Collaborative CP-NIZKs: Modular, Composable Proofs for Distributed Secrets

Mohammed Alghazwi
University of Groningen
m.a.alghazwi@rug.nl

Tariq Bontekoe
University of Groningen
t.h.bontekoe@rug.nl

Leon Visscher
University of Groningen
l.visscher.2@student.rug.nl

Fatih Turkmen
University of Groningen
f.turkmen@rug.nl

***Abstract*—Non-interactive zero-knowledge (NIZK) proofs of knowledge have proven to be highly relevant for securely realizing a wide array of applications that rely on both *privacy* and *correctness*. They enable a prover to convince any party of the correctness of a public statement for a *secret witness*. However, most NIZKs do not natively support proving knowledge of a secret witness that is distributed over multiple provers. Previously, collaborative proofs [51] have been proposed to overcome this limitation. We investigate the notion of composability in this setting, following the Commit-and-Prove design of LegoSNARK [17]. Composability allows users to combine different, specialized NIZKs (e.g., one arithmetic circuit, one boolean circuit, and one for range proofs) with the aim of reducing the prove generation time. Moreover, it opens the door to efficient realizations of many applications in the collaborative setting such as mutually exclusive prover groups, combining collaborative and single-party proofs and efficiently implementing publicly auditable MPC (PA-MPC).**

We present the first, general definition for collaborative commit-and-prove NIZK (CP-NIZK) proofs of knowledge and construct distributed protocols to enable their realization. We implement our protocols for two commonly used NIZKs, Groth16 and Bulletproofs, and evaluate their practicality in a variety of computational settings. Our findings indicate that composability adds only minor overhead, especially for large circuits. We experimented with our construction in an application setting, and when compared to prior works, our protocols reduce latency by 18–$55\times$ while requiring only a fraction (0.2%) of the communication.

I. Introduction

A zero-knowledge proof (ZKP) [37] is a cryptographic construction, enabling a prover to convince a verifier of the truth of a statement, without revealing anything other than its validity. Its zero-knowledge property ensures that *privacy* of the prover's secrets is preserved, whilst its soundness property guarantees that the statement is valid (i.e., we have *correctness*) and that the prover did not cheat. In this work, we focus on a broad class of ZKPs known as NIZK proofs of knowledge. They are *publicly verifiable* due to their non-interactiveness, i.e., the verifier is not involved in proof generation. Being a proof of knowledge entails that the prover not only convinces the verifier of the validity of a statement, but also that it *knows* the secret data, or *witness*, for which the statement is valid. Note, that zk-SNARKs are a subset of NIZK proofs of knowledge, i.e., they form the subset of succinct proofs.

In recent years, many different constructions for NIZK proofs have been proposed for generic statements, e.g., those expressible as an arithmetic circuit. Each of these constructions comes with its own advantages, e.g., no trusted setup, constant proof size, efficient verification, or relying only on standard cryptographic assumptions. However, these generic schemes have a few major limitations [51]:

1) *Proof generation is costly;*
2) *The witness should be known by the prover (a single party).*

The former refers to the time/space it takes to generate proofs. The latter instead refers to the limitation that the generation of a proof where the witness is *distributed* over multiple parties is not directly supported.

At the same time, there is an array of specialized NIZK schemes, e.g., [24], [60], [17], that enable provers to generate proofs for specific types of statements more efficiently. This brings us to our third and final major limitation:

3) *Proof schemes are not efficiently composable.*

Thus, we cannot natively benefit from the distinguishing advantages of different schemes, by creating a proof for a single statement using a composition of different schemes.

Collaborative proofs. The first two limitations could be tackled by considering the notion of a distributed prover (where all provers have access to the full witness). While this would not directly improve efficiency, it would enable the outsourcing of proof generation in a privacy-preserving manner with the additional benefit of load distribution [61], [34], [21].

Even more interesting is the case where the secret witness is also distributed amongst multiple provers, i.e., no prover knows the full witness. This is particularly useful when several parties wish to prove a statement about secret data distributed amongst them without revealing their part of the witness to each other or the verifier. Recently, a generic definition for collaborative zk-SNARKs [51] was proposed as a solution for this setting. Notably, this approach can also be used to generically construct PA-MPC [6]. However, we note that for PA-MPC, the provers need to open a large number of commitments inside a proof circuit, which is computationally inefficient. As we will show in this work, our proposal for collaborative CP-NIZKs can be used to make this construction scalable.

Composability. In accordance with the third limitation, we observe that proof statements are often composed of several substatements, for example by conjunction (see Section IV-B). One substatement may be more efficiently proved in one scheme (e.g., arithmetic circuits NIZK) and a second more efficiently in another scheme (e.g., specialized or boolean circuit NIZK). To benefit from both the efficiency of specialized NIZKs and the flexibility of general-purpose NIZKs, it

would be ideal to combine multiple schemes to gain efficiency. At first glance, it may seem sufficient to naively generate a proof for each substatement and have the verifier check all proofs separately. However, since the verifier does not know the witness, which is often (partially) shared amongst substatements, it cannot verify whether the proofs are for the same witness, i.e., correctness is no longer guaranteed.

The approach where multiple subproofs are combined into a single main proof is called proof composition, and various works have proposed methods to achieve this [18], [1]. It has been most generically defined in [17], and is known as the *commit-and-prove* paradigm. Following this method, proofs can be generated for combined statements, where part of the witness has been committed to beforehand, thereby enabling composition, by linking proofs through witness commitments. Finally, we note that this approach is also *adaptive*: the commitments may be generated ahead-of-time and independently of the statement.

This work. It is evident that both collaborative proofs and the commit-and-prove paradigm provide a generic way of dealing with the limitations of most NIZK schemes. However, neither approach solves these limitations simultaneously, even though they are of equal importance for improving the efficiency and applicability of ZKPs. In this paper, we introduce collaborative CP-NIZKs, combining the best of both worlds to enable efficient proving in settings that are otherwise not (natively) supported, and sketch several such use cases for their use (Section I-A). Therefore, our work is centered around the following question:

Can users efficiently prove knowledge of a distributed witness using adaptive, composable NIZK proofs, and does proof composition in this setting result in faster proof generation?

Our contributions. We answer this question positively, by defining, implementing and evaluating the novel concept we call collaborative CP-NIZK. In summary, our contributions are as follows:

- We formally define collaborative CP-NIZKs (Section IV).
- We propose two methods for generating Pedersen-like commitments collaboratively (Section V-A).
- We design two collaborative CP-NIZKs by transforming two existing (commit-and-prove) provers into their MPC counterparts (Section V). Specifically, we construct a collaborative CP-SNARK from LegoGro16 [17] (efficient verification and small proof size) and collaborative, commit-and-prove Bulletproofs [16] (no trusted setup, logarithmic proof size, efficient range proofs).
- We implement all our collaborative CP-NIZKs and collaborative commitment schemes in Arkworks [4] in a modular fashion and make our code available (Section VI). Modularity allows for future extensions based on novel NIZK and MPC schemes.
- We evaluate our constructions and demonstrate that our approach incurs only a minor overhead, which is more than compensated for by the benefits of modularity and composition (Section VII). In our experiments, we show several scenarios in which composition significantly improves performance. Splitting large statements into smaller substatements and using a combination of CP-NIZKs lead to improved performance compared to using a single NIZKs (Section VII-E). Additionally, splitting prover groups improves efficiency by 1.3–$2\times$. When compared to the construction of [51], our protocols show an improvement of 18–$55\times$ when both constructions are applied in a realistic application scenario (Section VIII).

