, with overwhelming probability, $v_i \neq v'_i$ where $\mathbf{v} = (v_1, \ldots, v_n)$ is determined by error-correcting the received vector $\mathbf{v}' = (v'_1, \ldots, v'_n)$. Thus the corresponding message A_i is omitted from the verification check. Since there are sufficient honest workers, the reconstruction still remains feasible.

Robust Completeness. We show that when the prover is honest, and has a correct witness $\mathbf{s}$, the verifier outputs 0 with negligible probability, identifying the corrupt workers. Again, let $\mathcal{A}$ be an adversary corrupting set C' of workers with $|\mathsf{C}'| = d < (n-t)/3$. We consider the following hybrid: We consider a simulator Sim, which for each $i \in \mathsf{C}'$ where the proofs π_{i1} and π_{i2} in Step 4 are valid

extracts $(\mathbf{s}'_i, r'_i, \omega'_i)$ from the proofs π_{i1} and π_{i2} and then sends messages $(A'_i, B'_i, \pi'_{i1}, \pi'_{i2})$ to $\mathcal{V}$, where $A'_i = \mathbf{g}^{\mathbf{s}'_i}$, $B'_i = h_1^{r'_i} h_2^{\omega'_i}$ and π'_{i1} and π'_{i2} are simulated proofs on statements determined by A'_i and B'_i. The simulator Sim forwards other messages (corresponding to invalid proofs) as is, and from this point onwards runs the adversary $\mathcal{A}$ by forwarding verifier's challenges to it. By knowledge-soundness and zero-knowledge property of NIPK proofs, the probability of $\mathcal{A}$ resulting in failed verification is negligibly close to the corresponding probability in the original protocol. We bound the probability of $\mathcal{A}$ causing verification to fail in the modified game. Let $T \subseteq \mathsf{C}'$ denote the set such that all proofs $\pi_{i1}, \pi_{i2}, \pi_{i3}$ output by $\mathcal{A}$ are valid for $i \in T$. Let $\mathbf{S}'$ denote the matrix with i^{th} column as: $(\mathbf{s}_i, r_i)$ for $i \notin \mathsf{C}'$, $(\mathbf{s}'_i, r'_i)$ for $i \in T$ and sampled uniformly from $\mathbb{Z}_p^{\ell+1}$ otherwise. Let S denote the matrix with i^{th} column as $(\mathbf{s}_i, r_i)$ for all $i \in [n]$. Let $E \subseteq [n]$ denote the column indices where S and S' differ. By assumption, $|E| \leq (n-t)/3$. Now, the reconstruction in Step 7 fails only if one of the positions in E is not identified as part of set C in Step 6, i.e, there exists $i \in E$ such that $v'_i = v_i$ in Step 6. With overwhelming probability we must have $i \in T$, as for $i \in \mathsf{C}' \backslash T$, the verifier chooses v'_i uniformly at random. However, $i \in T$ implies that $v_i = [\boldsymbol{\gamma}, 1]^T \mathsf{S}$, $v'_i = [\boldsymbol{\gamma}, 1]^T \mathsf{S}'$ (this holds due to soundness of proof π_{i3}) and thus the vectors $[\boldsymbol{\gamma}, 1]^T \mathsf{S}$ and $[\boldsymbol{\gamma}, 1]^T \mathsf{S}$ agree in position i, while the matrices S and S' differ in the i^{th} column. By Lemma 1, this occurs with negligible probability. This completes the proof of d-robustness.

Remark 4. Note that the bound for d and t stated in the theorem statement ensures that, given the total parties corrupted by the adversary $t < n/4$, we can ensure that we achieve both robust-completeness along with privacy against the adversary.

Publicly Verifiable Version of Protocol Π_{rob}. We now state a publicly verifiable version of Π_{rob}; once again, we rely on the Fiat-Shamir heuristic [FS87] and a random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{Z}_p^\ell$. We call this publicly verifiable version of the protocol $\Pi_{\mathsf{rob}}^{\mathsf{pv}}$. Note that, similar to $\Pi_{\mathsf{d\text{-}pok}}$, Π_{rob} is also a public-coin protocol with multiple first rounds messages from distributed provers. As a result, similar to $\Pi_{\mathsf{d\text{-}pok}}$, Π_{rob} cannot be made completely non-interactive using the standard Fiat-Shamir transformation [FS87] of interactive protocols into NIZK proofs. Instead, we transform Π_{rob} into $\Pi_{\mathsf{rob}}^{\mathsf{pv}}$ as follows:

- In the first round, each prover W_i computes $A_i = \mathbf{g}^{\mathbf{s}_i}$, $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{Z}_p$ and broadcasts $\{A_i, B_i\}$
- In the second round, each prover W_i does the following:
  1. Query the random oracle RO on the concatenation of all first round messages to compute
  $$\boldsymbol{\gamma} = \mathsf{RO}\left(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n\right) \in \mathbb{Z}_p^\ell.$$
  2. Broadcast the second round message $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$.
  3. Simultaneously, broadcast NI-ZKPoKs of openings of A_i, B_i and an NI-ZKPoK of $\mathbf{w}_i$, which opens the commitment $A_i^\gamma \cdot B_i$ and the (publicly computable) linear form $v_i = \langle (\boldsymbol{\gamma}, 1, 0), \mathbf{w}_i \rangle$.

Note that the protocol already satisfies succinctness and robust completeness (this is immediate from the corresponding properties of the underlying Π_{rob} protocol). We argue knowledge-soundness and zero-knowledge for the modified protocol below.

Soundness and Zero-Knowledge. The arguments of soundness and zero-knowledge for $\Pi_{\mathsf{rob}}^{\mathsf{pv}}$ follow in a straightforward way. In particular, we argue soundness by invoking the extractors for the NI-ZKPoKs. We argue zero-knowledge 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_{rob}).

3.4 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 6 (Threshold Linear Secret Sharing Scheme). *A (t, n, r) threshold linear secret-sharing (TLSS) scheme over a finite field $\mathbb{F}$ consists of algorithms ($\mathsf{Share}, \mathsf{Reconstruct}$) as described below:*

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

A TLSS *scheme satisfies the following properties:*

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

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

Distributed Proof of Knowledge using Threshold Linear Secret Sharing 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 $\mathsf{TLSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n, r) threshold linear secret sharing over $\mathbb{F} = \mathbb{Z}_p$. We follow the framework presented in Section 3.2 for DlogGen; namely $\Pi_{\mathsf{d\text{-}pok}}$ (Figure 3.2), that satisfies t-privacy, 0-robustness and $(O(rn), O(r\ell))$-efficiency. We present our generalized protocol $\Pi_{\mathsf{d\text{-}pok\text{-}tlss}}$ (Figure 3.4) with the same guarantees. We note that the only variation arises due to the usage of different (Share, Reconstruct) algorithm. Here, Reconstruct algorithm uses reconstruction vector of length $\mathbf{k} = (k_{11}, \ldots, k_{nr})$ of length nr where the coefficients $k_i = (k_{i,1}, \ldots, k_{i,r})$ are associated with the share $\mathbf{s}_i = (\mathbf{s}_{i1}, \ldots, \mathbf{s}_{ir})$ held by i^{th} party. Accordingly, the protocol below exactly replicates $\Pi_{\mathsf{d\text{-}pok}}$, with the difference being that each party broadcasts r components as part of first and third message, one corresponding to each share.

