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 trusted 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.

Input Authenticity. A malicious party can potentially modify its input such that all other parties recieve incorrect output, but it can locally "undo" the modification and learn the correct result. Additionally, a modified input can reveal more information about other parties' inputs, beyond what would be available if the function was computed on truthful inputs. 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 authority such as FDA. Real-world applications of MPC require that the inputs used for computing the function are *authentic*.

One way to achieve this authenticity is to run the MPC protocol on inputs that are signed by some trusted 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.

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 $w = \mathsf{Reconstruct}(w_1, \ldots, w_n)$.

Note that, as stated, this is a special case of MPC and any generic MPC protocol can be used to achieve this. However, we impose the following restrictions: (i) the provers cannot communicate with each other, and (ii) the verifier communicates only via a broadcast channel and is public coin. These two restrictions make the task non-trivial. 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 5, 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 2) used in constructions of efficient zkSNARKs like [AHIV17,BCR$^+$19]. Informally, the results state that to check that a set of k sharings of messages $s_1, \ldots, s_k$ have not been tampered (by corrupt provers), it is sufficient to publicly reveal a suitably blinded linear combination of the above sharings. The deviant positions in the revealed sharing (from a consistent sharing) with overwhelming probability capture deviations across all the sharings. We leave applicability of these techniques to other relations as an interesting future work.

Authenticated MPC. We consider a notion of input authenticity where the inputs possess a valid signature from a trusted entity. 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 $i \in [n]$.

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

	Succint Signature	Succinct Communication	Multi-Auth Efficiency	Robustness
BJ18 [BJ18]	No	No	No	No
ADEO21 [ADEO21]	Yes	Yes	Yes**	No
$\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ (Sec. 4.2)	Yes	Yes	Yes	No
$\Pi_{\mathsf{ps\text{-}auth\text{-}opt}}$ (Appendix A.4)	Yes	Yes	Yes*	No
$\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}$ (Sec. 5)	Yes	Yes	No	Yes

Table 1: Comparison of features embodied by authenticated MPC protocols. Succinctness refers to signature size and communication being sublinear with respect to message size and unauthenticated communication respectively. Multi-Auth efficiency implies protocol is efficient for authenticating inputs from several parties, where $*$ denotes that the property holds when the signatures come from the same issuer and $**$ denotes that only communication overhead is efficient, not the computational overhead.

– 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 7, we propose a generic compiler that transforms a protocol Π from a class of secret-sharing based protocols to an *authenticated* protocol Π'. The class of protocols we consider are malicious protocols based on Shamir secret sharing (it generalizes to any linear secret sharing scheme). For authentication, our techniques allow signature schemes that are algebraically compatible: these include Camenisch-Lysyanskaya (CL) signatures [CL01], Boneh-Boyen-Shaham (BBS) signatures [BBS04], 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 A. 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 A.3 and Table 3.

A more detailed overview of our technical ideas is in Section 1.3, and Figure 1 highlights key components and the roadmap for realizing the authenticated MPC functionality described in this paper.

Implementation. We implement our protocol to illustrate the practical viability of our approach. In Section 8, 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 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*. If the underlying MPC protocol uses broadcast, then this is not an additional assumption. Otherwise, broadcast will have to be implemented using point-to-point channels and cryptography. 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 8.

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] con-

[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.

structs 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 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 in [WZC$^+$18] considers a distributed prover in order to improve prover efficiency, but the witness is still held by one entity. In Feta [BJO$^+$22], the distributed notion is a generalization of designated verifier to the threshold setting where a set of verifiers jointly verify the correctness of the proof. Prio [CB17] proposes secret shared non-interactive proofs where again, there is a single prover and many verifiers.

Our formulation of distributed proofs of knowledge also differs from recent works on distributed zkSNARKs [SVdV16,OB21,DPP$^+$22], where the focus is on jointly computing a non-interactive publicly verifiable proof (with specific focus on Groth16 [Gro16], Plonk [GWC19] and Marlin [CHM$^+$20]). Their constructions require additional interaction among the workers over private channels; on the other hand, we consider distributed proofs of knowledge where all interaction with the verifier takes place over a public broadcast channel. We also study the notion of *robust completeness* that guarantees that the protocol runs to completion even in the presence of malicious behavior, which was not considered in prior works.

Fully Linear PCPs. 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 provide a comparison of our work with FLPCP [BBC$^+$19] in terms of our definition, applications, and efficiency of our constructions.

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,

	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.

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 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 $g^{\Sigma z_i} = g^{\Sigma(\alpha_i + cw_i)} = \prod_i a_i x^c$.

This idea generalizes to any linear secret sharing scheme, and also extends to other relations. For instance, to prove knowledge of representation of a vector of discrete logarithms with respect to public generators. In our final construction we use additional ideas like randomization of the first message of each $\mathcal{P}_i$ via a sharing of 0 in order to ensure zero-knowledge.

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.

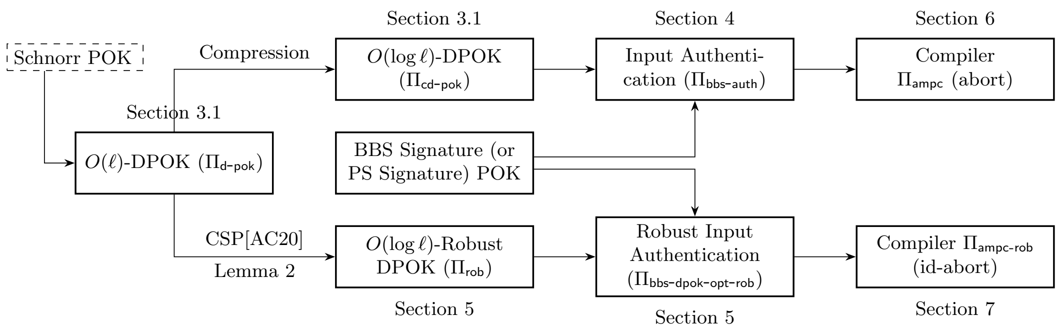

Fig. 1: Overview of the main components of our construction. The arrows denote dependence among the components. Acronyms POK, DPOK stand for proof-of-knowledge and distributed-proof-of-knowledge respectively. We also indicate the tools used to realize specific components.

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 2) 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 A), which leads to vastly more efficient distributed protocols. Figure 1 highlights key components and the roadmap for realizing the authenticated MPC functionality described in this paper.

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. [6]

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

In this section, we recall the formal definition of threshold secret sharing.

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: For $\mathbf{s} \in \mathbb{F}^m$, $\mathsf{Share}(\mathbf{s})$ samples $m \times n$ matrix $\mathbf{S}$, where j^{th} row of $\mathbf{S}$ is obtained as output of $\mathsf{Share}(\mathbf{s}[\,j\,])$ (Here $\mathbf{s}[\,j\,]$ denotes j^{th} component of $\mathbf{s}$). Subsequently it outputs $(\mathbf{s}_1, \ldots, \mathbf{s}_n)$ as shares of $\mathbf{s}$, where $\mathbf{s}_i$ denotes the i^{th} column of $\mathbf{S}$. Similarly, given a set $\mathcal{I} \subseteq [n]$ and vectors $(\mathbf{s}_1, \ldots, \mathbf{s}_{|\mathcal{I}|})$, the $\mathsf{Reconstruct}$ algorithm first constructs a matrix $\mathbf{S}$ of size $m \times |\mathcal{I}|$ with vector $\mathbf{s}_i$ as its i^{th} column. Subsequently $\mathsf{Reconstruct}$ outputs a vector $\mathbf{s} \in \mathbb{F}^m$ by reconstructing each row of $\mathbf{S}$ individually. We record the following useful fact regarding Shamir Secret Sharing.