A. Example Use Cases.

In this section, we discuss some example use cases, that benefit from collaborative CP-NIZKs. As mentioned in [51] many real-world and conceptual settings ask for collaborative proofs. Our collaborative CP-NIZKs apply to similar use cases, but additionally offer significant performance improvements and modularity. Moreover, many applications require users to commit to their data, which makes our construction a more efficient and natural choice.

Auditing. Many government organizations, regulatory bodies, and financial institutions regularly perform audits on companies or individuals. Oftentimes, these audits also involve data from external sources, next to information already held by the party under audit.

Consider the process of a mortgage application. In general, the bank needs to verify that the applicant satisfies certain requirements, i.e., sufficient, stable income, decent credit score, no other mortgages, potential criminal history, et cetera. The majority of this information is privacy-sensitive. Since the banks need only verify that the requirements are satisfied, we can significantly reduce the privacy loss by replacing these checks by ZKPs [50], [43].

The applicant can thus construct a proof for all statements together with the external parties that hold the data using collaborative NIZKs. Note that the applicant often cannot access this external data themselves. Since the number of external parties is rather large, it would be infeasible to create one large collaborative proof. Fortunately, most statements will be largely disjunct. Therefore, we can use the composability of collaborative CP-NIZKs, by splitting the proof into several parts, such that only a small subset of parties is needed to prove each part. By using the same commitments for all parts, correctness is guaranteed.

Moreover, the eventual CP-NIZKs proof guarantees public verifiability, due to its non-interactive nature. This allows regulatory bodies to confirm that banks adhere to the rules when giving out mortgages, without seeing any sensitive data.

The idea, of splitting a large prover group into several smaller parts, can be applied to many practical use cases similar to this. Not only does it improve efficiency between parties, it also reduces the risk of aborts due to adversarial behavior, and removes the need for secure communication channels between all parties.

PA-MPC for e-voting and online auctions. Secure multiparty computation (MPC) allows a group of parties to collaboratively evaluate a function, whilst guaranteeing input privacy and correctness. Generally, however, correctness cannot be verified by parties that are not involved in the computation. PA-MPC [6], [44] aims to solve this problem by making results publicly verifiable.

While many applications are thinkable, two prominent ones are found in online auctions and e-voting. In fact, verifiable auctions and e-voting have been an active topic of research for a few decades [23], [45], [57], [19], [38], [47], [56], [2], [52]. Both voting and auctions often have a large number of participants, making it computationally infeasible to involve all parties in the computation. Rather, the computation is run by a fixed group of servers, that take inputs from participants and compute the result, i.e., highest bid or vote tally. Generally, participants do not send their plain input to the server but rather a transformation that hides its true value, e.g., a secret sharing, hiding commitment, or blinded value, depending on the privacy-preserving protocol that is used.

So, while participants can guarantee privacy of their own inputs, they have no control over the results being computed correctly. But, by using PA-MPC and adding a correctness proof to the computation, each participant can verify the correctness of the result, thereby guaranteeing an honest auction or voting process. To verify the proof, one often needs access to all commitments to the original input data. Collaborative CP-NIZKs are a very efficient way of constructing proofs for committed data, as we argue in Section IV-C.

Improving efficiency through modularity. Collaborative CP-NIZKs can be constructed from any combination of MPC and CP-NIZK schemes, as we show in Section IV and onwards. Moreover, we note that many regular NIZKs can be transformed into CP-NIZKs for select commitment schemes [17].

The modularity of our construction in combination with its composition property, i.e., we can compose proofs for differing statements on partially overlapping commitments, gives its users significant freedom. This freedom can be used to improve efficiency, by cleverly combining MPC and CP-NIZK schemes, depending on the users' requirements and proof statements. This potential for efficiency improvements is especially useful to reduce the overhead introduced by MPC, as exemplified in the use cases above.

II. Related Work

Below, we give an overview of prior works on (adaptive) composability and distributed ZKP provers and discuss their relation to our unifying construction. In Appendix A, we also provide an additional discussion on existing approaches for distributed provers on committed data in PA-MPC, which unlike our work are tailored to one specific, often monolithic, combination of MPC, commitment, and NIZKs scheme.

Commit-and-Prove constructions. The concept of combining different NIZKs to improve efficiency when handling heterogeneous statements, i.e., handling substatements of a larger proof statement with tailored NIZK schemes has previously been considered in [18], [1]. These works, however, only focus on combining select schemes, e.g., [1] combines Pinocchio [53] with Σ-protocols.

Our work builds upon the definition for commit-and-prove zk-SNARKs (CP-SNARKs) as given in LegoSNARK [17]. In their work, the authors flexibly define CP-SNARKs, i.e., succinct CP-NIZKs. And, they show how to compose different CP-SNARKs for Pedersen-like and polynomial commitments. Additionally, they define the concept of commit-carrying zk-SNARKs (cc-SNARKs), a weaker form of CP-SNARKs, and show how to compile cc-SNARKs into CP-SNARKs. Specifically, they introduce a commit-and-prove version of Groth16 [40] for Pedersen-like commitments, that we make collaborative in this work.

Next to this, we observe a number of schemes with commit-and-prove functionality that were introduced before LegoSNARK. Geppetto [25] can be used to create proofs on committed data, however, the commitment keys depend on the relation to be proven, i.e., it is not adaptive. Adaptive Pinocchio [59] removes this restriction, making it a CP-SNARK according to the definition of LegoSNARK. Other CP-SNARKs are presented in [49] and [39], however, they rely on specific commitment schemes, that may not be composable with other NIZK schemes. Finally, we note that [25] and [49] give alternative definitions for CP-SNARKs, however, we choose LegoSNARK over these definitions due to its flexibility.

Distributed provers. The notion of distributed provers, i.e., N parties respectively knowing N witnesses, has already been considered by Pedersen in 1991 [54]. He describes the general paradigm of distributing prover algorithms using MPC and presents applications to undeniable signatures. More recently, the authors of Bulletproofs [16], discuss how to use MPC to create aggregated range proofs, however this does not extend to generic arithmetic circuits. Both solutions are specific to one honest-majority MPC scheme and not very efficient.

The authors of [27] aim to improve upon existing work, by introducing a compiler to generate distributed provers for ZKPs based on the interactive oracle proof (IOP) paradigm [11]. They construct distributed versions of Ligero [3], Aurora [10], and a novel scheme called Graphene. Unlike our work, their solution is restricted to a particular subset of ZKPs, and does not consider composability.