Protocol $\Pi_{\mathsf{d\text{-}pok\text{-}tlss}}$

– **Public Parameters**: $(\mathbb{G}, (\mathbf{g}, h), p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, \ell)$ defining relation $\mathcal{R}^{\mathrm{DL}}$.
– **Input Phase**: $\mathcal{P}$ receives $\mathbf{s}$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$. Each worker $\mathcal{W}_i$ receives $(z, \mathbf{s}_i)$ respectively, where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$. The verifier receives z.
– **Preprocessing Phase**: The prover samples $(\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_n) \leftarrow_R (\mathbb{F}^r)^n$ uniformly subject to $\sum_{i \in [n]} \rho_{ij} = 0$ for all $j \in [r]$. Here $\boldsymbol{\rho}_i = (\rho_{i1}, \ldots, \rho_{ir})$. It sends $\boldsymbol{\rho}_i$ to $\mathcal{W}_i$ over private secure channel.
– **Interactive Phase**:
  1. $\mathcal{W}_i$ $(i \in [n])$ for each $j \in [r]$: samples $\mathbf{t}_{ij} \leftarrow_R \mathbb{Z}_p^\ell$ and broadcasts $\alpha_{ij} = \mathbf{g}^{k_{ij}\mathbf{t}_{ij}} \cdot h^{\rho_{ij}}$.
  2. $\mathcal{V}$ chooses $c \leftarrow_R \mathbb{Z}_p$ and broadcasts c.
  3. $\mathcal{W}_i$ $(i \in [n])$ for each $j \in [r]$: computes $x_{ij} = c\mathbf{s}_{ij} + \mathbf{t}_{ij}$ and broadcasts $\mathbf{x}_i = (x_{i1}, \ldots, x_{ir})$
– **Output Phase**: $\mathcal{V}$ outputs 1 if $\mathbf{g}^{\mathsf{Reconstruct}(\mathbf{x}_1, \ldots, \mathbf{x}_n)} = z^c \cdot \alpha_{11} \cdots \alpha_{nr}$, 0 otherwise.

We can also obtain protocol for the above relation with succinct communication, i.e with $(O(rn), O(rn \log l))$-efficiency, for a (t, n, r)-TLSS scheme by applying the compression technique discussed earlier in the protocol $\Pi_{\mathsf{d\text{-}csp}}$. The public verifiable versions also follow analogously.

Theorem 5 (Distributed Proof of Knowledge for Discrete Log for TLSS). *Assuming that the discrete log assumption holds over the group* $\mathbb{G}$*, there exists a* DPoK *for relation generator* DlogGen *and an* (n, t, r)-TLSS *scheme which achieves* $(O(nr), O(nr \log \ell))$*-efficiency.*

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 2 (below), which generalizes Lemma 1 to the case when linear code is over an extension field $\mathbb{F}_{p^r} \cong \mathbb{F}_p^r$ of the field $\mathbb{F} = \mathbb{F}_p$. Throughout, we assume (Share, Reconstruct) to define a (t, n, r)-TLSS scheme and $\mathbb{F} = \mathbb{F}_p$ to be the finite field of order p.

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 7 (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*

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

consisting of all possible sharings output by the Share *algorithm.*

We now state the generalization of Lemma 1 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 2. *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_{rob} 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 preceeding 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 $\mathbb{F}^n_{p^r}$. 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 2 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 $\prod_{j \in [q], t \in [r]} A_{i_j,t}^{k'_{j_t}} = z$, and $(0, \{\mathcal{P}\})$ otherwise.

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

The proof of the above theorem is similar to that for the protocol Π_{rob}, except that we use Lemma 2 instead of Lemma 1 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.

3.5 (Corollary) Distributed Proof of Knowledge using Replicated Secret Sharing

Our earlier results obtained for Shamir Secret Sharing [Sha79] in Theorem 4 can be seen as special case of Theorem 6 for $r = 1$ and $\mathsf{dist} = (n - t)$. Here we additionally specialise Theorem 6 to the case of *replicated secret sharing*. We recall the definition of Replicated Secret Sharing (RSS) Scheme provided in [Esc22].

Definition 8 (Replicated Secret Sharing Scheme). *A $(t, n, \binom{n-1}{t})$ replicated linear secret-sharing (RSS) scheme over a finite field $\mathbb{F}$ consists of algorithms ($\mathsf{Share}, \mathsf{Reconstruct}$) as described below:*

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

A RSS scheme satisfies the following properties:

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

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

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

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

Remark 7. We note that the corruption threshold of $t < n/4$ 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 $\mathbb{F}_{p^k}$.

4 DPoK for BBS+ Signatures

The publicly verifiable variants of our protocols in the previous section enabled the following scenario: Given n-parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and a designated prover (say $\mathcal{P}_1$), it allows $\mathcal{P}_1$ to secret-share its input $\mathbf{s}_1$ and convince the other participants that $\mathbf{s}_1$ satisifies a publicly known discrete-log relation. By parallel repetition of n-instances of the protocol, each party can convince others that its input satisfies a known discrete-log constraint (we note that $\mathcal{V}$'s role can be performed by all parties in publicly verifiable variant). The end-goal of this section is to extend the methods of previous section, to allow parties to claim that: "*the input that I have secret-shared has a valid signature (BBS+) with respect to a public key* pk". We then use this distributed protocol to design our final 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+ signatures as above. We use the variant of BBS+ signature scheme from [CDL16], which was adapted from earlier schemes in [BBS04,ASM06] (a construction using PS signatures [PS16] also appears in the Appendix B). We start by defining a relation relevant to BBS+ signature verification.

Definition 9 (BBS+ Relation). *Let* BBSGen *denote the relation generator, such that* $\mathsf{BBSGen}(1^\lambda, \ell)$ *outputs a bilinear group* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R BBS.\mathsf{Setup}(1^\lambda)$. *The corresponding relation* $\mathcal{R}^{\text{bbs}}$ *is defined by* $(\mathbf{x}, (\mathbf{m}, \mathbf{t})) \in \mathcal{R}^{\text{bbs}}$ *for* $\mathbf{x} = \mathsf{pk} = (g_1, w, h_0, \ldots, h_\ell) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1^\ell$, $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ *and* $\mathbf{t} = \sigma = (A, \beta, s) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$ *if* $e(A, wg_2^\beta) = e(g_1 h_0^s \prod_{i=1}^\ell h_i^{m_i}, g_2)$.

4.1 DPoK for BBS+ Signatures

We first present the protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$, which allows a designated prover $\mathcal{P}$, to show knowledge of a BBS+ signature (A, β, s) over the message $\mathbf{m} \in \mathbb{Z}_p^\ell$ that is secret-shared amongst the workers $\mathcal{W}_1, \ldots, \mathcal{W}_n$. For ease of exposition, we describe the protocol assuming an honest verifier $\mathcal{V}$. The single prover variant underlying the distributed variant involves: (i) prover randomly chooses auxiliary inputs, and combines them with the signature to output a randomised first message, and then (ii) it shows knowledge of these auxiliary inputs and components of the signature satisfying discrete-log relations determined from the first message. The DPoK for the same follows a similar blueprint: The prover shares the "auxiliary" inputs it uses to randomise the first message, and then broadcasts the first message (identical to the single-prover variant). The subsequent discrete-log relation is now proved in distributed manner using DPoK for the $\mathsf{DlogGen}$ relation from the previous section. We prove the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ to be a DPoK for the relation generator BBSGen in the following lemma.

Lemma 3. *The protocol* $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{BBSGen}}$ *for the relation generator* BBSGen *and Shamir Secret Sharing scheme.*

Proof. We provide a proof-sketch. Completeness follows using direct calculation, and the completeness of the DPoK protocols for $\mathsf{DlogGen}$ used as sub-protocols in steps (2) and (3). For knowledge-soundness, we describe the extractor Ext for an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which corrupts $\mathcal{P}$ and $\mathcal{W}_i$, $i \in \mathsf{C}$. On input $\{\mathbf{m}_i\}_{i \notin \mathsf{C}}$, where $\mathbf{m}_i$ is the share of $\mathbf{m}$ provided to $\mathcal{W}_i$, the extractor Ext proceeds as follows: 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 extractors Ext' and Ext'' for DPoK sub-protocols in steps (2) and (3) respectively and computes 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-protocols and verifier's checks, with overwhleming 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 preceeding 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 $\mathbf{m}$. The extractor Ext outputs the same.