Lemma 1. *Let $(\mathsf{Share}, \mathsf{Reconstruct})$ constitute a (t, n) Shamir Secret Sharing over finite field $\mathbb{F}$. There exist vectors $\mathbf{t}_j \in \mathbb{F}^t$ for $j \in [1, (n-t+1)]$ such that:*

- **Scalar Version:** *For any $s \in \mathbb{F}$ and $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$, we have $s_{t-1+j} = \langle (s, s_1, \ldots, s_{t-1}), \mathbf{t}_j \rangle$.*
- **Vector Version:** *For any $\mathbf{s} \in \mathbb{F}^m$, $m \geq 2$ and $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \leftarrow_R \mathsf{Share}(\mathbf{s})$, we have $\mathbf{s}_{t-1+j} = \mathbf{S}_t \mathbf{t}_j$, where $\mathbf{S}_t = [\mathbf{s}, \mathbf{s}_1, \ldots, \mathbf{s}_{t-1}]$ denotes the $m \times t$ matrix with $\mathbf{s}, \mathbf{s}_1, \ldots, \mathbf{s}_{t-1}$ as its columns.*

Proof. The vectors $\mathbf{t}_j$ correspond to coefficients of Largangian Interpolation formula for interpolating the value of a $\leq t-1$ degree polynomial at η_{t-1+j} in terms of its values at $0, \eta_1, \ldots, \eta_{t-1}$.

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 $\mathbf{Q} \in \mathcal{L}^m$.

[6] Note that 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.

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 5. It has been previously used in construction of zero knowledge proofs in the interactive oracle setting (e.g [AHIV17,BCR+19]), to check that the oracle represents "low degree polynomials".

Lemma 2. *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], along with the associated proof of knowledge [CDL16].

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

Definition 4 (BBS Signature Scheme [BBS04]). *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* $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$*.*
– $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: 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 $\mathsf{NI\text{-}CSP}$.

3 Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge 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 interact with each other, and all the interaction with the verifier takes place over a public broadcast channel. These restrictions imply that the notion does not trivially follow from general multiparty computation.

Definition 5 (Distributed Proof of Knowledge). *Let $\mathcal{R} = \{\mathcal{R}_\kappa\}_{\kappa\in\mathbb{N}}$ be a family of relations. An n-worker distributed proof of knowledge for $\mathcal{R}$ consists of interactive adversaries $\mathcal{P}, W_1, \ldots, W_n$ and $\mathcal{V}$ where $\mathcal{P}$ is called the* Prover*, $\mathcal{V}$ is called the* Verifier *and $\mathcal{W} = \{W_1, \ldots, W_n\}$ are called the* workers. *Additionally we have algorithms* Setup, Share, *and* $\mathsf{Reconstruct}$ *where* Setup *generates relation specific parameters while* $(\mathsf{Share}, \mathsf{Reconstruct})$ *constitute a (t, n) secret sharing scheme. The distributed proof of knowledge $\Pi = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Reconstruct})_{\mathcal{P},\mathcal{W},\mathcal{V}}$ for $\mathcal{R}$ is described as below:*

- **Setup:** *The algorithm* Setup *takes the relation description and outputs public parameters as* $\mathsf{Setup}(R_\kappa) \to \mathsf{pp}$.
- **$\mathcal{P}$'s Input:** *Instance $\mathbf{x}$ and witness $\mathbf{s}$ such that $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}_\kappa$.*
- **W_i's Input:** $(\mathbf{x}, \mathbf{s}_i)$*, where $\mathbf{s}_i$ is the i^{th} share of the message vector $\mathbf{s}$, s.t.* $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = \mathbf{s}$.
- **Pre-processing:** *$\mathcal{P}$ sends aux_i privately to W_i for all $i \in [n]$.*
- **Interactive Protocol:** *The workers $W_1, \ldots, W_n$ interact with the verifier $\mathcal{V}$ over a public broadcast channel over k rounds for some $k \in \mathbb{N}$. In round j, each worker W_i broadcasts a message m_{ij} depending on its randomness and messages received in prior $(j-1)$ rounds (including the inputs $\mathbf{s}_i, \mathsf{aux}_i$). Similarly, in each round $j \in [k]$, $\mathcal{V}$ broadcasts a uniformly sampled challenge $\mathbf{c}_j$ from appropriate domain.*
- **Output:** *At the end of k rounds, $\mathcal{V}$ outputs* 1 *(Accept) or* 0 *(Reject).*

Let $\Pi = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Reconstruct})_{\mathcal{P},\mathcal{V},\mathcal{W}}$ be a distributed proof of knowledge for relation family $\mathcal{R}$. For an adversary $\mathcal{A}$ controlling a subset of parties in the protocol Π, we use $\Pi(\mathcal{A}, \mathbf{x})$ to denote the output of the protocol for statement $\mathbf{x}$, while we use $\langle \Pi(\mathcal{A}, \mathbf{x}) \rangle$ to denote the transcript of the interactive protocol consisting of all messages broadcast by *honest* parties. Note that, due to the restrictions imposed on the interaction to be over broadcast, the view of $\mathcal{A}$ in Π is precisely the transcript $\langle \Pi(\mathcal{A}, \mathbf{x}) \rangle$ together with it's own randomness. We define the following properties for Π:

- **Completeness**: For all $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}_\kappa$, when all parties follow the protocol (i.e $\mathcal{A} = \varnothing$), we have $\Pi(\mathcal{A}, \mathbf{x}) = 1$ with probability 1.
- **Soundness**: For any PPT adversary $\mathcal{A}$ controlling the parties $\mathcal{W} \cup \{\mathcal{P}\}$, there exists an efficient extractor $\mathcal{E}$ with rewinding access to $\mathcal{A}$ such that whenever $\Pi(\mathcal{A}, \mathbf{x}) = 1$, with overwhelming probability $\mathcal{E}^{\mathcal{A}}(\mathbf{x})$ outputs $\mathbf{s}$ such that $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}_\kappa$.
- **Zero Knowledge**: For any adversary $\mathcal{A}$ controlling at most t parties in $\mathcal{W}$, there exists a simulator Sim which given $\mathbf{x}$ and inputs $\{(\mathbf{s}_i, \mathsf{aux}_i)\}$ for all parties i controlled by $\mathcal{A}$ outputs a transcript indistinguishable from $\langle \Pi(\mathcal{A}, \mathbf{x}) \rangle$.
- **Robust Completeness**: We define a stronger notion of completeness that is *robust* to the presence of some corupt parties. We say that the protocol satisfies robust completeness with threshold $\ell > 0$ if for any adversary controlling at most ℓ parties in $\mathcal{W}$ it holds that: for $(\mathbf{x}, \mathbf{s}) \in \mathcal{R}_\kappa$, and an honest prover $\mathcal{P}$ using $\mathbf{s}$ as witness, with overwhelming probability $\Pi(\mathcal{A}, \mathbf{x}) = 1$ while identifying the adversarial workers. Intuitively, the protocol protects an honest $\mathcal{P}$ from upto ℓ corrupt workers.
- **Succinctness**: We say that the protocol Π is succinct if the total length of all the messages broadcast by a party

We assume an honest verifier $\mathcal{V}$ for ease of exposition. However, our eventual goal is to have a publicly verifiable transcript as detailed in Section 3.1.

Remark. The notion of distributed interactive proofs has appeared in [Ped91], in the context of relations describing the verification of undeniable signature schemes. Looking ahead, we consider distributed proofs of knowledge for the discrete log relation, which when combined with appropriate signature scheme (e.g., BBS [BBS04] or PS [PS16]), proves knowledge of signature, where the signed message is shared among the workers. In particular, both the message and the signature remain private, unlike in [Ped91], where the signature is public.

3.1 Distributed Proof of Knowledge for Discrete Log

In this section, we provide a construction of distributed proof of knowledge for the discrete log relation.