Another solution, that we use as inspiration for our constructions is presented by Ozdemir and Boneh[51], who define collaborative zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). They show how to construct these from generic zk-SNARKs in combination with MPC. Their implementation is evaluated for Groth16 [40], Marlin [20], and Plonk [33]. Finally, they describe how to implement collaborative Fractal [22].

Additionally, we note three schemes that consider distributing the prover to increase efficiency, e.g., for outsourcing purposes: DIZK [61], Eos [21], and zkSaaS [34]. DIZK distributes the prover across multiple machines in one cluster, by identifying the computationally costly tasks and distributing these. Yet, their method is not privacy-preserving and thus not applicable to our setting. zkSaaS [34], on the other hand, does guarantee privacy-preservation when outsourcing the prover to a group of untrusted servers. The authors evaluate their construction for three zk-SNARKs: (Groth16 [40], Marlin [20], and [33]). Eos achieves similar results, but only considers zk-SNARKs based on polynomial IOPs and polynomial commitments, such as Marlin [20]. Neither Eos nor zkSaaS considers composability and modularity, and thus their advantages.

Other notions of distribution. Lastly, we observe that distribution is also considered for other ZKP aspects. Feta [7]

considers threshold distributed verification in the designated-verifier setting. To increase the reliability of the trusted setup, several works present solutions for replacing the setup algorithm by a distributed setup ceremony [9], [46].

III. Preliminaries

The finite field of integers modulo a prime p is denoted by $\mathbb{Z}_p$. Additionally, $\mathbb{G}$ is a cyclic group of order q, where $g, h \in \mathbb{G}$ are arbitrary generators, all group operations are written multiplicatively. $x \xleftarrow{\$} \mathbb{Z}_p^*$ denotes uniform random sampling of an element from $\mathbb{Z}_p^*$.

Some NIZK schemes make heavy use of bilinear groups $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are of prime order q. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, or *pairing*, such that $e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}$, for all $a, b \in \mathbb{F}_q$ and all $h_1 \in \mathbb{G}_1$, $h_2 \in \mathbb{G}_2$.

Vectors and matrices. We denote vectors as $\mathbf{v}$ and matrices as $\mathbf{A}$. For example, $\mathbf{A} \in \mathbb{Z}^{n \times m}$ represents a matrix with n rows and m columns, where $a_{i,j}$ is the element located in the ith row and jth column of $\mathbf{A}$. We denote multiplication of a scalar $c \in \mathbb{Z}_p$ and vector $\mathbf{a} \in \mathbb{Z}_p^n$ as $\mathbf{b} = c \cdot \mathbf{a} \in \mathbb{Z}_p^n$, with $b_i = c \cdot a_i$.

Additionally, 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 as $\mathbf{g}^{\mathbf{x}} = g_1^{x_1} \cdots g_n^{x_n}$. Also, $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i \cdot b_i$ denotes the inner product of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$, and $\mathbf{a} \circ \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n)$ their Hadamard product.

A vector polynomial is denoted by $p(X) = \sum_{i=0}^{d} \mathbf{p_i} \cdot X^i \in \mathbb{Z}_p^n[X]$ where coefficients are n-vectors, i.e., $\mathbf{p_i} \in \mathbb{Z}_p^n$. Given $0 \leq \ell \leq n$, we use $\mathbf{a}[:\ell] = (a_1, \ldots, a_\ell) \in \mathbb{Z}^\ell$ to denote the left slice and $\mathbf{a}[\ell:] = (a_{\ell+1}, \ldots, a_n) \in \mathbb{Z}^{n-\ell}$ for the right slice.

A. Multiparty computation

MPC is a collection of techniques that allow multiple parties to jointly compute a function on their combined inputs while preserving privacy of their respective inputs. Typically, N parties $P_1, \ldots, P_N$ collaboratively evaluate $f \colon \mathcal{X}^N \to Y$, where the i-th party P_i holds input $x_i \in \mathcal{X}$. A secure MPC protocol Π should at least satisfy *privacy*, *correctness*, and *independence of inputs* [48]. Informally, this means that no party should learn anything other than its prescribed output, that each honest party is guaranteed correctness of its received output, and that corrupted parties choose their inputs independently of honest parties' inputs.

Concretely, we say that Π is secure against t corrupted parties, if Π is secure against any adversary $\mathcal{A}$ corrupting no more than t out of N parties. Furthermore, security depends on the type of behaviour that corrupted parties may show. For *semi-honest*, or *honest-but-curious*, adversaries, corrupted parties follow the protocol specification as prescribed. $\mathcal{A}$ simply tries to learn as much information as possible from the corrupted parties' states. Adversely, in the *malicious* model, corrupted parties may deviate *arbitrarily* from the specified protocol to try and break security.

The majority of practical MPC protocols satisfy security in either model, only when *aborts* are allowed. In this *security-with-abort* model, the adversary may cause an abort of the protocol, thereby denying outputs to honest parties. We refer the reader to [35] for all formal definitions related to MPC.

Arithmetic circuits for MPC. An *arithmetic circuit* allows us to describe a large class of computations using elementary gates. Many generic MPC frameworks can be used to securely evaluate bounded size arithmetic circuits. A circuit is represented by a directed acyclic graph (DAG) consisting of wires and gates. Wires carry values in a finite field $\mathbb{Z}_p$, and can be either public or private. Each gate takes two input values and returns one output, either a *multiplication* or *addition* of its inputs. A subset of the wires is dedicated for assigning the circuit inputs and outputs.

SPDZ. Many MPC protocols are based on *secret sharing*, in which private inputs and intermediate values are represented by so-called *shares*. Oftentimes, inputs are secret shared using a t-out-of-N secret sharing scheme, where each value is split in N shares (one share per party), such that given at least t shares one can reconstruct the full input, and otherwise no information is revealed.

SPDZ [26] is an MPC framework based on *additive N-out-of-N secret sharing*, i.e., a value $y \in \mathbb{Z}_p$ is split into $y_1, \ldots y_N$, such that all shares together sum to y. We will use $[\![y]\!]_i$ to denote the share of y held by party i. Additionally, $[\![y]\!]$ will denote the vector of all shares of y. The SPDZ framework can be used to evaluate arbitrary arithmetic circuits of bounded size in the presence of *malicious* adversaries.

All shares in SPDZ are associated with a message authentication code (MAC) that is used to maintain correctness of the computation. A share together with its MAC is called an authenticated share. SPDZ uses authenticated shares to achieve active security, meaning that when a corrupted party deviates from the protocol, the honest parties can detect this and abort the protocol. SPDZ is thus a secure-with-abort protocol, being able to handle up to and including $N - 1$ corrupted parties. Since, each operation on an authenticated share updates both the value and its MAC, each operation takes double the time of a regular operation.

Note that, due to its additive property, arithmetic operations between public and shared values, as well as addition of shared values can be done 'for free', i.e., locally. However, multiplying two shared values requires some form of interaction. To solve this problem efficiently, SPDZ adopts somewhat homomorphic encryption to create a sufficiently large amount of Beaver triplets [8] in a circuit-independent *offline* phase. These triplets can be used to multiply shared values more efficiently during the *online* phase.