For proving zero-knowledge, assume that the adversary $\mathcal{A}$ corrupts workers $\mathcal{W}_i$,$i \in \mathsf{C}$ where $|\mathsf{C}| \leq t$. The simulator Sim proceeds as follows: 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 preceeding message, the simulation follows by invoking simulators for the respective DPoKs on the statements derived from the simulated first message.

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

- **Public Parameters**: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R \mathsf{BBSGen}(1^\lambda)$ defining BBS+ relation $\mathcal{R}^{\text{bbs}}$. Let $\mathsf{pk} = (g_1, w = g_2^x, h_0, \ldots, h_\ell)$ be a known public key for secret key $\mathsf{sk} = x \leftarrow_R \mathbb{Z}_p$.
- **Input Phase**: $\mathcal{P}$ receives public key pk, message-vector $\mathbf{m} \in \mathbb{Z}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\mathbf{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$. For $i \in [n]$, $\mathcal{W}_i$ recieves the i^{th} share $\mathbf{m}_i$ of the message vector $\mathbf{m}$, where $(\mathbf{m}_1, \ldots, \mathbf{m}_n) \leftarrow_R \mathsf{Share}(\mathbf{m})$.
- **Pre-processing Phase** : $\mathcal{P}$ samples $u \leftarrow_R \mathbb{Z}_p^*, r \leftarrow_R \mathbb{Z}_p, \eta \leftarrow_R \mathbb{Z}_p$, 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 $i \in [n]$.
- **Interactive Phase:**
  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 protocol $\Pi_{\mathsf{cd\text{-}pok}}$ 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; and $\mathbf{g} = (h_0, \ldots, h_\ell)$, $z = D$ is available to all parties.
  3. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the protocol $\Pi_{\mathsf{d\text{-}pok}}$ 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 $C \cdot D = g_1^{-1}$, $e(A', w) = e(\bar{A}, g_2)$, $\Pi_{\mathsf{cd\text{-}pok}}$ and $\Pi_{\mathsf{d\text{-}pok}}$ accept.

Finally, we can again achieve a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$, by relying on the Fiat-Shamir heuristic and using a random oracle.

Extension for Authenticating All Inputs. The protocol for setting where all the parties show a valid signature over their inputs is implied by the parallel repetition of n-instances of the protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. We however, present an optimized variant in $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$, where n-parallel instances of the DPoK protocol in Step (2) of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ can be replaced by one instance of the same (with negligible loss of soundness) using a random challenge γ. In $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ this reduction is performed in Step (2), resulting in single instance of DPoK protocol in Step (3).

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

- **Public Parameters**: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R \mathsf{BBSGen}(1^\lambda)$ defining BBS+ relation $\mathcal{R}^{\text{bbs}}$. Let $\mathsf{pk} = (g_1, w = g_2^x, h_0, \ldots, h_\ell)$ be a known public key for secret key $\mathsf{sk} = x \leftarrow_R \mathbb{Z}_p$.
- P_i**'s inputs**:
  - Message $\mathbf{m}_i \in \mathbb{Z}_p^\ell$ and signature $\sigma_i = (A_i, \beta_i, s_i)$ on $\mathbf{m}_i$ under pk.
  - i^{th} share of the message $\mathbf{m}_j$ of P_j.
- **Pre-processing**: $\mathcal{P}_i$ samples $u_i \leftarrow_R \mathbb{Z}_p^*, r_i \leftarrow_R \mathbb{Z}_p, \eta \leftarrow_R \mathbb{Z}_p$, and computes $d_i = b_i^{u_i} \cdot h_0^{-r_i}$ and $t_i = s_i - r_i \cdot v_i$ where $v_i = u_i^{-1}, b_i = g_1 h_0^{s_i} \prod_{i=1}^{\ell} h_i^{m_i}$. and secret shares $r_i, v_i, t_i, \eta_i, \beta_i$ among $P_1, \ldots, P_n$. All parties set $\mathbf{g} = (h_0, \ldots, h_\ell)$.
- **Interactive Protocol**

1. $\mathcal{P}_i$, $i \in [n]$ computes $A'_i = A_i^{u_i}$, $\bar{A}_i = (A')^{-\beta} \cdot b^u (= (A')^x)$. $\mathcal{P}$ sets $C_i = d_i^{-v_i} h_0^{t_i - \eta_i}$, $D_i = \mathbf{g}^{\eta_i, \mathbf{m}_i}$, and broadcasts $(A'_i, \bar{A}_i, d_i, C_i, D_i)$.
2. Each P_i, $i \in [n]$ computes challenge $\gamma \leftarrow_R \mathbb{Z}_p$ by querying the Random Oracle RO on $(A_i || \bar{A}_i || d_i || C_i || D_{i_i})$, and computes $\mathbf{y}_i = \sum_{j \in [n]} \gamma^j (\eta_{ij}, \mathbf{m}_{ij})$, where $\eta_{ij}, \mathbf{m}_{ij}$ denotes $\mathcal{P}_i$'s share of $\mathcal{P}_j$'s inputs $\mathbf{m}_j, \eta_{ij}$.
3. All parties compute $D = \prod_{j \in [n]} D_j^{\gamma^j}$.

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

4. Parties run the interactive phase of the protocol $\Pi_{\mathsf{cd\text{-}pok}}$ on statement D with $\mathbf{g}$ as the generator. They run the interactive phase of the protocol $\Pi_{\mathsf{d\text{-}pok}}$ on statements $C_i = d_i^{-v_i} h_0^{t_i - \eta_i} \wedge (A'_i)^{-\beta_i} h_0^{r_i} = \frac{\bar{A}_i}{d_i}$, for each $i \in [n]$ with generators (d_i, h_0) and (A'_i, h_0) respectively.
5. Parties also check that $e\left(\prod_{i=1}^n A'_i, w\right) = e\left(\prod_{i=1}^n \bar{A}_i, g_2\right)$ holds.

– **Output**: P_j outputs $b_j = 1$ if all the above protocols lead to accept.

Robust Complete DPoK for BBS+ Signatures. In addition, robust variants of the protocols for verifying BBS+ signatures follows from using robust variants of underlying DPoKs. We defer the detailed descriptions of the protocols to Appendix A.2. We note that for robust variant, the optimisation of reducing n-parallel instances of DPoK to one instance is not possible. By definition, a protocol with robust completeness should identify the set of malicious parties, which is seemingly difficult to achieve if we combine n instances of the underlying protocol into a single instance using a random challenge.

5 Compiler for MPC with input authentication

In this section, we present our compiler for MPC with input authentication that builds upon our distributed (robust complete) proofs of knowledge for BBS+ signatures. We begin by presenting a non-robust version of our compiler that realizes a weaker ideal functionality for MPC with input authentication, and then present the final compiler that achieves the desired ideal functionality.

5.1 Our (Non-Robust) Compiler for MPC with input authentication

We describe the non-roust of our compiler that (informally speaking) takes as input any secret-sharing-based MPC protocol Π_{mpc} and outputs a corresponding secret-sharing based MPC protocol Π_{ampc} realizing a weaker ideal functionality for authenticated MP, as described below.

The Weaker Ideal Functionality. We formally describe below the ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth,abort}}$, which is a weaker version of the desired ideal functionality for MPC with input authentication in the sense that it only captures abort security (as opposed to id-abort/GOD security).

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

Inputs

The ideal functionality receives from each party P_i an input-signature pair of the form (x_i, σ_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 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 $\mathbf{x}_i \in \mathbb{Z}_p^\ell$ 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. Let $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ denote the publicly verifiable version of our protocol allowing parties to prove authenticity of their inputs to each other by proving knowledge of a valid BBS+ signature under pk on their inputs (recall that this protocol runs n underlying instances of $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$, where for instance i, party P_i acts as the prover and all other parties P_j for $j \neq i$ act as verifiers; see Section B.3 for details). For simplicity of exposition, we first present a version of our compiler using the un-optimized protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$. Our compiler can be easily extended to use the significantly more optimized $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$. We discuss the optimized version subsequently.