Definition 6 (Discrete Log Relation). *Let $\mathcal{R}_\kappa^{\mathrm{DL}}$ be a relation specified by $(p_\kappa, \mathbb{G}_\kappa, \mathbf{g}_\kappa)$ where p_κ is a prime integer, $\mathbb{G}_\kappa$ is a group of order p_κ and $\mathbf{g}_\kappa = (g_1, \ldots, g_{m_\kappa})$ is a vector of generators from $\mathbb{G}_\kappa$. Further we have $\log(p_\kappa) = O(\kappa)$ and $m_\kappa = \mathrm{poly}(\kappa)$. The relation $\mathcal{R}_\kappa^{\mathrm{DL}}$ consists of pairs $(z, \mathbf{s})$ with $z \in \mathbb{G}_\kappa$ and $\mathbf{s} \in \mathbb{Z}_{p_\kappa}^{m_\kappa}$ satisfying $z = \mathbf{g}_\kappa^{\mathbf{s}}$. Let $\mathcal{R}^{\mathrm{DL}} = \left\{\mathcal{R}_\kappa^{\mathrm{DL}}\right\}_{\kappa\in\mathbb{N}}$ be a family of* discrete log *relations. Additionally, we assume that any efficient adversary outputs a non-trivial discrete log relationship among $\mathbf{g}_\kappa$ with probability negligible in κ.*

For notational convenience, we will drop the subscript κ while specifying relations from $\mathcal{R}^{\mathrm{DL}}$ and instead consider $\mathcal{R} \in \mathcal{R}^{\mathrm{DL}}$ to consist of pairs $(z, \mathbf{s})$ satisfying $z = \mathbf{g}^{\mathbf{s}}$ where $\mathbf{s} \in \mathbb{Z}_p^m$, $\mathbf{g} \in \mathbb{G}^m$ for group $\mathbb{G}$ of prime order p, and suitable m.

Basic Distributed Protocol We first present a distributed protocol for $\mathcal{R}^{\mathrm{DL}}$, namely $\Pi_{\mathsf{d\text{-}pok}}$, that achieves soundness, completeness and zero-knowledge. We assume $(\mathsf{Share}, \mathsf{Reconstruct})$ constitute a (t, n) Shamir Secret Sharing over $\mathbb{Z}_p$, where $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = k_1\mathbf{s}_1 + \cdots + k_n\mathbf{s}_n$ for $k_i \in \mathbb{Z}_p$. 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 is required to ensure that the distribution of individual workers' messages is not dependent on their respective shares. Note that in protocol $\Pi_{\mathsf{d\text{-}pok}}$ (and in all other protocols described subsequently), we use the term "pre-processing" to denote any steps that either involve purely local computation for the prover, or steps that require the usage of point-to-point channels, and the term "interactive protocol" to denote steps that involve broadcast channels only.

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

- **Public Parameters**: $(p, \mathbb{G}, \mathbf{g}, h)$ where $\mathbf{g} \in \mathbb{G}^\ell$, $h \in \mathbb{G}$.
- **$\mathcal{P}$'s inputs**: $\mathbf{s}$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$.
- **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 $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = \mathbf{s}$.
- **Preprocessing :** $\mathcal{P}$ samples a random additive sharing $(\rho_1, \ldots, \rho_n)$ of 0 satisfying $\rho_1 + \cdots + \rho_n = 0$. Thereafter, $\mathcal{P}$ sends ρ_i to W_i.
- **Interactive Protocol**:
  1. 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. W_i $(i \in [n])$ computes $\mathbf{x}_i = c\mathbf{s}_i + \mathbf{t}_i$ and broadcasts $\mathbf{x}_i$.
- **Output**: $\mathcal{V}$ 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 $\Pi_{\mathsf{d\text{-}pok}}$ achieves perfect completeness, 2-special soundness, special honest-verifier zero-knowledge, and a communication complexity of ℓ elements of $\mathbb{Z}_p$ and 1 element of $\mathbb{G}$ from each worker to the verifier.*

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.

2-Special Soundness. Consider 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', $c \neq c'$. Then we have:

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

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[7].

[7] We note that, here if we have that a component of $\mathbf{s}_i$ as 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.

Special Honest-Verifier Zero Knowledge. We construct a simulator Sim that produces a transcript indistinguishable from the transcript of the protocol given a challenge $c \in \mathbb{Z}_p$. WLOG, we assume $n \notin \mathcal{C} \subsetneq [n]$. Sim receives $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$ and $\alpha_i \leftarrow_R \mathbb{G}$ from $\mathcal{A}$, for all $i \in \mathcal{C}$, samples $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^\ell$, $\forall$ $i \in [n], i \notin \mathcal{C}$, samples $\alpha_i \leftarrow_R \mathbb{G}$, $\forall$ $i \in [n-1], i \notin \mathcal{C}$, 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. The workers perform $O(\ell)$ operations over $\mathbb{G}$ and $\mathbb{Z}_p$ (total complexity of $O(\ell\|\mathbb{G}\| + \ell\|\mathbb{Z}_p\|)$). The verifier performs ℓ exponentiations over $\mathbb{G}$ and ℓn operations over $\mathbb{Z}_p$ (total complexity of $O(\ell n\|\mathbb{Z}_p\| + \ell\|\mathbb{G}\|)$).

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 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 W_i computes $c = \mathsf{RO}\left(\alpha_1\|\cdots\|\alpha_n\right)$ 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 $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ as described achieves perfect completeness, soundness and zero-knowledge in the random oracle model.*

Proof. Perfect completeness of $\Pi^{\mathsf{pv}}_{\mathsf{d\text{-}pok}}$ follows from perfect completeness of $\Pi_{\mathsf{d\text{-}pok}}$. For 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 $\{W_i : i \in \mathcal{C}\}$. Note that we fork the adversary $\mathcal{A}$ on behalf of all the workers $W_i, i \in \mathcal{C}$, 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 accepting transcript. The extraction of the witness thereafter follows from the 2-special 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, which only has knowledge of the relation (this simulator is essentially identical to the simulator for $\Pi_{\mathsf{d\text{-}pok}}$ except for Steps-1,7 and 8, which are additionally introduced):

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 receives $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^m$ and $\alpha_i \leftarrow_R \mathbb{G}$ from $\mathcal{A}$, for all $i \in \mathcal{C}$. Sim also samples $c \leftarrow_R \mathbb{Z}_p$ and $\mathbf{x}_i \leftarrow_R \mathbb{Z}_p^m$, $\forall$ $i \in [n], i \notin \mathcal{C}$. Finally, it samples $\alpha_i \leftarrow_R \mathbb{G}$, $\forall$ $i \in [n-1], i \notin \mathcal{C}$, 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 $(\alpha_1\|\cdots\|\alpha_n)$. Otherwise, Sim programs $\mathsf{RO}(\alpha_1\|\cdots\|\alpha_n) := c$ 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**: $(p, \mathbb{G}, \mathbf{g})$ where $\mathbf{g} \in \mathbb{G}^\ell$.
- **$\mathcal{P}$'s inputs**: $\mathbf{s}$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$.
- **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 Protocol**:
  1. 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. 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$(\mathbf{g}') = 2$: 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**: $\mathcal{V}$ 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 special soundness.*

Proof. The protocol $\Pi_{\mathsf{d\text{-}csp}}$ has perfect completeness, 3-special soundness. Each worker broadcasts two elements of $\mathbb{Z}_p$ (in the final round), and a total of $2\log(\ell)$ elements of $\mathbb{G}$ across all the rounds. Thus the communication incurred by each worker is $2\log(\ell)\log(|\mathbb{G}|) + 2\log(|\mathbb{Z}_p|)$. We now prove the properties satisfied by the protocol.

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 $\mathbf{g}'^{\mathbf{s}'} = z'$.

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

Special Soundness. We prove 3-special 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 consider 3 accepting transcripts

$$(\{A_i, B_i\}_{i \in [n]}, c_1, \{s_i'^1\}_{i \in [n]}), (\{A_i, B_i\}_{i \in [n]}, c_2, \{s_i'^2\}_{i \in [n]}),$$

$$(\{A_i, B_i\}_{i \in [n]}, c_3, \{s_i'^3\}_{i \in [n]}).$$

Let a_1, a_2 and a_3 be such that $a_1 + a_2 + a_3 = 0$, $c_1 a_1 + c_2 a_2 + c_3 a_3 = 1$ and $c_1^2 a_1 + c_2^2 a_2 + c_3^2 a_3 = 0$. 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 $\mathbf{s}_i = \sum_j a_j(c_j \mathbf{s}'^j_i, \mathbf{s}'^j_i)$ and consider $\mathbf{s} = k_1\mathbf{s}_1 + \cdots + k_n\mathbf{s}_n$. For $j \in [3]$, we have:

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

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

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

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

Efficiency. The workers perform $O(\ell)$ operations over $\mathbb{G}$ (exponentiations) and $\mathbb{Z}_p$, giving efficiency of $O(\ell\|\mathbb{G}\| + \ell\|\mathbb{Z}_p\|)$. The verifier performs $O(\ell)$ exponentiations over $\mathbb{G}$ and $O(n)$ operations over $\mathbb{Z}_p$ giving efficiency of $O(n\|\mathbb{Z}_p\| + \ell\|\mathbb{G}\|)$. We can further improve the concrete efficiency of verifier by replacing the $O(\ell)$ exponentiations over $\mathbb{G}$ by a single multiexponentiation of size $\ell + 2\log(\ell)$, by directly computing the generators for the final round in terms of the initial generators (see Section 6.2 of [BBB+18]). Each worker broadcasts two elements of $\mathbb{Z}_p$ (in the final round), and a total of $2\log(\ell)$ elements of $\mathbb{G}$ across all the rounds. Thus the communication incurred by each worker is $2\log(\ell)\log(|\mathbb{G}|) + 2\log(|\mathbb{Z}_p|)$.