B. Commitments

A non-interactive commitment scheme is a 3-tuple of probabilistic polynomial time (p.p.t.) algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: Given the security parameter λ, this outputs the commitment key ck, which also describes the input space $\mathcal{D}$, commitment space $\mathcal{C}$, and opening space $\mathcal{O}$;
- $\mathsf{Commit}(\mathsf{ck}, u, o) \to c$: Given ck, an input u and opening o, this outputs a commitment c;
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \to \{0, 1\}$: Asserts whether (u, o) opens the commitment c under ck, returns 1 (accept) or 0 (reject),

and should satisfy at least the following properties:

- *Correctness:* For all ck and for each $u \in \mathcal{D}$, $o \in \mathcal{O}$, if $c = \mathsf{Commit}(\mathsf{ck}, u, o)$, then $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) = 1$;
- *Binding:* Given a commitment c to an input-opening pair (u, o), it should be hard to find (u', o') with $u' \neq u$, such that $\mathsf{VerCommit}(\mathsf{ck}, c, u', o') = 1$;
- *Hiding:* For any ck and every pair $u, u' \in \mathcal{D}$, the distribution (with respect to o) of $\mathsf{Commit}(\mathsf{ck}, u, o)$ should be indistinguishable from that of $\mathsf{Commit}(\mathsf{ck}, u', o)$.

C. Zero Knowledge Proofs

ZKPs [37] allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of the existence of a secret *witness* w for a public *statement* x, such that both satisfy an NP-relation $R \in \mathcal{R}_\lambda$, i.e., $(x, w) \in R$, for some family of relations $\mathcal{R}_\lambda$. In this work, we focus on NIZK proofs of knowledge, in which $\mathcal{P}$ not only proves existence of w, but also that it *knows* w. Specifically, we consider NIZKs in the common reference string (CRS) model [13], [28], [40] (where the CRS can be reused for any polynomial number of proofs), which are defined by a 3-tuple of p.p.t. algorithms:

- $\mathsf{KeyGen}(1^\lambda, R) \to (\mathsf{ek}, \mathsf{vk})$: takes the security parameter λ and a relation R and outputs the CRS $(\mathsf{ek}, \mathsf{vk})$.
- $\mathsf{Prove}(\mathsf{ek}, x, w)$. Given the evaluation key ek, statement x, and witness w, generates a valid proof π (if $(x, w) \in R$).
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to \{0, 1\}$. Given the verification key vk, x, and π, outputs 1 if π is valid and 0 otherwise.

A secure NIZK proof of knowledge should at least satisfy the following (informal) properties:

- *Completeness:* Given a true statement $(x, w) \in R$, an honest prover $\mathcal{P}$ is able to convince an honest verifier $\mathcal{V}$.
- *Knowledge soundness:* For any prover $\mathcal{P}$, there exists an extractor $\mathcal{E}^{\mathcal{P}}$ that produces a witness w such that $(x, w) \in R$, whenever $\mathcal{P}$ convinces $\mathcal{V}$ for any given x.
- *Zero-knowledge:* A proof π should reveal no information other than the truth of the statement x.

We refer to [28] for their formal definitions. When knowledge soundness only holds computationally, the scheme is called an *argument* of knowledge, rather than a *proof*. For simplicity, we refer to both as proofs in the remainder of this work unless we wish to specify it explicitly. An example of a secure a NIZK, that is also used in our work, is ***Bulletproofs*** [16], where non-interactiveness follows from the Fiat-Shamir heuristic [32].

When a NIZK argument of knowledge is also succinct, i.e., the verifier runs in $\mathsf{poly}(\lambda + |x|)$ time and the proof size is $\mathsf{poly}(\lambda)$, we call it a zk-SNARK [12]. In this work, we build upon the *Groth16* [40] zk-SNARK, to implement a collaborative CP-SNARK.

D. Collaborative NIZKs

Ozdemir and Boneh [51] define collaborative zk-SNARKs, building upon previous definitions [11]. They consider the setting of multiple provers wanting to prove a statement about a distributed witness, i.e., each party holds a vector of secret shares that together reconstruct the full witness.

When leaving out the requirement of *succinctness*, we obtain the general definition for collaborative NIZKs arguments of knowledge. These are defined analogously to regular NIZKs by a 3-tuple $(\mathsf{KeyGen}, \Pi, \mathsf{Verify})$, where KeyGen and Verify have the same definition. Π is an interactive protocol serving to replace Prove, where N provers with private inputs $w_1, \ldots, w_N$ together create a proof π for a given statement x.

Completeness and *knowledge soundness* for collaborative NIZKs arguments are defined analogously to their non-collaborative counterpart. On the other hand, *zero-knowledge* is replaced by the notion of *t-zero-knowledge*, which guarantees that t colluding, malicious provers cannot learn anything about the witnesses of other provers, other than the validity of the full witness. The relation between regular and collaborative NIZKs follows from [51]:

Theorem 1. *If* $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *is a NIZK argument of knowledge for* $\mathcal{R}_\lambda$*, and* Π *an MPC protocol for* Prove *for* N *parties that is secure-with-abort against* t *corruptions, then* $(\mathsf{KeyGen}, \Pi, \mathsf{Verify})$ *is a collaborative NIZK argument of knowledge for* $\mathcal{R}_\lambda$. (Proof follows directly from [51, Thm. 1].)

IV. Collaborative CP-NIZKs

A collaborative CP-NIZK argument of knowledge is an argument that, given x, is used to prove knowledge of $\mathbf{w} = (w_1, \ldots, w_N)$ such that $(x, w_1, \ldots, w_N) \in R$, where $w_i = (u_i, \omega_i)$ and $\mathbf{u} = (u_1, \ldots, u_N)$ opens a commitment $c_{\mathbf{u}}$. It is collaborative in the sense that N provers, who have *distributed knowledge* [41] of the witness vector $\mathbf{w}$, together construct a single argument of knowledge.

In practice, the commitment to $\mathbf{u}$ might be split over ℓ commitments $\{c_j\}_{j \in [\ell]}$. As noted in [17], this splitting is crucial to efficiently exploit the compositional power of CP-NIZKs, which we explain shortly after. We assume that the specification of this splitting is described in R.

Formally, a CP-NIZK argument of knowledge is defined by a 3-tuple of p.p.t. algorithms:

- $\mathsf{KeyGen}(1^\lambda, \mathsf{ck}, R) \to (\mathsf{ek}, \mathsf{vk})$: Given the security parameter λ, commitment key ck, and relation R, this algorithm outputs the CRS, i.e., the evaluation ek and verification vk keys;
- $\Pi(\mathsf{ek}, x, \{c_j\}_{j \in [\ell]}, \{\mathbf{u}_j\}_{j \in [\ell]}, \{\mathbf{o}_j\}_{j \in [\ell]}, \boldsymbol{\omega}) \to \pi$: In this interactive protocol, given ek, statement x, and commitments c_j, all N provers use their respective private inputs, i.e., commitment openings $(u_i, o_i)_j$ and non-committed witness ω_i, to produce a proof π;
- $\mathsf{Verify}(\mathsf{vk}, x, \{c_j\}_{j \in [\ell]}, \pi) \to \{0, 1\}$: Given vk, x, all c_j's, and π, this algorithm returns 1 when π is a valid proof and 0 otherwise.