Our Compiler. Given $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$ and $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ as defined above, we design an MPC protocol with input authentication $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$ as described 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 $\mathbf{x}_i \in \mathbb{Z}_p^\ell$, together with a signature σ_i, such that $(\mathsf{pk}, (\mathbf{x}_i, \sigma_i)) \in \mathcal{R}^{\mathrm{bbs}}$.
- $\overline{\Pi}_{\mathsf{sh}}$: This phase is identical to Π_{sh}, i.e., each party P_i shares its input $\mathbf{x}_i$ to all other parties exactly as in Π_{sh}.
- $\overline{\Pi}_{\mathsf{on}}$: In this phase, the parties do the following:
  - For each $j = 1, \ldots, n$, the parties execute an instance of $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ for $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \in \mathcal{R}^{\mathrm{bbs}}$ with $\mathcal{P}_j$ acting as the Prover, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ constituting the workers and $\mathcal{P}_i$, $i \neq j$ acting as verifiers, . If any party outputs 0 at the end of this phase, the protocol aborts.
  - Otherwise, the parties jointly execute Π_{on}.

We now state and prove the following theorem for the security of Π_{ampc}.

Theorem 8 ((Non-Robust) Security of Π_{ampc}). *Assuming that: (a) the MPC protocol Π_{mpc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, and (b) $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ is knowledge-sound and zero-knowledge, our compiled MPC protocol with input authentication Π_{ampc} securely emulates the ideal functionality $\mathcal{F}^{\mathsf{auth},\mathsf{abort}}_{\mathsf{MPC}}$.*

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. Let $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ be the simulator guaranteed by the ZK property of the underlying $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol.

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:

- Simulate the sharing phase $\overline{\Pi}_{\mathsf{sh}}$ 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 $P_j, j \in \mathcal{C}$.
- For each P_j s.t. $j \in \mathcal{C}$, let $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_j$ denote the instance of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ used by the parties where P_j acts as the prover, and all of the remaining parties acting as both workers and verifiers.

- To simulate the online phase, for each party $P_j, j \in \mathcal{H}$, invoke $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ to simulate the view of the adversary in a (distributed) proof of knowledge of a BBS+ signature $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ where P_j acts as the prover, and all remaining parties act as both workers and verifiers.
- For each instance $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$, where a corrupt party P_i is acting as the prover, invoke the extractor Ext on $(\mathbf{s}^i_j)_{j\in\mathcal{H}}$ to extract the witness $(\mathbf{x}_i, \sigma_i)$ from P_i.
- Invoke $\mathsf{Sim}_{\mathsf{on}}$ to simulate the online phase.
- Send $\{(\mathbf{x}_i, \sigma_i)\}_{i\in\mathcal{C}}$ to $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$. If $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$ aborts, abort; otherwise output whatever $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$ 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 we use $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ to simulated distributed proof of knowledge corresponding to the input of honest party $P_{|\mathcal{C}|+j}$. 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 $\sigma_{|\mathcal{C}|+j}$, we simulate a distributed proof of knowledge of a BBS+ signature by using $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ to simulate the instance of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ 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 $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ where P_j is the prover, invoke the extractor Ext on $(\mathbf{s}^j_i)_{i\in\mathcal{H}}$ to extract the witness $(\mathbf{x}_j, \sigma_j)$. If $(\mathsf{pk}, (\mathbf{x}_j, \sigma_j)) \notin \mathcal{R}^{\mathrm{bbs}}$, 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 $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$ 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 the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol. 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 an adversary $\mathcal{A}'$ that breaks the ZK property of the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol.

$\mathsf{Hyb}_{3,j-1} \equiv \mathsf{Hyb}_{3,j}$**.** This follows from knowledge soundness of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. The two hybrids differ only when the bad event occurs. The probability of the bad event occurring is negligible by knowledge soundness. Thus, conditioned on the bad event not occurring, $\mathsf{Hyb}_{3,j-1}$ and $\mathsf{Hyb}_{3,j}$ are identical.

$\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_{3,|\mathcal{C}|}$**.** 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 outputing 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 $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$ 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 8.

Protocol	Communication Complexity	Computational Complexity
BJ18 [BJ18]	$O(\ell n \log \lvert\mathbb{G}\rvert)$	$9\ell n$ exp + $2\ell n$ pairings.
ADEO21 [ADEO21]	$\approx 9n^2 \log \lvert\mathbb{G}\rvert$	$O(\ell n)$ exp + $O(n)$ pairings
$\Pi_{\mathsf{bbs\text{-}auth}}$ (our work)	$O(n^2 \log \ell \log \lvert\mathbb{G}\rvert)$	$O(\ell n)$ exp. + $O(n)$ pairings.
$\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ (our work)	$\approx 2n \log \ell \log \lvert\mathbb{G}\rvert$	$O(\ell + n)$ exp + $O(1)$ pairings.

Table 3: Overhead for input authentication with abort for different protocols. The communication reported is total communication across all parties. The computational overhead is per party. The complexities are reported for authenticating each participant's input of size ℓ, whereas n denotes the number of parties. As before, our communication overhead consists entirely of broadcast messages.

Optimized Version of Our Compiler. In the above description, we presented a version of our compiler using the un-optimized protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ for simplicity of exposition. Our compiler can be easily extended to use the significantly more optimized $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$ for proving authenticity of inputs. We omit a formal description and proof as they are conceptually very similar to our un-optimized compiler described above.

Performance and Efficiency. We summarize in Table 3 the overheads incurred by our non-robust compiler and compare it with the overheads incurred by existing approaches for achieving MPC protocols with authenticated inputs [BJ18,ADEO21] (all of which are also non-robust and achieve comparable security guarantees with our non-robust compiler).

5.2 The Final (Robust) Compiler

In this section, we present our final compiler for MPC with input authentication that builds upon our distributed (robust complete) proofs of knowledge for BBS+ signatures. In particular, the robust completeness property of the underlying DPoK allows the protocol to identify all malicious parties with non-authenticated inputs. Note, however, that for the DPoK to achieve robust completeness, we can tolerate a maximum corruption threshold of $t < n/4$ (assuming that the secret-sharing used is Shamir's secret sharing); hence, our compiled MPC protocol also tolerates a maximum corruption threshold of $t < n/4$.

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

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

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

Our Robust Compiler and its Security. Given $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$, our robust compiler outputs an authenticated MPC protocol $\Pi_{\mathsf{ampc\text{-}rob}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$, which is identical to Π_{ampc} output by the non-robust version of our compiler described above, except that we use the protocol $\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}$ from Section A.2 as the robust complete DPoK for authenticating the BBS+ signatures on all of the inputs (as opposed to the $\Pi_{\mathsf{bbs\text{-}auth}}$ protocol used by the non-robust version of our compiler).