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 $\Pi_{\mathsf{d\text{-}pok}}$ (3.1), 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.

4 Distributed PoK for BBS Signatures

The ZKPoK for BBS signatures outlined in Section **??** assumes a *single* prover holding a valid BBS 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 BBS signature on the message, can together prove knowledge of the (message, signature) pair with respect to a public verification key. The straightforward distributed proof PoK protocol for BBS signature (protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$ in Section 4.1, which is a simple adaptation of the BBS PoK from [CDL16]) does not lend itself to aggregation of many proofs since the statement is not with respect to a public set of generators; in particular, there is a witness v with respect to d where d is sent by the prover as part of randomized signature. In order to allow for aggregation, we formulate proving knowledge of a BBS signature as two different statements: a proof of knowledge over the generators available from the public key pk which can be aggregated, and another proof of opening with respect to the d's that are given by each prover. The latter proof is constant-sized, and we compress the former aggregated statement. The initial straightforward distributed proof PoK protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$ for BBS signature is shown below, followed by the improved protocol $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$.

4.1 Distributed PoK 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 $(W_1, \ldots, W_n)$ 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 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 $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
- **W_i's inputs** : 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 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 W_i, and $\mathcal{V}$.
  2. Each 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 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 4. *Assuming that the discrete log assumption holds over the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the proposed protocol $\Pi_{\mathsf{bbs\text{-}dpok}}$ as described above achieves perfect completeness, $(2, k_1, \ldots, k_{(\log_2(\ell+1)-1)})$-special soundness with $k_i = 3$ for all $i = [\log_2(\ell+1) - 1]$, and special honest-verifier zero-knowledge.*

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.

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

- **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 $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
- **W_i's inputs** : 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 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 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 $d = b^u \cdot h_0^{-r}$. $\mathcal{P}$ 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 W_i, and $\mathcal{V}$.
  2. The workers 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 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 $C \cdot D = g_1^{-1}$, $e(A', w) = e\left(\bar{A}, g_2\right)$, $\Pi_{\mathsf{cd\text{-}pok}}$ 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.

4.2 A Multi-Verifier Extension of the BBS PoK Authenticating All Inputs

In this section, we describe an extension of the original BBS PoK protocol where multiple parties issue proofs of signatures on their own (private input) messages, and each party verifies the proof issued by all other parties, subject to the restriction that all signatures are verified against a common public key pk. Concretely, in the distributed setting, 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 3.1). The protocol $\Pi_{\mathsf{bbs\text{-}auth}}$ is described in Figure 4.2. We use this protocol is a building block for our eventual compiler to achieve authenticated MPC.

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

- **Public Key** $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$.
- 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 $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ 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 $O(n^2 \log(\ell))$ corresponding to n invocations of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. The computational effort of a party is similarly $O(\ell + n \log \ell)$ exponentiations and $O(n)$ pairings.

Achieving Public Verifiability. Finally, it is straightforward to see that we can achieve a publicly verifiable two-round version of this protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ that achieves soundness and zero-knowledge, by running n instances of the publicly verifiable protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ outlined earlier as opposed to $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. The resulting construction and proof techniques are very similar to that used for our other publicly verifiable protocols, and hence we omit the details.

We now present an optimized variant of the above protocol $\Pi_{\mathsf{bbs\text{-}auth}}$, which we call $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ (the corresponding publicly verifiable two-round version is called $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$). This optimized protocol reduces the overheads of $\Pi_{\mathsf{bbs\text{-}auth}}$ further to achieve $O(n \log(\ell))$ computational complexity and a computational overhead of $O(\ell n)$ exponentiations and $O(1)$ pairings per party. This is enabled by using carefully designed optimizations that combine n instances of the sub-protocol $\Pi_{\mathsf{cd\text{-}pok}}$ (one corresponding to each instance of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$) into a single instance of $\Pi_{\mathsf{cd\text{-}pok}}$, using a random challenge. Concretely, suppose that the j-th of the sub-protocol $\Pi_{\mathsf{cd\text{-}pok}}$ allows party P_j to prove that $\mathbf{h}^{\mathbf{m}_j} \cdot h_0^{\eta_j} = D_j$ where $\mathbf{m}_j$ is $\mathcal{P}_j'$s private input, η_j is commitment randomness, and D_j is the commitment broadcast by $\mathcal{P}_j$ in Step 2 of $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$. 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 $\mathbf{h}^{\mathbf{s}} \cdot h^\eta = \prod_j D_j^{\gamma_j}$. The parties can compute shares of satisfying $\mathbf{s} = \sum_j \gamma^j \mathbf{m}_j$ and $\eta = \sum_j \gamma^j \eta_j$ using their shares of $\mathbf{m}_j, \eta_j$ for $j \in [n]$. The detailed protocol is presented below.

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

- **Public Key** $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$.
- 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 $D = \prod_{j\in[n]} D_j^{\gamma^j}$.

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

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.

We refer to the publicly verifiable two-round version of this protocol as $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$.

5 Robust Complete DPoK

While the protocols of Section 3 ensured privacy against a malicious adversary controlling up to t parties, the completeness was guaranteed only if all the workers follow the protocol. This is sometimes undesirable as the shares of the honest parties are sufficient to determine the secret, and so an honest prover should expect to be able to "ride over" a few deviating workers – a property that we call *robust* completeness.

The technical difficulty in achieving robust completeness for our proposed proof of knowledge arises from the fact that several messages contain the shares "in the exponent", which makes it harder to distinguish messages issued by corrupt parties from those issued by the honest ones. In this section, we build upon the protocols presented so far to simultaneously achieve both succinctness and robust completeness while withstanding corruptions of up to $\ell < (n-t)/3$ workers. This is achieved by forcing the workers to commit to their shares and then forcing them to reveal certain linear form over their share. The revealed linear form over all the shares allows us to identify and discard corrupt messages. Robust completeness also ensures that input authentication does not abort when the protocol is used as part of a larger multiparty computation, i.e. if the remainder of the protocol has resilience against malicious behavior, input authentication preserves it.

Robust Complete DPoK for Discrete Log. We first describe the key ideas of our modified protocol for achieving robust completeness in addition to succinctness:

– **Pre-processing:** The prover distributes shares r_i of a random element $r \in \mathbb{Z}_p$ amongst the workers, which will be used to blind the linear form over the shares that the workers will later reveal.
– **Workers Commit to Shares:** In the interactive phase, the workers first commit to their respective shares by sendng $A_i = \mathbf{g}^{\mathbf{s}_i}$ and $B_i = h_1^{r_i} h_2^{\omega_i}$ for uniformly sampled ω_i. Here h_1 and h_2 are additional generators of $\mathbb{G}$.
– **Reveal Linear Form over Shares:** The verifier sends a challenge vector $\boldsymbol{\gamma} \in \mathbb{Z}_p^\ell$, and the workers reveal the linear form $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$. Here r_i is used to blind the contribution of the vector $\mathbf{s}_i$. Observe that the honest tuple $(v_1, \dots, v_n)$ forms a valid secret sharing of $v = \langle \boldsymbol{\gamma}, \mathbf{s} \rangle + r$. Using results from "low degree testing" (Lemma 2), we show that if i^{th} party deviates from the protocol, with overwhelming probability, the received tuple $(v_1', \dots, v_n')$ differs from the honest tuple at position i. This allows us to identify corrupt parties.
– **Determine Honest Commitments:** Using error correction, we identify the set $\mathcal{H} = \{i_1, \dots, i_q\}$of honest parties. Let $k_1', \dots, k_q'$ denote the reconstruction coefficients corresponding to this set $\mathcal{H}$.
– **Output:** $\mathcal{V}$ outputs 1 if $\prod_{j\in[q]} A_{i_j}^{k_j'} = z$, and 0 otherwise.