Building upon definitions for CP-SNARKs in [17], we define commitment-enhanced (CE) relations as follows.

Definition 1 (Commitment-enhanced relation)**.** *Let* $\mathcal{R}_\lambda$ *be a family of relations* R *over* $\mathcal{D}_x \times \mathcal{D}_{\mathbf{u}} \times \mathcal{D}_{\boldsymbol{\omega}}$*, such that* $\mathcal{D}_{\mathbf{u}} = \mathcal{D}_{\mathbf{u}_1} \times \cdots \times \mathcal{D}_{\mathbf{u}_\ell}$*. Given a commitment scheme* $\mathsf{Comm} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{VerCommit})$ *with input space* $\mathcal{D}$*, commitment space* $\mathcal{C}$ *and opening space* $\mathcal{O}$*, such that* $\mathcal{D}_{\mathbf{u}_j} \subseteq \mathcal{D}, \forall j \in [\ell]$*. The CE version of* $\mathcal{R}_\lambda$ *is given by* $\mathcal{R}_\lambda^{\mathsf{Comm}}$*, such that:*

- *every* $R^{CE} \in \mathcal{R}_\lambda^{\mathsf{Comm}}$ *can be represented by a pair* (ck, R) *where* ck *is a possible output from* $\mathsf{Setup}(1^\lambda)$*,* $R \in \mathcal{R}_\lambda$*; and*

- *R^{CE} is the collection of statement-witness pairs $(\phi; \mathbf{w})$, with $\phi = (x, \{c_j\}) \in \mathcal{D}_x \times \mathcal{D}^\ell$ and $\mathbf{w} = (\{\mathbf{u}_j\}, \{\mathbf{o}\}, \boldsymbol{\omega}) \in \mathcal{D}^\ell \times \mathcal{O}^\ell \times \mathcal{D}_{\boldsymbol{\omega}}$ such that $\mathsf{VerCommit}(\mathsf{ck}, c_j, \mathbf{u}_j, \mathbf{o}_j) = 1, \forall j \in [\ell]$ and $(x, \{\mathbf{u}_j\}, \boldsymbol{\omega}) \in R$.*

A. Security properties

Using Definition 1 and building upon [51], [11], we present a unified definition for collaborative CP-NIZKs arguments:

Definition 2 (Collaborative CP-NIZK argument of knowledge)**.** *Given a family $\mathcal{R}_\lambda$ of relations R over $\mathcal{D}_x \times \mathcal{D}_{\mathbf{u}} \times \mathcal{D}_{\boldsymbol{\omega}}$ and a commitment scheme* Comm*. Let $\mathcal{U}(\lambda)$ denote the uniform distribution over all functions $\rho : \{0,1\}^* \to \{0,1\}^\lambda$, i.e., if $\rho \leftarrow\!\$\ \mathcal{U}(\lambda)$, then ρ is a* random oracle*. In the random oracle model,* $(\mathsf{KeyGen}, \Pi, \mathsf{Verify})$ *is a collaborative CP-NIZK argument of knowledge for $\mathcal{R}_\lambda$ and* Comm *with N provers, secure against t malicious provers if for all $\lambda \in \mathbb{N}$, $R \in \mathcal{R}_\lambda$, with corresponding $R^{CE} \in \mathcal{R}_\lambda^{\mathsf{Comm}}$ as in Definition 1:*

1) Completeness*: For all $(x, \{c_j\}; \{\mathbf{u}_j\}, \{\mathbf{o}_j\}, \boldsymbol{\omega}) \in R^{CE}$ and* ck*, the following is negligible in λ:*

$$\Pr\left[\mathsf{Verify}^\rho(\mathsf{vk}, x, \{c_j\}, \pi) = 0 \,\middle|\, \begin{array}{l} \rho \leftarrow\!\$\ \mathcal{U}(\lambda) \\ (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}^\rho(1^\lambda, \mathsf{ck}, R) \\ \pi \leftarrow \Pi^\rho(\mathsf{ek}, x, \{c_j\}, \{\mathbf{u}_j\}, \{\mathbf{o}_j\}, \boldsymbol{\omega}) \end{array}\right]$$

2) Knowledge soundness*: For all $(x, \{c_j\})$,* ck *and p.p.t. provers $\vec{\mathcal{P}} = (\mathcal{P}_1^*, \ldots, \mathcal{P}_N^*)$, there exists a p.p.t. extractor $\mathcal{E}$ such that the following is negligible in λ:*

$$\Pr\left[\begin{array}{l|l} (x, \{c_j\}; \mathbf{w}) \notin R^{CE} & \rho \leftarrow\!\$\ \mathcal{U}(\lambda) \\ \mathsf{Verify}^\rho(\mathsf{vk}, x, \{c_j\}, \pi) = 1 & (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}^\rho(1^\lambda, \mathsf{ck}, R) \\ & \pi \leftarrow \vec{\mathcal{P}}^\rho(\mathsf{ek}, \mathsf{vk}, x, \{c_j\}) \\ & \mathbf{w} \leftarrow \mathcal{E}^{\rho, \vec{\mathcal{P}}^\rho}(\mathsf{ek}, \mathsf{vk}, x, \{c_j\}) \end{array}\right]$$

Where $\mathcal{E}^{\rho, \vec{\mathcal{P}}^\rho}$ means that $\mathcal{E}$ has access to the random oracle ρ and can re-run $\vec{\mathcal{P}}^\rho(\mathsf{ek}, x, \{c_j\})$, where $\mathcal{E}$ can reprogram ρ each time, and only receives the output of $\vec{\mathcal{P}}$.

3) t-zero-knowledge*: For all p.p.t. adversaries $\mathcal{A}$ controlling $k \leq t$ provers $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$, there exists a p.p.t. simulator $\mathcal{S}$ such that for all $(x, c; \mathbf{w})$ and p.p.t. distinguishers $\mathcal{D}$*

$$\Pr\left[\mathcal{D}^\rho(\mathsf{tr}) = 1 \,\middle|\, \begin{array}{l} \rho \leftarrow\!\$\ \mathcal{U}(\lambda) \\ (\mathsf{ek}, \mathsf{vk}) \to \mathsf{KeyGen}^\rho(1^\lambda, R) \\ b \leftarrow (x, \{c_j\}; \mathbf{w}) \in R^{CE} \\ (\mathsf{tr}, \mu) \leftarrow \mathcal{S}^\rho(\mathsf{ek}, \mathsf{vk}, x, \{c_j\}, \\ \qquad\qquad w_{i_1}, \ldots, w_{i_k}, b) \end{array}\right] \stackrel{c}{\equiv}$$

$$\Pr\left[\mathcal{D}^{\rho[\mu]}(\mathsf{tr}) = 1 \,\middle|\, \begin{array}{l} \rho \leftarrow\!\$\ \mathcal{U}(\lambda) \\ (\mathsf{ek}, \mathsf{vk}) \to \mathsf{KeyGen}^\rho(1^\lambda, R) \\ \mathsf{tr} \leftarrow \mathsf{View}_{\mathcal{A}}^\rho(\mathsf{ek}, \mathsf{vk}, x, \{c_j\}, \mathbf{w}) \end{array}\right],$$

where tr denotes a transcript. $\mathsf{View}_{\mathcal{A}}^\rho$ is the view of $\mathcal{A}$ when the provers interact for the given inputs, where the honest provers follow Π and dishonest provers may deviate. μ is a partial function, and $\rho[\mu]$ is the function that returns $\mu(x)$ if μ is defined on x and $\rho(x)$ otherwise.