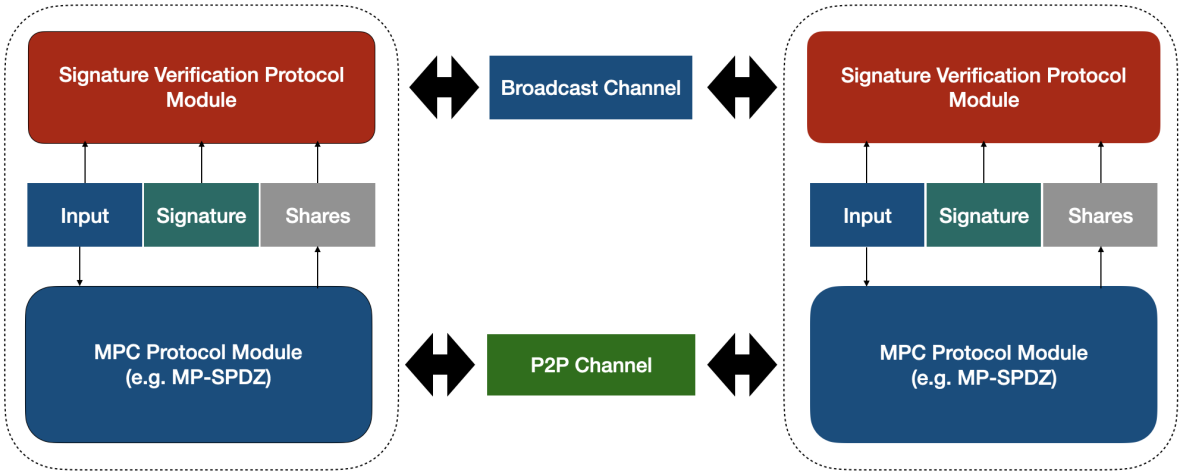

Fig. 1: Component diagram illustrating integration of our scheme with existing MPC frameworks.

	n = 3	n = 5
N = 125	650MB/11sec	2113MB/11sec
	7KB/0.16sec	11KB/0.18sec
N = 250	1367MB/19sec	4326MB/20sec
	7KB/0.3sec	11KB/0.3sec

Table 4: Computational and communication overhead for verifying signature on the biometric database for database sizes of 125 and 250, for 3 and 5 parties respectively. The numbers in blue denote overall communication and time for unauthenticated computation, while that in black denotes the corresponding overhead due to input authentication.

Theorem 9 (Security of $\Pi_{\mathsf{ampc\text{-}rob}}$). *Assuming that: (a) the MPC protocol Π_{mpc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, and (b) $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}rob}}$ satisfies soundness, zero-knowledge and robust completeness for a maximum corruption threshold of $t < n/4$, our compiled MPC protocol with input authentication $\Pi_{\mathsf{ampc\text{-}rob}}$ securely emulates the ideal functionality $\mathcal{F}^{\mathsf{auth}}_{\mathsf{MPC}}$ for the same corruption threshold of $t < n/4$.*

The description and security proof of our robust compiler are very similar to that of the non-robust case, and are hence not detailed.

Remark 8. In the robust case, 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).

6 Implementation and Experiments

We leverage the modularity of our compiler to obtain a modular prototype implementation of authenticated MPC. Our implementation easily extends to support any existing MPC framework that supports linear secret-sharing based protocols, such as MP-SPDZ [Kel20] and SCALE-MAMBA [NUH+22] among others. Here, we present an instance of authenticated MPC by augmenting the popular MP-SPDZ [Kel20] library with our DPoK for BBS+ signatures; concretely, we implemented our DPoK for BBS+ signatures on top of the MP-SPDZ implementation of maliciously secure MPC in the honest majority setting based on Shamir's secret sharing. Our implementation allows any existing computations expressed using MP-SPDZ tooling to additionally support input authentication essentially unchanged. We only depend on the underlying framework to expose interface to access shares of the parties inputs. See Figure 1 for a component diagram illustrating how our implementation integrates our proposed DPoK for BBS+ signatures with existing (linear secret-sharing based) MPC frameworks to achieve authentication of inputs on top of the underlying MPC protocol.

The public parameters of BBS+ signature are generated over BN254 [PSNB10] curve, which supports a prime order group of 254 bits. We compile the MP-SPDZ circuits against the same prime modulus. To illustrate the practical viability of our approach, we use a moderately complex computation provided with the MP-SPDZ distribution on determining the closest biometric match for a given sample, in a given biometric database. The computation involves a party supplying the database of N samples, modelled as $N \times 4$ matrix. Another party supplies a target sample (a, b, c, d) and the computation outputs the squared euclidean distance to the closest sample. In our experiments with 3 and 5 parties respectively, we introduce remaining parties without any inputs to the computation, while signature verification is performed for the biometric database. We make no changes to the original specification of the computation, except adding a library call to export the shares of the inputs corresponding to the database.

The overheads over the vanilla (unauthenticated) computation using malicious shamir secret sharing are summarized in Table 4. It highlights that our overheads of our approach are negligible for moderately complex computations. To obtain the results of Table 4, we only recurse in the compressed sigma protocol till the size of the witness vector is ≥ 64. This saves us some rounds of communication and computation time, at the cost of slightly larger communication. Also see Tables 1, 2 and 3 for a comparison of our approach with prior works. In conclusion, our approach presents a comprehensive progress over existing works on authenticated MPC in terms of ease of integration with existing tooling, asymptotic considerations as well as practical performance.

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.

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.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

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.

Kel20. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1575–1590. ACM Press, November 2020.

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.

NUH+22. NigelSmart, Idoia Gamiz Ugarte, Ben Hamlin, Asif Mallik, and Dragoș Rotaru. idoiagamiz/scale-mamba: v1.0.0, 2022.

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.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

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.

PSNB10. Geovandro C. C. F. Pereira, Marcos A. Simplício Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. Cryptology ePrint Archive, Report 2010/429, 2010. `https://eprint.iacr.org/2010/429`.

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 Distributed PoK for BBS+ Signatures: Additional Details

This section presents additional details on distributed PoK protocols for BBS+ signatures.

A.1 DPoK for BBS+ Signatures: Straightforward Version

We use the distributed protocol $\Pi_{\mathsf{cd\text{-}pok}}$ to provide a proof of knowledge for BBS+ signature where a set of distributed provers $(\mathcal{W}_1, \ldots, \mathcal{W}_n)$ with access to the shares of a message vector $\mathbf{m}$, show proof of knowledge of signature on $\mathbf{m}$ with respect to a public key pk.

Protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$

- **Public Parameters**: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R \mathsf{BBSGen}(1^\lambda)$ defining BBS+ relation $\mathcal{R}^{\text{bbs}}$. Let $\mathsf{pk} = (g_1, w = g_2^x, h_0, \ldots, h_\ell)$ be a known public key for secret key $\mathsf{sk} = x \leftarrow_R \mathbb{Z}_p$.
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{Z}_p^\ell$ 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 $u \leftarrow_R \mathbb{Z}_p^*, r \leftarrow_R \mathbb{Z}_p$, 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 $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$, $(v_1, \ldots, v_n) \leftarrow_R \mathsf{Share}(v)$, $(\beta_1, \ldots, \beta_n) \leftarrow_R \mathsf{Share}(\beta)$, $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$. $\mathcal{P}$ sends the shares (r_i, v_i, β_i, t_i) to $\mathcal{W}_i$, for all $i \in [n]$.
- **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 $d = b^u \cdot h_0^{-r}$. $\mathcal{P}$ broadcasts $(A', \bar{A}, d)$ to each $\mathcal{W}_i$, and $\mathcal{V}$.
  2. Each $\mathcal{W}_i$ locally holds the i-th share $\mathbf{s}_i = (v_i, t_i, \mathbf{m}_i, \beta_i, r_i)$ such that
  $$\mathbf{s} = (v, t, \mathbf{m}, \beta, r) = \mathsf{Reconstruct}\left(\{\mathbf{s}_i\}_{i \in [n]}\right).$$
  3. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the protocol $\Pi_{\mathsf{cd\text{-}pok}}$ for the relation $d^{-v} h_0^t \prod_{i=1}^\ell h_i^{m_i} = g_1^{-1} \wedge (A')^{-\beta} h_0^r = \frac{\bar{A}}{d}$, where $(v, t, m_1, \ldots, m_\ell)$ and (β, r) is secret-shared; $(\mathbf{g} = (d, h_0, \ldots, h_\ell), z = g_1^{-1})$ and $(\mathbf{g}' = (A', h_0), z' = \frac{\bar{A}}{d})$ is available to all parties.
  4. $\mathcal{V}$ accepts if the $\Pi_{\mathsf{cd\text{-}pok}}$ in the previous step accepts, and $e(A', w) = e\left(\bar{A}, g_2\right)$ holds.