The detailed protocol, called Π_{rob} is presented below. We note that standard (non-robust) completeness is straightforward to verify, while succinctness follows immediately from the use of NI-CSP in Step 4.

Protocol Π_{rob}

- **Public Parameters**: $(p, \mathbb{G}, \mathbf{g}, h_1, h_2)$ where $\mathbf{g} \in \mathbb{G}^\ell$ and $h_i \in \mathbb{G}$ for $i = 1, 2$.
- **$\mathcal{P}$'s inputs**: $\mathbf{s} \in \mathbb{Z}_p^\ell$ such that $(z, \mathbf{s}) \in \mathcal{R}^{\mathrm{DL}}$
- **W_i's inputs :** W_i possesses the i^{th} share $\mathbf{s}_i$ of the message vector $\mathbf{s}$, such that $\mathsf{Reconstruct}(\mathbf{s}_1, \ldots, \mathbf{s}_n) = \mathbf{s}$
- **Pre-processing** : $\mathcal{P}$ samples $r \leftarrow_R \mathbb{Z}_p$, computes $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$. $\mathcal{P}$ sends the shares r_i to W_i, for all $i \in [n]$.
- **Interactive Protocol**:
  1. 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 the tuple (A_i, B_i).
  2. $\mathcal{V}$ broadcasts $\boldsymbol{\gamma} \leftarrow_R \mathbb{Z}_p^\ell$.
  3. W_i broadcasts $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$.
  4. W_i broadcasts following NI-ZKPoKs: (i) $\mathsf{NI\text{-}CSP}$ π_{i1} for showing knowledge of opening for A_i, (ii) $\mathsf{NI\text{-}CSP}$ π_{i2} showing knowledge of opening for B_i and (iii) $\mathsf{NI\text{-}CSP}$ π_{i3} showing knowledge of opening $\mathbf{w}_i$ for $A_i \cdot B_i$ over generators $(\mathbf{g}, h_1, h_2)$ which satisfies the linear form $v_i = \langle \mathbf{w}_i, (\boldsymbol{\gamma}, 1, 0) \rangle$. An honest worker uses $\mathbf{w}_i = (\mathbf{s}_i, r_i, \omega_i)$ as its private input in the above protocol.
- **Output**: $\mathcal{V}$ performs following checks:
  - Construct vector $\mathbf{v}' = (v_1', \ldots, v_n')$ of v-values broadcast by the workers.
  - Error correct $\mathbf{v}'$ to obtain $\mathbf{v} = (v_1, \ldots, v_n) \in \mathcal{RS}_{n,t,\boldsymbol{\eta}}$.
  - Output $(0, \{\mathcal{P}\})$ if $\Delta(\mathbf{v}, \mathbf{v}') \geq (n-t)/3$, else proceed to next steps.
  - Determine the set $E_1 \subseteq [n]$ of workers who provide an incorrect proof in Step (4).
  - Determine the set $E_2 \subseteq [n]$ of positions where the vectors $\mathbf{v}$ and $\mathbf{v}'$ differ.
  - Set $\mathcal{C} = E_1 \cup E_2$ and $\mathcal{H} = [n] \backslash \mathcal{C}$. Let $\mathcal{H} = \{i_1, \ldots, i_q\}$.
  - Compute reconstruction coefficients $k_1', \ldots, k_q'$ for the set $\mathcal{H}$.
  - Output $(1, \mathcal{C})$ if $\prod_{j \in [q]} A_{i_j}^{k_j'} = z$, and $(0, \{\mathcal{P}\})$ otherwise.

Theorem 5. *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, protocol Π_{rob} achieves robust-completeness, soundness, honest-verifier zero-knowledge, and a communication complexity of $O(\log \ell)$ elements of $\mathbb{G}$ from each worker to verifier.*

Proof. **Soundness.** To prove soundness, we describe an extractor $\mathcal{E}$ that extracts a valid witness with overwhelming probability, whenever the protocol accepts. Since each worker is a successful adversary with respect to the zero knowledge protocols in Step (4), $\mathcal{E}$ uses the extractors for the compressed sigma protocols to extract witnesses $\mathbf{s}_i$, r_i and ω_i such that $A_i = \mathbf{g}^{\mathbf{s}_i}$ and $B_i = h_1^{r_i} h_2^{\omega_i}$. For an accepting transcript, we also have indices $i_1, \ldots, i_q$ and reconstruction coefficients $k_1', \ldots, k_q'$ such that $\prod_{j=1}^q A_{i_j}^{k_j'} = z$ which implies $\mathbf{g}^{\mathbf{s}} = z$ for $\mathbf{s} = \sum_{j=1}^q k_j' \mathbf{s}_{i_j}$. The extractor thus outputs $\mathbf{s}$.

Zero Knowledge. Without loss of generality, we assume that the adversary $\mathcal{A}$ controls workers $W_1, \ldots, W_\lambda$ where $\lambda \leq t$. We describe a simulator Sim that takes the statement $\mathbf{x} = (\mathbf{g}, z)$ and inputs of corrupted parties $\{(\mathbf{s}_i, r_i)\}_{i=1}^\lambda$ as inputs, and produces a transcript indistinguishable from $\Pi(\mathcal{A}, \mathbf{x})$. Let $\mathcal{H} = \{\lambda + 1, \ldots, n\}$ denote the set of honest parties. Then,

$$\Pi(\mathcal{A}, \mathbf{x}) = \left(\boldsymbol{\gamma}, \{A_i, B_i, \pi_{i1}, \pi_{i2}, \pi_{i3}, v_i\}_{i \in \mathcal{H}}, b\right)$$

where π_{i1}, π_{i2} and π_{i3} denote the three zero knowledge arguments in Step (4) and $b \in \{0, 1\}$ denotes the output of the protocol. The operation of the simulator Sim is described below:

1. Sim receives $\{(\mathbf{s}_i, r_i)\}_{i=1}^\lambda$ as input.
2. Compute $A_i = \mathbf{g}^{\mathbf{s}_i}$ for $1 \leq i \leq \lambda$. Choose A_i, $\lambda < i \leq t-1$ uniformly from $\mathbb{G}$. Computes $A_{t+j} = \mathbf{a}^{\mathbf{t}_j}$ where $\mathbf{a} = (z, A_1, \ldots, A_{t-1})$. Here the vectors $\mathbf{t}_j$ are as guaranteed by Lemma 1.
3. Sample B_i, $\lambda + 1 \leq i \leq n$ uniformly and independently from $\mathbb{G}$.
4. Choose $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}^m$ uniformly.
5. Compute $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$ for $1 \leq i \leq \lambda$. Choose v_i uniformly and independently from $\mathbb{Z}_p$ for $i \in \{0\} \cup \{\lambda + 1, \ldots, t-1\}$. Compute $v_{t-1+j} = \langle (v_0, \ldots, v_{t-1}), \mathbf{t}_j \rangle$.
6. Invoke simulators for the NI-ZKPoKs to obtain $\pi_{i1} \leftarrow_R \mathsf{Sim}^{\mathrm{zk}}(\mathbf{g}, A_i)$, $\pi_{i2} \leftarrow_R \mathsf{Sim}^{\mathrm{zk}}(\mathbf{h}, B_i)$ and $\pi_{i3} \leftarrow_R \mathsf{Sim}^{\mathrm{zk}}((\mathbf{g}, \mathbf{h}, \boldsymbol{\gamma}), A_i \cdot B_i)$.
7. Set $b = (1, \{1, \ldots, \lambda\})$.