As a corollary, we obtain the following definition:

Definition 3 (cCP-SNARK)**.** *A collaborative CP-SNARK is a collaborative CP-NIZK that additionally satisfies* succinctness*: the verifier runs in $\mathsf{poly}(\lambda + |x|)$ time and the proof size is $\mathsf{poly}(\lambda)$.*

Completeness, *soundness*, and *zero-knowledge* are defined similarly to their (non-collaborative) CP-NIZK counterparts, however there are some distinctions (in agreement with [51]).

For collaborative proofs, knowledge soundness only guarantees that the N provers together 'know' the complete witness vector $\mathbf{w}$. However, it does not state anything about how this knowledge is distributed over the provers, i.e., there is no guarantee that party $\mathcal{P}_i$ knows w_i. Actually, an honest verifier, observing only the proof π, is not even able to determine the number of provers N that participated in Π. If desired, this 'limitation' could be circumvented, by including additional conditions in R that link elements of $\mathbf{w}$ to secrets that are known to be in possession of $\mathcal{P}_i$.

In the regular definition of zero-knowledge, the simulator $\mathcal{S}$ should be able to simulate a proof π for any $(x, \{c_j\}, \mathbf{w}) \in R^{\text{CE}}$, without having access to w. Observe that, in this case, it is sufficient to consider only valid pairs (x, w), since we consider an honest prover. However, for *t-zero-knowledge*, up to t provers may act maliciously, implying that we cannot make the same assumption. Actually, each malicious prover $\mathcal{P}_{i_k}$ can arbitrarily choose w_{i_k}. This is modelled by providing all w_{i_k}'s to $\mathcal{S}$ along with a bit b, denoting whether $(x, \{c_j\}, \mathbf{w}) \in R^{\text{CE}}$. All in all, t-zero-knowledgeness guarantees that, for no more than t malicious provers, nothing other than the validity of the witness $\mathbf{w}$ and (possibly) the entries w_{i_k} of the malicious provers is revealed.

Theorem 2. *Let* $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *be a CP-NIZK argument of knowledge for $\mathcal{R}_\lambda$ and* Comm*, and Π an MPC protocol for* Prove *for N parties that has security-with-abort against t corrupted parties, then* $(\mathsf{KeyGen}, \Pi, \mathsf{Verify})$ *is a collaborative CP-NIZK argument of knowledge for $\mathcal{R}_\lambda$ and* Comm *with N provers, secure against t malicious provers.*

Proof: A CP-NIZK argument of knowledge for $\mathcal{R}_\lambda$ and Comm is a NIZK argument of knowledge for $\mathcal{R}_\lambda^{\mathsf{Comm}}$ as defined in Definition 1, where $\mathbf{u}$, $\boldsymbol{\omega}$, $\mathbf{o}$, $\mathbf{u}_j$ are all single-element vectors, i.e., there is only one prover.

Similarly, a collaborative a CP-NIZK argument of knowledge with N provers for $\mathcal{R}_\lambda$ and Comm is a collaborative NIZK argument of knowledge for $\mathcal{R}_\lambda^{\mathsf{Comm}}$ as defined in Definition 1, where all vectors are of length N.

Thus, we are essentially in the situation of Theorem 1 and can conclude that security holds. ∎

We have opted to define security in the *random oracle model* in Definition 2, since some NIZK schemes are only proven secure in the random oracle model, such as those based on the Fiat-Shamir paradigm [36], [32], e.g., non-interactive *Bulletproofs* [16]. However, not all NIZK schemes depend upon the random oracle model for security, e.g., *Groth16* [40], in which case security can be defined analogously, but without giving parties access to a random oracle.

B. Composition of collaborative CP-NIZKs

To underscore the relevance of collaborative CP-NIZKs, we show how to compose two different CP-NIZK schemes on (partially) overlapping commitments and what benefits come with. We specifically focus on the novel, added benefits of composability for the collaborative setting.

Composition of relations with shared inputs. Building upon [17], consider two families of relations $\mathcal{R}_\lambda^a$ and $\mathcal{R}_\lambda^b$, such that $\mathcal{D}_\mathbf{u}^a$ and $\mathcal{D}_\mathbf{u}^b$ can be split as: $\mathcal{D}_\mathbf{u}^a = \mathcal{D}_\mathbf{u}^0 \times \mathcal{D}_\mathbf{u}^1$ and $\mathcal{D}_\mathbf{u}^b = \mathcal{D}_\mathbf{u}^0 \times \mathcal{D}_\mathbf{u}^2$. In other words, we consider two relations where part of the committed inputs (those in $\mathcal{D}_\mathbf{u}^0$) are identical. We define the family of conjunctions of these relations as $\mathcal{R}_\lambda^\wedge = \left\{ R_{R^a,R^b}^\wedge : R^a \in \mathcal{R}_\lambda^a, R^b \in \mathcal{R}_\lambda^b \right\}$, where $(x_a, x_b, \mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, (\boldsymbol{\omega}_a, \boldsymbol{\omega}_b)) \in R_{R^a,R^b}^\wedge$ if and only if $(x_a, \mathbf{u}_0, \mathbf{u}_1, \boldsymbol{\omega}_a) \in R^a \wedge (x_b, \mathbf{u}_0, \mathbf{u}_2, \boldsymbol{\omega}_b) \in R^b$.

Given a commitment scheme Comm, let $\mathsf{cCP}^{a/b}$ be the respective collaborative CP-NIZK arguments of knowledge for $\mathcal{R}_\lambda^{a/b}$ and Comm. Building a collaborative CP-NIZK argument $\mathsf{cCP}^\wedge$ for $\mathcal{R}_\lambda^\wedge$ and Comm from this is straightforward. It is sufficient to use the algorithms provided by $\mathsf{cCP}^{a/b}$ to compute two proofs π_a and π_b on the same commitment c^s to $\mathbf{u}^s$. Verifying these proofs for the same value of c^s is sufficient to guarantee correctness of the conjunction. For a more formal description and security proof of the non-collaborative construction we refer to [17] and note that our construction can be defined and proven secure analogously.

Next to this, we observe that this composition can be readily extended to more than two relations, by applying the above trick multiple times. Finally, we note that a disjunction of relations can be efficiently constructed using a simple trick with two additional witness entries, following [17].