Theorem 10. *Assuming that the discrete log assumption holds over the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the proposed protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$ is a t-private $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{BBSGen}}$ for the relation generator BBSGen and Shamir Secret Sharing Scheme.*

Proof. The proof is very similar to the proof of Theorem 1 and is omitted.

Efficiency. The protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$ inherits its communication complexity essentially from the underlying protocol $\Pi_{\mathsf{cd\text{-}pok}}$ which is $O(\log(\ell)\log(|\mathbb{G}|)+\log(|\mathbb{Z}_p|))$ per worker and $O(n\log(\ell)\log(|\mathbb{G}|))$ overall.

A.2 Robust Complete DPoK for BBS+ Signatures

We build upon the robust complete DPoK Π_{rob} for discrete log to propose a DPoK achieving robust completeness for BBS+ signatures. The protocol is called $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$, and is essentially identical to its non-robust counterpart $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$, but additionally achieves robust completeness by using the robust complete protocol Π_{rob} as opposed to the non-robust protocols $\Pi_{\mathsf{cd\text{-}pok}}$ and $\Pi_{\mathsf{d\text{-}pok}}$, in steps 2 and 3 of the interactive phase of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. The detailed protocol is described below, with the changes from $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ highlighted in red.

Protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$

- **Public Key** $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ 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 $u \leftarrow_R \mathbb{Z}_p^*, r \leftarrow_R \mathbb{Z}_p, \eta \leftarrow_R \mathbb{Z}_p$, 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 $\mathcal{W}_i$.
  2. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the protocol Π_{rob} 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; and $\mathbf{g} = (h_0, \ldots, h_\ell)$, $z = D$ is available to all parties.
  3. The workers $\mathcal{W}_i, i \in [n]$ and $\mathcal{V}$ run the protocol Π_{rob} 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 $C \cdot D = g_1^{-1}$, $e(A', w) = e\left(\bar{A}, g_2\right)$ and Π_{rob} accept.

We can again construct a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$, by relying on the Fiat-Shamir heuristic and using a random oracle.

Extension for Authenticating All Inputs. The robust complete protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$ works in a setting where a designated prover $\mathcal{P}$ proves authenticity of its input by sharing it among the workers $\mathcal{W}_1, \ldots, \mathcal{W}_n$. We can again extend this protocol for usage in an MPC protocol where all the parties need to establish authenticity of their inputs to each other. The simple extension, involving n parallel invocations of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$, is called $\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}$, This protocol is very similar in flavor to its non-robust counterpart $\Pi_{\mathsf{bbs\text{-}auth}}$, and we avoid explicitly detailing it for brevity. As in all prior protocols, we can also construct a publicly verifiable two-round version of $\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}$, which we call $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}rob}}$.

We do not have a counterpart of the optimized variant $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ in the case of robust completeness. By definition, a protocol with robust completeness should identify the set of malicious parties, which is seemingly difficult to achieve if we combine n instances of the underlying protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}$ into a single instance using a random challenge.

B Authentication using PS Signatures

In this section we show the generality of techniques shown above by providing distributed protocols for another pairing-based signature scheme, whose proof of knowledge of signature also reduces to discrete logarithm relation. We begin by recalling the Pointcheval Sanders (PS) signature scheme from [PS16], along with the associated proof of knowledge. For our authenticated MPC protocol, we use to use a distributed version of the PS signature-based proof of knowledge to allow a set of distributed provers $\mathcal{W}_i$, $i \in [n]$ holding shares $\mathbf{s}_i \in \mathbb{Z}_p^\ell$ of a secret input vector $\mathbf{s} \in \mathbb{Z}_p^\ell$ to prove knowledge of a PS signature on $\mathbf{s}$ (here, the j^{th} component of $\mathbf{s}_i$ contains the i^{th} share of the j^{th} component of $\mathbf{s}$). We first describe the non-distributed proof of knowledge, and then show how to design a distributed version of the same.

PS Signatures. At a high level, the (multi-message version of) the Pointcheval-Sanders (PS) signature scheme [PS16] works as follows. Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be the description of an efficiently computable non-degenerate bilinear map where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q (q being a λ-bit prime for security parameter λ). The PS signature scheme uses a signing and verification key pair $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} = (x, y_1, \ldots, y_\ell)$, $\mathsf{pk} = \left(\tilde{g}, \tilde{X} := \tilde{g}^x, \tilde{Y}_1 := \tilde{g}^{y_1}, \ldots, \tilde{Y}_\ell := \tilde{g}^{y_\ell}\right)$ for $x, y_1, \ldots, y_\ell \leftarrow_R \mathbb{Z}_p$ and $\tilde{g} \leftarrow_R \mathbb{G}_1$. The signing algorithm takes as input the signing key sk and a message vector $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$, and outputs a signature $\sigma = \left(\sigma_1 := h, \sigma_2 := h^{x + \sum_j y_j m_j}\right)$, where $h \leftarrow_R \mathbb{Z}_p$. Finally, the verification algorithm takes as input the public verification key pk, a signature σ, and a message vector $\mathbf{m}$, and outputs 1 if $\sigma_1 \neq \mathsf{e}_1$ (where e_1 is the identity element for the group $\mathbb{G}_1$) and $e\left(\sigma_1, \tilde{X} \cdot \prod_j \tilde{Y}_j^{m_j}\right) = e(\sigma_2, \tilde{g})$. Otherwise it outputs 0.

Note that PS Signatures are *re-randomizable* since given a valid signature $\sigma = (\sigma_1, \sigma_2)$ on a message vector $\mathbf{m}$ under a key-pair $(\mathsf{sk}, \mathsf{pk})$, we can publicly compute a re-randomized valid signature on the same message $\mathbf{m}$ under the same key-pair $(\mathsf{sk}, \mathsf{pk})$ as $\sigma' = (\sigma_1^r, \sigma_2^r)$ for $r \leftarrow_R \mathbb{Z}_p$. We refer to Appendix B for a more formal exposition.

Definition 10 (PS Signature Scheme [PS16]). *The PS Signature Scheme to sign a message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ 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 $(x, y_1, \ldots, y_\ell) \leftarrow_R \mathbb{Z}_p^{n+1}$, 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.

B.1 Proof of Knowledge

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

– **Public Key** $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$
– $\mathcal{P}$**'s inputs**: Message $\mathbf{m} \in \mathbb{Z}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\mathbf{m}$
  1. $\mathcal{P}$ samples $r, t \leftarrow_R \mathbb{Z}_p$ 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 $\mathbf{s} \in \mathbb{Z}_p^\ell$ 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**: $\mathbf{s} \in \mathbb{Z}_p^\ell$.
  1. $\mathcal{P}$ samples $\mathbf{r} \leftarrow_R \mathbb{Z}_p^\ell$ and computes $\alpha = g^\mathbf{r}$.
  2. $\mathcal{P} \rightarrow \mathcal{V}$: α.
  3. $\mathcal{V} \rightarrow \mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p$.
  4. $\mathcal{P} \rightarrow \mathcal{V}$: $\mathbf{s}' = c\mathbf{s} + \mathbf{r}$.
  5. $\mathcal{V}$ checks: $g^{\mathbf{s}'} = \alpha z^c$.