To prove indistinguishability, we employ a hybrid argument where $H_1 = \Pi(\mathcal{A}, \mathbf{x})$ denotes the transcript in the real protocol. We define H_2 to be the hybrid, where the proofs $\{\pi_{i1}, \pi_{i2}, \pi_{i3}\}_{i=\lambda+1}^{n}$ are replaced by the simulated proofs, as computed by the simulator. Finally we obtain H_3 by generating $\{A_i, B_i, v_i\}_{i=\lambda+1}^{n}, \boldsymbol{\gamma}, b$ according to the simulator. This makes H_3 identical to the simulator output.

The first two hybrids are indistinguishable due to the zero knowledge property of the respective sigma protocols. To show indistinguishability of H_2 and H_3 we need to show that $(\{A_i, B_i, v_i\}_{i=\lambda+1}^{n}, \boldsymbol{\gamma}, b)$ are distributed identically in the real protocol and simulator output, conditioned on $\{\mathbf{s}_i, r_i\}_{i=1}^{\lambda}$. In an honest prover's sharing of $\mathbf{s}$ as $(\mathbf{s}_1, \ldots, \mathbf{s}_n)$, $\mathbf{s}_i$ is distributed uniformly in $\mathbb{Z}_p^{\ell}$ conditioned on $\mathbf{s}, \mathbf{s}_1, \ldots, \mathbf{s}_\lambda$ for $i \in (\lambda, t)$. Therefore, $\mathbf{g}^{\mathbf{s}_i}$ for $i \in (\lambda, t)$ is distributed uniformly in $\mathbb{G}$ conditioned on $\mathbf{s}, \mathbf{s}_1, \ldots, \mathbf{s}_\lambda$, and thus the vector $\mathbf{a} = (\mathbf{g}^{\mathbf{s}}, \mathbf{g}^{\mathbf{s}_1}, \ldots, \mathbf{g}^{\mathbf{s}_{t-1}})$ is distributed identically in H_2 and H_3. Since, $A_{t-1+j} = \mathbf{a}^{\mathbf{t}_j}$ for $j \in [1, n-t+1]$ in both H_2 and H_3 we conclude that $(A_{\lambda+1}, \ldots, A_n)$ has identical distribution across both hybrids. Similar argument also shows that the vector $(v_{\lambda+1}, \ldots, v_n)$ is identically distributed across the two hybrids. Finally, $(B_{\lambda+1}, \ldots, B_n)$ are distributed independently and uniformly over $\mathbb{G}$ in both hybrids, due to blinding using ω_i. As we show in the proof of "robust completeness" below the real protocol outputs $(1, \{1, \ldots, \lambda\})$ (i.e it identifies the adversarial set) with overwhelming probability, and hence b output by Sim is distributed statistically close to that output by the real protocol. This completes the indistinguishability of the hybrids, and by transitivity, shows that $\Pi(\mathcal{A}, \mathbf{x}) \approx H_3$.

Robust Completeness. We show that when the prover is honest, and has a correct witness $\mathbf{s}$, the protocol accepts with overwhelming probability, identifying the corrupt workers. Again, let $\mathcal{A}$ be an adversary corrupting $\lambda < (n-t)/3$ workers. A corrupt worker can deviate from the protocol in two ways:

1. It supplies an incorrect proof in Step 4. We call this set of workers as I_1.
2. All its proofs in Step 4 are correct, but it uses incorrect inputs $(\mathbf{s}'_i, r'_i) \neq (\mathbf{s}_i, r_i)$ to compute the proofs. We call this set of workers as I_2.

We show that any worker in $I = I_1 \cup I_2$ is identified with overwhelming probability. Let E_1 and E_2 be the sets of corrupt workers as identified by the verifier in the protocol Π_{rob}. Then, we need to show that $I_1 \cup I_2 = E_1 \cup E_2$. It is clear that $E_1 \cup E_2 \subseteq I_1 \cup I_2$. This is so because honest workers cannot be identified as corrupt due to perfect completeness of zero knowledge arguments, and the fact that the bound $\ell < (n-t)/3 < (n-t+1)/2$ on the number of corruptions ensures that the decoded codeword $\mathbf{v}$ is the same as honestly computed codeword. Next, we show $I_1 \cup I_2 \subseteq E_1 \cup E_2$ with overwhelming probability. Let $\mathbf{S}$ denote the "ideal" matrix with $(\mathbf{s}_i, r_i)$ as its i^{th} column. Similarly, let $\mathbf{v} = (v_1, \ldots, v_n)$ denote the correct vector with $v_i = \langle \boldsymbol{\gamma}, \mathbf{s}_i \rangle + r_i$. Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ denote the real vector of v-values in the transcript. We will construct a matrix $\mathbf{S}'$ corresponding to the real execution. For $i \in I_1$ we set i^{th} column of $\mathbf{S}'$ arbitrarily. For $i \in I_2$ we set i^{th} column of $\mathbf{S}'$ as $(\mathbf{s}'_i, r'_i)$, where $(\mathbf{s}'_i, r'_i)$ are extracted inputs corresponding to the valid arguments of knowledge supplied by W_i. Due to binding property of the commitment scheme, and the soundness of zero knowledge arguments, we also note that $v'_i = \langle \boldsymbol{\gamma}, \mathbf{s}'_i \rangle + r'_i$ for $i \in I_2$ with overwhelming probability. For $i \notin I_1 \cup I_2$, we set i^{th} column of $\mathbf{S}'$ to honest inputs $(\mathbf{s}_i, r_i)$. Notice that $I_1 = E_1 \subseteq E_1 \cup E_2$. Let E denote the column indices where $\mathbf{S}$ and $\mathbf{S}'$ differ. Clearly $I_2 \subseteq E$ and moreover $|E| \leq \lambda < (n-t)/3$. From Lemma 2, it follows that with overwhelming probability $E = \{i \in [n] : \langle (\boldsymbol{\gamma}, 1), \mathbf{S}'_i \rangle \neq v_i\}$. Since $v'_i = \langle (\boldsymbol{\gamma}, 1), \mathbf{S}'_i \rangle$ for $i \in I_2$, it follows that $v'_i \neq v_i$ for $i \in I_2$ and hence $I_2 \subseteq E_2 \subseteq E_1 \cup E_2$. Thus $I_1 \cup I_2 = E_1 \cup E_2$ with overwhelming probability as needed to be shown.

Efficiency. Each worker communicates $O(\log \ell)$ elements of $\mathbb{G}$, most of them as part of $\mathsf{NI\text{-}CSP}$ proofs in Step 4. This gives an overall communication complexity of $O(n \log \ell \log |\mathbb{G}|)$. Computationally, the workers incur $O(\ell)$ exponentiations as part of generating the $\mathsf{NI\text{-}CSP}$ proofs. The verifier incurs $O(\ell)$ exponentiations (which can be combined into $O(1)$ multiexponentiations of size $O(\ell)$ each) and additionally Reed Solomon Decoding to identify corrupt messages.

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

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

  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 special 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}).

Robust Complete DPoK for BBS. We build upon Π_{rob} to propose a distributed proof of knowledge 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 $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ (Figure 4.1), 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 $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.
- **W_i's inputs :** 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 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 $d = b^u \cdot h_0^{-r}$. $\mathcal{P}$ 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 W_i.
  2. The workers 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 $W_i, i \in [n]$ 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 $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt\text{-}rob}}^{\mathsf{pv}}$, 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 $W_1, \ldots, W_n$. 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 $\Pi_{\mathsf{bbs\text{-}auth\text{-}rob}}^{\mathsf{pv}}$.

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.

6 Our (Non-Robust) Compiler for Authenticated MPC

In this section, we build upon the above distributed POKs for PS signatures to present a non-robust version 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}. We begin by fixing some notation, and then 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 UC 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 additionally assume that each party P_i holds a PS signature σ_i on its input 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 the parties to prove authenticity of their inputs to each other by proving knowledge of a valid PS 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 of the other parties P_j for $j \neq i$ act as a verifier; see Section A.3 for details). For simplicity, we first present 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 $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$. 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 authenticated MPC protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$ as described below.