Composing proofs with distinct prover sets. A useful observation, which opens an efficient way of tackling a wide range of practical use cases, is that the prover sets for $\mathsf{cCP}^{a/b}$ need not be equal. I.e., it is possible for two sets of provers with only partial overlap[1] to prove a single relation, as long as this relation can be split into parts. We can simply adopt the construction as described above for proving compositions of relations and use different prover sets for different 'subproofs'.

Improving efficiency and security by composition. We observe 4 main ways of using the composability of collaborative CP-NIZKs to significantly improve practical efficiency and security properties:

- By splitting a conjunction of several statements, such that each smaller statement contains witnesses held by a smaller set of provers, we can let each smaller statement be proven only by this smaller set of provers. Since many MPC protocols have communication that scales quadratically in the number of parties, having multiple smaller prover sets is likely much more efficient. Moreover, many MPC protocols require direct communication channels between each party and that all parties are present simultaneously during the online phase. This quickly becomes infeasible for large prover groups. By splitting the statement and prover sets into smaller parts, we avoid this issue.
- By splitting a large statement into several smaller ones with different prover sets for each, we can also use different MPC schemes for each substatement. This is useful in settings where different subsets of provers put different security requirements than others. By splitting the proof along these subsets, one could for example use a less efficient, dishonest majority MPC scheme where needed and a more efficient honest majority MPC scheme where possible.
- Similarly, one can use different NIZK schemes for different substatements. This can improve efficiency, as some specialized NIZK schemes are more efficient at proving certain statements than other generic schemes. Moreover, we can improve the security guarantees by using different NIZK schemes. For example, one could avoid schemes with trusted setups for critical substatements, and only use those with a trusted setup for less critical statements.
- Finally, in practice, many substatements of a larger statement will only require witnesses held by a single party. These substatements could be extracted and proved using a regular CP-NIZK, that does not suffer from the overhead caused by using MPC.

[1] As a matter of fact, the prover sets need not have any overlap at all.

C. Practical PA-MPC from collaborative CP-NIZKs

PA-MPC [6] extends MPC with a publicly-verifiable proof attesting to correct computation of the output. This correctness holds with respect to the public commitments to each party's inputs. There exist several ways of instantiating this method for verifiable privacy-preserving computation (VPPC). However, most practical schemes are based on NIZKs. Especially those based on succinct NIZKs are highly practical, due to their small communication and verification costs. For an overview and discussion of PA-MPC schemes we refer to [15].

Ozdemir and Boneh [51] discuss how to construct PA-MPC from collaborative proofs. They use collaborative proofs to construct a proof for the MPC computation result, whilst keeping the witness secret. Since each party commits to their inputs as part of the zero-knowledge proof, their construction also requires the opening of a commitment inside the proof.

For general-purpose NIZKs this is possible, however does lead to a significant increase in the number of constraints needed to encode the statement, thus leading to, e.g., increased proof generation times. Especially for an MPC prover, this increase quickly becomes a bottleneck. We observe that dedicated CP-NIZK schemes are notably more efficient at proving statements including commitment openings.

Thus, the adoption of collaborative CP-NIZKs likely leads to significant performance improvements for PA-MPC. Even more so, when taking the techniques of Section IV-B into account. For formal definitions and security guarantees of PA-MPC from collaborative proofs we refer to [51], [15].

V. Distributed Protocols for CP-NIZKs

In this section, we describe our methods for constructing MPC protocols for two specific NIZKs: *LegoGro16* and *Bulletproofs*. Both can be used to construct CP-NIZKs as we will show. We begin by describing two techniques that we use to construct the collaborative CP-NIZKs: collaborative commitment and $\mathsf{CP}_{\mathsf{link}}$. Our techniques are modular, and we strongly believe that they are more generally applicable to any NIZK that works over Pedersen-like commitments.

A. Collaborative Pedersen-like commitments

Overview of Pedersen-like commitments. Most CP-NIZKs, such as those introduced in LegoSNARK [17] or based on Bulletproofs [16] rely upon Pedersen-like commitments. These are commitment schemes whose verification algorithm follows the structure of the Pedersen vector commitment scheme [55]. We recall the Pedersen vector commitment scheme, for message vectors of length n:

Definition 4 (Pedersen vector commitment)**.** *Given a group $\mathbb{G}$ of order p, define input space $\mathcal{D} = \mathbb{Z}_p^n$, commitment space $\mathcal{C} = \mathbb{G}$, and opening space $\mathcal{O} = \mathbb{Z}_p$.*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck} = (g_0, \dots g_n) \leftarrow\$ \mathbb{G}^{n+1}$;
- $\mathsf{Commit}(\mathsf{ck}, u, o) \to c = g_0^o \cdot \prod_{i=1}^n g_i^{u_i}$, *with* $o \leftarrow\$ \mathbb{Z}_p$;
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \to c \stackrel{?}{=} g_0^o \cdot \prod_{i=1}^n g_i^{u_i}$.

The above scheme is perfectly hiding and computationally binding, given that the discrete log problem is hard in $\mathbb{G}$. We note that it is possible to open a Pedersen commitment within a non-CP-NIZK for arithmetic circuits, as done in, e.g., [42], [14]. However, this is significantly less efficient than in CP-NIZKs [17].

Our Protocol. One of the main distinguishing components of the CP-NIZK prover is generating a commitment to the witness. The collaborative CP-NIZKs we consider operate over data committed to with a Pedersen vector commitment and the private witness is distributed. Below, we present an MPC protocol that extends $\mathsf{Commit}(\mathsf{ck}, u, o)$ to compute a single commitment from a shared witness vector $u = (u_1, \dots, u_n)$, for simplicity we assume that party i knows u_i, and thus $n = N$. In practice, the witness vector may hold more elements than the number of parties, and each party could hold an arbitrary number of these witnesses. The methods we present are trivially extended to that case.

Intuitively, since Pedersen commitments are additively homomorphic [55], i.e., the product of two commitments is a commitment to the sum of the committed vectors, there are two ways the parties can collaboratively commit to the witness vector. We will refer to the approaches as *Commit-then-Share* and *Share-then-Commit*.

Commit-then-Share (CtS). In the CtS approach, each party commits to their witness element u_i before sharing. To keep the commitments hidden, each party must add a blinding factor $g_0^{o_i}$ to the commitment as follows:

$$c_i = g_0^{o_i} \cdot g_i^{u_i}$$

Then, each party sends their commitment to the other party where the commitments are multiplied to obtain a single, final commitment to the full witness vector u: $c = g_0^{o'} \cdot \prod_{i=1}^n g_i^{u_i}$, where $o' = \sum_{i=1}^n o_i$. As can be observed, the size of the commitment key is one group element per witness vector element, plus one common group element g_0 for the blinding factor, i.e., $n+1$ group elements in total.