We also describe another variant of PS Signature Scheme, based on a stronger assumption (Assumption 1 in [PS16]), that leads to much more efficient distributed prover protocols. This variant is same as the one described in Definition 10, 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 11 (PS Signature: B [PS16]). *The PS Signature Scheme to sign a message* $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ *consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *as described in Definition 10, except* KeyGen *which is described below:*

– $\mathsf{KeyGen}(\mathsf{pp})$*: The algorithm samples* $g \leftarrow_R \mathbb{G}_1$, $\tilde{g} \leftarrow_R \mathbb{G}_2$, $(x, y_1, \ldots, y_{\ell+1}) \leftarrow_R \mathbb{Z}_p^{\ell+1}$ *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.*

B.2 Alternate Proof of Knowledge

We describe a protocol for showing knowledge of a PS signature (σ_1, σ_2) on a message $\mathbf{m} \in \mathbb{Z}_p^\ell$ 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 $\mathbf{m} \in \mathbb{Z}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\mathbf{m}$
  1. $\mathcal{P}$ samples $r, t, s \leftarrow_R \mathbb{Z}_p$ 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 $\mathbb{Z}_p^\ell$. Using the extractors for the ZKPoKs, $\mathcal{E}$ obtains $(m_1, \ldots, m_\ell, t, s)$ such that

$$C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}, \quad e(\sigma'_2, \tilde{g}) = e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, C) \cdot e(Y_{\ell+1}, \tilde{g})^s$$

The extractor $\mathcal{E}$ computes $\left(\sigma_1 = \sigma'_1, \sigma_2 = \sigma'_2 (\sigma'_1)^{-t} (Y_{\ell+1})^{-s}\right)$. To see that (σ_1, σ_2) is a valid signature we verify:

$$\begin{aligned} e(\sigma_2, \tilde{g}) &= e(\sigma'_2, \tilde{g}) \cdot e(\sigma'_1, \tilde{g})^{-t} \cdot e(Y_{\ell+1}, \tilde{g})^{-s} \\ &= e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, C) \cdot e(\sigma'_1, \tilde{g})^{-t} \\ &= e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \\ &= e(\sigma_1, \tilde{X} \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \end{aligned}$$

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

B.3 Distributed PoK for PS Signatures

The ZKPoK for PS signatures outlined in Section 2 assumes a *single* prover holding a valid PS signature. A core technical centerpiece of this paper is a distributed version of this ZKPoK, where (informally speaking) multiple provers, each holding a secret-share of the message and a PS signature on the message, can together prove knowledge of the (message, signature) pair with respect to a public verification key. We then use this distributed protocol to design our final 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 PS signatures, and the parties prove knowledge of the same.

Concretely, we use the distributed protocol $\Pi_{\mathsf{cd\text{-}pok}}$ to provide proof of knowledge for PS signature over an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where a set of distributed provers $(\mathcal{W}_1, \ldots, \mathcal{W}_n)$ with access to the shares of a message vector $\mathbf{m}$, show proof of knowledge of signature on $\mathbf{m}$ with respect to a public key pk. The solution essentially follows from casting the proof of knowledge of a signature as proof of opening of commitment and leveraging the protocol $\Pi_{\mathsf{cd\text{-}pok}}$. The details of reformulating knowledge of a PS signature as proof of opening of a commitment appears in Appendix B.1. The protocol is detailed below.

Protocol $\Pi_{\mathsf{ps\text{-}dpok}}$

- **Public Key** $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$
- $\mathcal{P}$**'s inputs**: Message $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ 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 $t \leftarrow_R \mathbb{Z}_p$, 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 $r \leftarrow_R \mathbb{Z}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r)$.
  2. $\mathcal{P}$ broadcasts the computed value $\sigma' = (\sigma'_1, \sigma'_2)$ to $\mathcal{V}$.
  3. Each $\mathcal{W}_i$ and $\mathcal{V}$ locally computes $\mathbf{g} = (g_1, \ldots, g_\ell, g_{\ell+1}), z$ where $g_1 = e(\sigma'_1, \tilde{Y}_1), \ldots, \quad g_\ell = e(\sigma'_1, \tilde{Y}_\ell)$, $g_{\ell+1} = e(\sigma'_1, \tilde{g})$ and $z = e(\sigma'_2, \tilde{g}) / e(\sigma'_1, \tilde{X})$.
  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. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the protocol $\Pi_{\mathsf{cd\text{-}pok}}$ for the relation $\mathbf{g}^{\mathbf{s}} = z$, where $\mathbf{s}$ is secret-shared and $(\mathbf{g}, z)$ is available to all parties.
  6. $\mathcal{V}$ accepts if the $\Pi_{\mathsf{cd\text{-}pok}}$ in the previous step accepts.

Theorem 11. *Assuming that the discrete log assumption holds over the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, protocol $\Pi_{\mathsf{ps\text{-}dpok}}$ is a t-private $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the relation generator $\mathsf{DlogGen}$ and Shamir Secret Sharing Scheme.*

Proof. The proof is very similar to the proof of Theorem 1 and is omitted.

Efficiency. The protocol $\Pi_{\mathsf{ps\text{-}dpok}}$ inherits its communication complexity essentially from the underlying protocol $\Pi_{\mathsf{cd\text{-}pok}}$ which is $O(\log(\ell)\log(|\mathbb{G}|) + \log(|\mathbb{Z}_p|))$ per worker and $O(n\log(\ell)\log(|\mathbb{G}|))$ overall. Computationally, each worker in $\Pi_{\mathsf{ps\text{-}dpok}}$ incurs additional overhead of computing $\ell + 1$ pairings over the computational complexity of the protocol $\Pi_{\mathsf{cd\text{-}pok}}$.

Optimized Version. The previous protocol suffers from the computational expense of computing $\ell + 1$ pairings as the generators for the sub-protocol $\Pi_{\mathsf{cd\text{-}pok}}$ need to be computed by all the workers based on the first message. To mitigate this computational burden, we consider another formtation of proving knowledge of a PS signature where proof of knowledge is over the generators available from the public key pk. This formulation is detailed in Appendix B.2. The improved protocol $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$ appears in Figure B.3. As we shall see, the improved protocol also leads to a vastly more efficient protocol when multiple parties need to show knowledge of signatures on their inputs.

Protocol $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$

- **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 $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ 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 $t, v \leftarrow_R \mathbb{Z}_p$, computes $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$, $(v_1, \ldots, v_n) \leftarrow_R \mathsf{Share}(v)$. $\mathcal{P}$ sends the shares (t_i, v_i) to $\mathcal{W}_i$, for all $i \in [n]$. All the parties set $\mathbf{g} = (\tilde{g}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$ and $h = e(Y_{\ell+1}, \tilde{g})$.
- **Interactive Protocol**
  1. $\mathcal{P}$ samples $r, v \leftarrow_R \mathbb{Z}_p$ 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 NI-ZKPoK 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 compute $z = e(\sigma_2', \tilde{g}) e(\sigma_1', \tilde{X})^{-1} e(\sigma_1', C)^{-1}$. Parties hold shares of $\mathbf{s} = (\mathbf{m}, t)$ and v satisfying $\mathbf{g}^{\mathbf{s}} = C$ and $h^v = z$
  4. All $\mathcal{W}_i$ for $i \in [n]$ and $\mathcal{V}$ run ZKPoK protocol $\Pi_{\mathsf{cd\text{-}pok}}$ for the relation $\mathbf{g}^{\mathbf{s}} = C$ and protocol $\Pi_{\mathsf{d\text{-}pok}}$ for the relation $h^v = z$.
  5. $\mathcal{V}$ accepts if both the protocols accept.

Finally, we can again achieve a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps\text{-}dpok\text{-}opt}}$ that achieves soundness and zero-knowledge, by relying on the Fiat-Shamir heuristic and using a random oracle.