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

- $\overline{\Pi}_{\mathsf{sh}}$: This phase is identical to Π_{sh}, i.e., each party P_i shares its input x_i to all other parties exactly as in Π_{sh}.
- $\overline{\Pi}_{\mathsf{on}}$: In this phase, the parties do the following:
  - The parties jointly execute the interactive phase of $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$. If any party outputs 0 at the end of this phase, the protocol aborts.
  - Otherwise, the parties jointly execute Π_{on}.

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

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

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

Theorem 6 ((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) $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth}}$ satisfies soundness and zero-knowledge, our compiled authenticated MPC protocol Π_{ampc} securely emulates the ideal functionality $\mathcal{F}^{\mathsf{auth,abort}}_{\mathsf{MPC}}$.*

To prove this theorem, we first construct a simulator for the Π_{ampc} protocol, and then prove the indistinguishability of the simulation from a real-world execution of Π_{ampc}. Let $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$ be a non-uniform PPT simulator that securely emulates an ideal-world execution of Π_{mpc} in a manner that is computationally indistinguishable from a real-world execution of Π_{mpc}. Also, we denote by $\mathsf{Ext}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ and $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ the extraction and ZK simulation algorithms corresponding to 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 $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$ be the simulator for Π_{mpc}, and let $\mathsf{Ext}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ and $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ be the extraction and ZK simulation algorithms corresponding to the underlying $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol. Also, 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).
- For each corrupt party P_i s.t. $i \in \mathcal{C}$, let $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_i$ denote the instance of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ used by the parties to prove authenticity of the input $\mathbf{x}_i$ (with corrupt party P_i acting as the prover, and all of the remaining parties acting as both workers and verifiers). Use the extractor Ext to extract from $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_i$ the input $\mathbf{x}_i$ of the corrupt party P_i.[8]
- If there exists some corrupt party P_i s.t. $i \in \mathcal{C}$ for which extraction fails, abort. Otherwise proceed to the next step.
- For each honest party P_j s.t. $j \in \mathcal{H}$, simulate a (distributed) proof of knowledge of a PS signature by using $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ to simulate an instance $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_j$ of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ (where P_j acts as the prover, and all of the remaining parties act as both workers and verifiers).[9]
- Finally, simulate the online phase $\overline{\Pi}_{\mathsf{on}}$ by invoking $\mathsf{Sim}_{\mathsf{on}}$ with the extracted inputs of the corrupt parties $\{\mathbf{x}_i\}_{i \in \mathcal{C}}$.

Completing the Security Proof. We now prove the UC 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 $\mathsf{Sim}_{\mathsf{sh}}$.
- $\{\mathsf{Hyb}_{2,i}\}_{i \in [0, |\mathcal{C}|]}$: Hybrid $2, 0$ is identical to hybrid 1, while for each $i \in [1, |\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{2,i}$ is identical to $\mathsf{Hyb}_{2,(i-1)}$ except that we use extracted input for corrupt party P_i s.t. $i \in \mathcal{C}$ during of the protocol. More concretely, for corrupt party P_i, let $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_i$ denote the instance of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ used by the parties to prove authenticity of the input $\mathbf{x}_i$ (with corrupt party

[8] Recall that the extractor Ext relies on the forking lemma to extract $\mathbf{x}_i$ by forking each party acting as a worker as part of $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_i$.

[9] Recall that $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ allows simulating the corresponding messages on behalf of all of the honest parties $\{P_j\}_{j \in \mathcal{H}}$ upon receipt of the messages from the corrupt parties $\{P_i\}_{i \in \mathcal{C}}$ using the random oracle RO).

P_i acting as the prover, and all the other parties acting as both workers and verifiers). We use the extractor Ext to extract from $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_i$ the input $\mathbf{x}_i$ of the corrupt party P_i, and use this extracted input for the rest of the protocol. If extraction fails, we abort.

– $\{\mathsf{Hyb}_{3,j}\}_{j\in[0,n-|\mathcal{C}|}$: Hybrid $3,0$ is identical to hybrid $\mathsf{Hyb}_{2,|\mathcal{C}|}$, while for each $j \in [1, n - |\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{3,j}$ is identical to $\mathsf{Hyb}_{3,(j-1)}$ except that we use a 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 PS signature $\sigma_{|\mathcal{C}|+j}$, we simulate a (distributed) proof of knowledge of a PS signature by using $\mathsf{Sim}_{\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}}$ to simulate an instance $\left(\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}\right)_{|\mathcal{C}|+j}$ of the protocol $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}dpok\text{-}opt}}$.

– Hyb_4: This hybrid is identical to $\mathsf{Hyb}_{3,n-|\mathcal{C}|}$ except that we simulate the online phase $\overline{\Pi}_{\mathsf{on}}$ of the underlying Π_{mpc} protocol by invoking $\mathsf{Sim}_{\mathsf{on}}$ with the extracted inputs of the corrupt parties $\{\mathbf{x}_i\}_{i\in\mathcal{C}}$.

$\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$. This follows from a simple argument based on the UC 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 the UC security of the underlying Π_{mpc} protocol.

$\mathsf{Hyb}_{2,i-1} \approx_c \mathsf{Hyb}_{2,i}$. This follows from the soundness of the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol. Again, suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{2,(i-1)}$ and $\mathsf{Hyb}_{2,i}$ for some $i \in [1, |\mathcal{C}|]$. It is easy to see that $\mathcal{A}$ can be used to construct a PPT adversary $\mathcal{A}'$ that breaks the soundness guarantees of the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol.

$\mathsf{Hyb}_{3,j-1} \approx_c \mathsf{Hyb}_{3,j}$. This follows from the ZK property of the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol. Again, suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{3,(j-1)}$ and $\mathsf{Hyb}_{3,j}$ for some $j \in [1, n - |\mathcal{C}|]$. It is easy to see that $\mathcal{A}$ can be used to construct a PPT adversary $\mathcal{A}'$ that breaks the ZK property of the $\Pi_{\mathsf{bbs\text{-}dpok\text{-}opt}}$ protocol.

$\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_{3,n-|\mathcal{C}|}$. This again follows from a simple argument based on the UC security of the underlying Π_{mpc} protocol. Suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_4 and $\mathsf{Hyb}_{3,n-|\mathcal{C}|}$. 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 UC security of the underlying Π_{mpc} protocol.

This completes the proof of Theorem 6.

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 now summarize the overheads incurred by (the optimized version of) our non-robust compiler and compare it with the overheads incurred by existing approaches for achieving MPC protocols with authenticated inputs [BJ18,ADEO21] (the comparison is also summarized in Table 3). The method in [BJ18] incurs a broadcast of $O(\ell)$ to authenticate each input (as the signature scheme used is not succinct). Computationally the prover incurs $O(\ell n)$ exponentiations and $O(\ell)$ pairings, while the verifiers incur $O(\ell)$ exponentiations and $O(\ell)$ pairings. Thus cumulatively (over all authentications), the communication overhead is $O(\ell n)$, while the computational overhead is $O(\ell n)$ exponentiations + $O(\ell n)$ pairings. The work [ADEO21] substantially improves upon [BJ18] by using a succinct signature scheme (Pointcheval-Sanders), and leveraging linear secret sharing isomorphisms between the scalar field $\mathbb{F}$ and elliptic curve groups to verify signatures in MPC. They realize input authentication by computing scalar products over shares and reconstructing the final shares to all parties. This approach incurs $\approx n^2 + 8n$ communication for each authentication. However, using a random linear combination, one reconstruction also suffices for n input authentications, thus yeilding a total communication complexity of $\approx 9n^2$. Computational overhead is $O(\ell n)$ exponentiations and $O(n)$ pairings due to n invocations of scalar multiplication protocol.

Protocol	Communication Complexity	Computational Complexity
$\Pi_{\mathsf{bbs\text{-}auth}}$ (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.
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

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.

7 Compiler for Authenticated MPC

In this section, we present our compiler for authenticated MPC that builds upon our distributed (robust complete) proofs of knowledge for BBS signatures. We define below our stronger ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{authid}}$ for authenticated MPC.