B.4 Extensions for Authenticating All Inputs

The protocols thus far have been described in a setting where a designated prover $\mathcal{P}$ proves authenticity of its input $\mathbf{m}$ by sharing it among the workers $\mathcal{W}_1, \ldots, \mathcal{W}_n$. Looking ahead, in a multiparty computation involving parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, with $\mathbf{m}_i$ as $\mathcal{P}_i'$s input; all the parties can establish authenticity of their inputs to each other by running n invocations of the protocols $\Pi_{\mathsf{cd\text{-}pok}}$ or $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$. In an invocation a party $\mathcal{P}_j$ acts as the Prover, the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ collectively act as workers while each $\mathcal{P}_i$, $i \neq j$ also acts as a Verifier (this is possible, since the transcript is publicly verifiable as shown in Section 3.2). Typically, the input authentication will be followed by computation phase to compute a function f on the inputs. We defer those details to Section 5.1. For now we describe a protocol that authenticates inputs of all parties (against a common public key pk for simplicity). The protocol $\Pi_{\mathsf{ps\text{-}auth}}$ is described in Figure B.4.

Protocol $\Pi_{\mathsf{ps\text{-}auth}}$

- **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$.
- P_i**'s inputs**:
  - Message $\mathbf{m}_i \in \mathbb{Z}_p^\ell$ and signature σ_i on $\mathbf{m}_i$ (under pk).
  - i^{th} share of the message $\mathbf{m}_j$ of P_j.
- **Interactive Protocol**:
  1. For $j = 1, \ldots, n$:
  2. Run phase j in which parties execute an instance of $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$ 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.
- **Output**: Party $\mathcal{P}_j$ outputs $b_j = 1$ if it successfully verifies the transcript for phases $i \neq j$.

The communication complexity of the above protocol is simply $O(n^2 \log(\ell))$ corresponding to n invocations of $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$. The computational effort of a party is similarly $O(\ell n)$ exponentiations and $O(n)$ pairings.

Finally, it is straightforward to see that we can achieve a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps\text{-}auth}}$ that achieves soundness and zero-knowledge, by running n instances of the publicly verifiable protocol $\Pi^{\mathsf{pv}}_{\mathsf{ps\text{-}dpok\text{-}opt}}$ outlined earlier as opposed to $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$.

Optimized Version. We now present an optimized variant of the protocol $\Pi_{\mathsf{ps\text{-}auth}}$ which reduces the communication complexity to $O(n \log \ell)$ and exponentiations to $O(\ell+n)$. We achieve this by combining n instances of the sub-protocol $\Pi_{\mathsf{cd\text{-}pok}}$ (one corresponding to each instance of $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$) into a single instance of $\Pi_{\mathsf{cd\text{-}pok}}$, using a random challenge. Observe that in phase j of the above protocol, the parties run an instance of $\Pi_{\mathsf{cd\text{-}pok}}$ to prove $\mathbf{g}^{\mathbf{m}_j} \cdot \tilde{g}^{t_j} = C_j$ where $\mathbf{m}_j$ is $\mathcal{P}'_j$s private input, t_j is commitment randomness, and C_j is the commitment broadcast by $\mathcal{P}_j$ in Step 2 of $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$. Using a randomly sampled $\gamma \in \mathbb{Z}_p$, we can (with overwhelming probability) combine the proofs for all $j \in [n]$ to a single proof showing $\mathbf{g}^{\mathbf{s}} \cdot \tilde{g}^t = \prod_j C_j^{\gamma_j}$. The parties can compute shares of satisfying $\mathbf{s} = \sum_j \gamma^j \mathbf{m}_j$ and $t = \sum_j \gamma^j t_j$ using their shares of $\mathbf{m}_j, t_j$ for $j \in [n]$.

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

- **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$.
- P_i**'s inputs**:
  - Message $\mathbf{m}_i \in \mathbb{Z}_p^\ell$ and signature σ_i on $\mathbf{m}_i$ (under pk).
  - i^{th} share of the message $\mathbf{m}_j$ of P_j.
- **Pre-processing**: P_i, $i \in [n]$ samples $t_i, v_i \leftarrow_R \mathbb{Z}_p$ and secret shares (t_i, v_i) among $P_1, \ldots, P_n$. All parties set $\mathbf{g} = (\tilde{Y}_1, \ldots, \tilde{Y}_n, \tilde{g})$, $h = e(Y_{\ell+1}, \tilde{g})$.
- **Interactive Protocol**
  1. $\mathcal{P}_i$, $i \in [n]$ computes $\sigma'_i = (\sigma_{i,1}^{r_i}, (\sigma_{i,2} \cdot \sigma_{i,1}^{t_i})^{r_i} \cdot Y_{\ell+1}^{v_i})$ for $r_i \leftarrow_R \mathbb{Z}_p$, and $C_i = \mathbf{g}^{(\mathbf{m}_i, t_i)}$.
  2. Each $\mathcal{P}_i$, $i \in [n]$ broadcasts σ'_i, C_i.
  3. Each $\mathcal{P}_i$, $i \in [n]$ computes $(z_1, \ldots, z_n)$ where $z_j = e(\sigma_{j,2}, \tilde{g}) \cdot e(\sigma_{j,2}, \tilde{X})^{-1} \cdot e(\sigma_{j,1}, C_j)^{-1}$.
  4. Each P_i, $i \in [n]$ computes challenge $\gamma \leftarrow_R \mathbb{Z}_p$ by querying the Random Oracle RO using the message $(C_1 || \sigma'_1 || \cdots || C_n || \sigma'_n)$. Subsequently P_i computes $\mathbf{y}_i = \sum_{j \in [n]} \gamma^j (\mathbf{m}_{ij}, t_{ij})$, $\mathcal{W}_i = \sum_{j \in [n]} v_{ij} \gamma^j$ where $\mathbf{m}_{ij}, t_{ij}$ and v_{ij} denote $\mathcal{P}_i$'s share of $\mathcal{P}_j$'s inputs $\mathbf{m}_j, t_j$ and v_j respectively.
  5. All parties compute $C = \prod_{j \in [n]} C_j^{\gamma^j}$, $z = \sum_{j \in [n]} z_j \gamma^j$ Parties hold shares $\mathbf{y}_i, \mathcal{W}_i$ of $\mathbf{y}, w$ satisfying
  $$\mathbf{g}^{\mathbf{y}} = C \text{ and } h^w = z$$
  6. Parties run the interactive phase of the protocol $\Pi_{\mathsf{cd\text{-}pok}}$ on statement C and protocol $\Pi_{\mathsf{d\text{-}pok}}$ on statement z with $\mathbf{g}$ and h as the respective generators.
- **Output**: P_j outputs $b_j = 1$ if both protocols $\Pi_{\mathsf{cd\text{-}pok}}$ and $\Pi_{\mathsf{d\text{-}pok}}$ accept.

It is again straightforward to see that we can achieve a publicly verifiable two-round version of this optimized protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps\text{-}auth\text{-}opt}}$, that achieves soundness and zero-knowledge.

B.5 Distributed PoK of PS Signatures with Robust Completeness

In this section, we build upon Π_{rob} to propose a distributed proof of knowledge achieving robust completeness for PS signature over an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where a set of distributed provers $(\mathcal{W}_1, \ldots, \mathcal{W}_n)$ with access to the shares of a message vector $\mathbf{m}$, along with a signature σ on $\mathbf{m}$, proves knowledge of σ. The protocol is called $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt\text{-}rob}}$, and is conceptually similar to its variant $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$ with non-robust completeness described in Section B.3 (including similar optimizations for efficiency improvement), but additionally achieves robust completeness by using Π_{rob} as its base protocol.

Protocol $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt\text{-}rob}}$

- **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 $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell$ 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 $t \leftarrow_R \mathbb{Z}_p$, 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 $r, v \leftarrow_R \mathbb{Z}_p$ 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 NI-ZKPoK 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 ZKPoK protocol Π_{rob} for the relation $\mathbf{g}^{\mathbf{s}} = C$
  6. $\mathcal{V}$ accepts if π is valid and Π_{rob} accepts.

Finally, we can again construct a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps\text{-}dpok\text{-}opt\text{-}rob}}$, that achieves soundness and zero-knowledge by relying on the Fiat-Shamir heuristic and using a random oracle.