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

Inputs

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

Verify Authenticity

1. If $\mathsf{Ver}(\mathsf{pk}, x_i, \sigma_i) \neq 1$ for some party P_i, then output a set of corrupted parties $\mathcal{C}$ 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 Compiler. Our compiler takes as input any secret-sharing-based MPC protocol $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$, 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 (we assume that this sharing is done using a linear secret-sharing scheme), and Π_{on} denotes the remaining steps of the protocol Π_{mpc}. We assume that each party P_i holds a BBS signature σ_i on its input x_i with respect to a common public verification key pk. Let $\Pi \in \{\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}, \Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}rob}}\}$ denote a DPoK for BBS signatures. Our compiler outputs an authenticated MPC protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$, where $\overline{\Pi}_{\mathsf{sh}}$ is identical to Π_{sh}, and where $\overline{\Pi}_{\mathsf{on}}$ is as follows:

1. The parties jointly execute the interactive phase of the DPoK protocol Π. If any party outputs 0 at the end of this phase, the protocol Π_{ampc} aborts.
2. Otherwise, the parties jointly execute Π_{on}.

Non-Robust Compiler. When $\Pi = \Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}opt}}$, we achieve a non-robust version of our compiler. The corresponding construction and the security theorem is in Section 6 (Theorem 6).

Robust Compiler. For the case where $\Pi = \Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}rob}}$, the security theorem is as follows.

Theorem 7 (Security of Robust Π_{ampc}). *Assuming that: (a) the MPC protocol Π_{mpc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, and (b) $\Pi^{\mathsf{pv}}_{\mathsf{bbs\text{-}auth\text{-}rob}}$ satisfies soundness, zero-knowledge and robust completeness for a maximum corruption threshold of $t < n/4$, our compiled authenticated MPC protocol Π_{ampc} 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. 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).

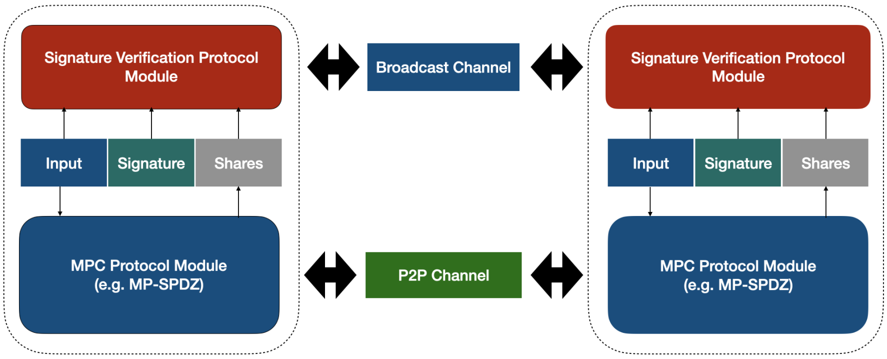

Fig. 2: Component diagram illustrating integration of our scheme with existing MPC frameworks.

	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.

8 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. 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. In MP-SDPZ, this interface is natively supported using `write_to_file()` call, which dumps the shares of a secret value into a file.

Our extension consists of a binary (written in `C++`) implementing our DPoK for BBS signatures using the `libff` library for elliptic curve operations [lib]. At a high level, we: (i) run the sharing phase of the underlying MPC protocol using MP-SPDZ, (ii) provide the input shares from the MP-SPDZ interface to our binary, (iii) run the DPoK over broadcast channels on the input-shares, and (iv) either abort (if the signature verification fails for any of the inputs), or (iv) we resume the computation on the shared inputs using MP-SPDZ. To share the auxiliary inputs during the preprocessing phase of our DPoK, we leverage the point-to-point communication infrastructure of the underlying MP-SPDZ library by augmenting the protocol input with these auxiliary inputs. Note that we could equivalently build point-to-point channels into our binary, but leveraging the existing framework for this purpose simplifies the implementation.

While the components of the underlying library interact over point to point channels as usual, our extension can be configured with a broadcast functionality when available. When broadcast is not available, we will need to realize it cryptographically (with associated communication overhead). The overall architecture of our implementation is illustrated in Figure 2.

The public parameters of our BBS+ scheme 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. We also compare our approach with prior works in Tables 1, 2 and 3. 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.

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.

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.

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.

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`.

lib. libff: C++ library for finite fields and elliptic curves. https://github.com/scipr-lab/libff. `https://github.com/scipr-lab/libff`.

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`.

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 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 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 A for a more formal exposition.

Definition 7 (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.

A.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. $\mathcal{P} \to \mathcal{V}$: α.
  3. $\mathcal{V} \to \mathcal{P}$: $c \leftarrow_R \mathbb{Z}_p$.
  4. $\mathcal{P} \to \mathcal{V}$: $\mathbf{s}' = c\mathbf{s} + \mathbf{r}$.
  5. $\mathcal{V}$ checks: $g^{\mathbf{s}'} = \alpha z^c$.

We also describe another variant of PS Signature Scheme, based on a stronger assumption (Assumption 1 in [PS16]), that leads to much more efficient distributed prover protocols. This variant is same as the one described in Definition 7, 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 8 (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 7, 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.*

A.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 $\big(\sigma_1 = \sigma_1', \sigma_2 = \sigma_2'(\sigma_1')^{-t}(Y_{\ell+1})^{-s}\big)$. To see that (σ_1, σ_2) is a valid signature we verify:

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

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

A.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 $(W_1, \ldots, W_n)$ 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 A.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 $\mathbf{m}$
- **W_i's inputs :** 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 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 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 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 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 8. *Assuming that the discrete log assumption holds over the groups $\mathbb{G}_1$ and $\mathbb{G}_2$, the proposed protocol $\Pi_{\mathsf{ps\text{-}dpok}}$ as described above achieves perfect completeness, $(2, k_1, \ldots, k_{(\log_2(\ell+1)-1)})$-special soundness with $k_i = 3$ for all $i = [\log_2(\ell + 1) - 1]$, and special honest-verifier zero-knowledge.*

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 formutation 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 A.2. The improved protocol $\Pi_{\mathsf{ps\text{-}dpok\text{-}opt}}$ appears in Figure A.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 $\mathbf{m}$
– **W_i's inputs :** 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 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 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 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.

A.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 $W_1, \ldots, W_n$. 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 $\Pi_{\textsf{cd-pok}}$ or $\Pi_{\textsf{ps-dpok-opt}}$. 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 3.1). Typically, the input authentication will be followed by computation phase to compute a function f on the inputs. We defer those details to Section 6. For now we describe a protocol that authenticates inputs of all parties (against a common public key pk for simplicity). The protocol $\Pi_{\textsf{ps-auth}}$ is described in Figure A.4.

Protocol $\Pi_{\textsf{ps-auth}}$

- **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 $\Pi_{\textsf{ps-dpok-opt}}$ 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 $\Pi_{\textsf{ps-dpok-opt}}$. 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 $\Pi^{\textsf{pv}}_{\textsf{ps-auth}}$ that achieves soundness and zero-knowledge, by running n instances of the publicly verifiable protocol $\Pi^{\textsf{pv}}_{\textsf{ps-dpok-opt}}$ outlined earlier as opposed to $\Pi_{\textsf{ps-dpok-opt}}$.

Optimized Version. We now present an optimized variant of the protocol $\Pi_{\textsf{ps-auth}}$ 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 $\Pi_{\textsf{cd-pok}}$ (one corresponding to each instance of $\Pi_{\textsf{ps-dpok-opt}}$) into a single instance of $\Pi_{\textsf{cd-pok}}$, using a random challenge. Observe that in phase j of the above protocol, the parties run an instance of $\Pi_{\textsf{cd-pok}}$ 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 $\Pi_{\textsf{ps-dpok-opt}}$. 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 $\Pi_{\textsf{ps-auth-opt}}$

- **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 $\mathbf{y}_i = \sum_{j \in [n]} \gamma^j (\mathbf{m}_{ij}, t_{ij})$, $w_i = \sum_{j \in [n]} v_{ij} \gamma^j$ 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 $\mathbf{y}_i, w_i$ 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.

A.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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where a set of distributed provers $(W_1, \ldots, W_n)$ 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 A.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 $\mathbf{m}$
– **W_i's inputs :** 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 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 W_i and $\mathcal{V}$ locally set $\mathbf{g} = (\tilde{g}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.
4. Each 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 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.