Share-then-Commit (StC). The StC approach involves sharing and distributing the witness vector u among the parties before commitment. Initially, the witness vector, including a blinding factor, is split into shares and distributed among the parties. Then, each party commits to their shared version of the witness vector. Because the Pedersen commitment process is a linear operation, it works under the additive secret-sharing scheme. Concretely party i computes its share as:

$$[\![c]\!]_i = g_0^{[\![o]\!]_i} \prod_{j=1}^n g_j^{[\![u_j]\!]_i}.$$

Just like for CtS, the commitment key consists of $n+1$ group elements. Finally, the parties reveal the final full commitment by exchanging and multiplying the commitment shares: $c = \prod_{i=1}^n [\![c]\!]_i$.

B. Collaborative CP_{link}

In this section, we demonstrate the technique used to link the input commitment in CP-NIZKs, whose verification algorithm is the same as the Pedersen commitment, to an external commitment. This approach allows us to convert a commit-carrying NIZK (cc-NIZK) into a CP-NIZKs. To achieve this we rely on the $\mathsf{CP}_{\mathsf{link}}$ construct from [17]. $\mathsf{CP}_{\mathsf{link}}$ provides a way to prove that two commitments, with different keys, open to the same vector u (witness). Essentially, in this work, we use this technique to prove that the committed witnesses in two CP-NIZKs (LegoGro16 and Bulletproof) are equal.

$\mathsf{CP}_{\mathsf{link}}$ is built from a zk-SNARK for linear subspaces Π_{ss}, which essentially provides a way to prove the relation $R_M(\mathbf{x}, \mathbf{w})$:

$$R_M(\mathbf{x}, \mathbf{w}) = 1 \iff \mathbf{x} = \mathbf{M} \cdot \mathbf{w} \in \mathbb{G}_1^l,$$

where $\mathbf{M} \in \mathbb{G}_1^{l \times t}$ is a public matrix, $\mathbf{x} \in \mathbb{G}_1^l$ a public vector, and $\mathbf{w} \in \mathbb{Z}_p^t$ a witness vector. The following are the key algorithms for the zk-SNARK for linear subspaces Π_{ss}:

- $\Pi_{ss}.\mathsf{KeyGen}(\mathbf{M}) \to (\mathbf{ek}, \mathbf{vk})$:
  $\mathbf{k} \stackrel{\$}{\leftarrow} \mathbb{Z}_p^l$, $a \stackrel{\$}{\leftarrow} \mathbb{Z}_p$; $\mathbf{P} \coloneqq \mathbf{M}^\top \cdot \mathbf{k}$; $\mathbf{C} \coloneqq a \cdot \mathbf{k}$
  return $(\mathbf{ek} \coloneqq \mathbf{P} \in \mathbb{G}_1^l, \mathbf{vk} \coloneqq (\mathbf{C}' = g_2^{\mathbf{C}}, a' = g_2^a) \in \mathbb{G}_2^l \times \mathbb{G}_2)$
- $\Pi_{ss}.\mathsf{Prove}(\mathbf{ek}, \mathbf{w}) \to \pi$:
  return $\pi \leftarrow \mathbf{w}^\top \mathbf{P} \in \mathbb{G}_1$
- $\Pi_{ss}.\mathsf{VerProof}(\mathbf{vk}, \mathbf{x}, \pi) \to \{0, 1\}$:
  check that $\mathbf{x}^\top \cdot \mathbf{C}' \stackrel{?}{=} \pi \cdot a'$

Using Π_{ss}, we list the key algorithms for $\mathsf{CP}_{\mathsf{link}}$, which are simplified below for a setting with two commitments c and c', commitment keys ck and ck', and vector u:

- $\mathsf{CP}_{\mathsf{link}}.\mathsf{KeyGen}(\mathbf{ck}, \mathbf{ck}') \to (\mathbf{ek}, \mathbf{vk})$: constructs the matrix $\mathbf{M} \leftarrow [g_0, 0, g_1, \dots, g_n; 0, g_0', g_1', \dots, g_n]$ for the relation R_M using $\mathbf{ck} = (g_0, \dots, g_n) \in \mathbb{G}^{n+1}$ and $\mathbf{ck}' = (g_0', \dots, g_n') \in \mathbb{G}^{n+1}$ and generates evaluation and verification keys $(ek, vk) \leftarrow \Pi_{ss}.\mathsf{KeyGen}(\mathbf{M})$.
- $\mathsf{CP}_{\mathsf{link}}.\mathsf{Prove}(\mathbf{ek}, o, o', \mathbf{u}) \to \pi$: computes the proof $\pi \leftarrow \Pi_{ss}.\mathsf{Prove}(\mathbf{ek}, \mathbf{w})$ by using the evaluation key $\mathbf{ek}$, and setting $\mathbf{w} \leftarrow [o, o', u_1, \dots, u_n]$ where $\mathbf{u} = (u_1, \dots, u_n) \in \mathbb{Z}_p^n$.
- $\mathsf{CP}_{\mathsf{link}}.\mathsf{Verify}(\mathbf{vk}, c, c', \pi) \to \{0, 1\}$: verifies the proof by setting the vector $\mathbf{x} \leftarrow [c, c']$ in the subspace relation R_M, and running the verification algorithm $\{0, 1\} \leftarrow \Pi_{ss}.\mathsf{Verify}(\mathbf{vk}, \mathbf{x}, \pi)$

The $\mathsf{CP}_{\mathsf{link}}$ algorithms can be extended to link multiple commitments as shown in [17]. Our goal is to compute the prove function ($\Pi_{ss}.\mathsf{Prove}$) using MPC, where each party i

has a share $[\![\mathbf{w}]\!]_i$. We observe that the proof is essentially a multiplication of matrix $\mathbf{P}$ by the witness vector $\mathbf{w}$. Thus using additive secret sharing based MPC, a simple approach is to run $\mathsf{CP}_{\text{link}}.\mathsf{Prove}$ on each share of the witness, and then open the proof π. We also observe that Π_{ss} is essentially a zk-SNARKs that works on bilinear groups, therefore, we can apply the same optimization proposed in [51] to Π_{ss}. This allows us to combine $\mathsf{CP}_{\text{link}}$ with collaborative LegoGro16 and bulletproof efficiently, whilst also making them CP-NIZKs.

C. Collaborative LegoGro16

Overview of LegoGro16. This CP-NIZK is first presented in [17], and is a commit-and-prove version of *Groth16* [40], a frequently used zk-SNARK. Its proof consists of a regular *Groth16* proof (which contains 2 elements from $\mathbb{G}_1$ and one from $\mathbb{G}_2$), plus one additional element D (from $\mathbb{G}_1$) that contains a commitment to the input. $\mathsf{CP}_{\text{link}}$ guarantees that the element D is a commitment to the same value as the external commitment c.

Proving a general statement about a commitment opening using *LegoGro16* is around 5000× faster than using *Groth16*, where the commitment is opened inside a regular Groth16 circuit. Moreover, the CRS is approximately 7000× shorter. These advantages come at a very small cost. Namely, the total proof size is slightly larger (191B versus 127B) and the verification time is around 1.2× slower. These performance results further emphasize the potential of collaborative CP-NIZKs for realizing efficient PA-MPC.

Our Protocol. To realize the collaborative *LegoGro16* protocol, we essentially combine the collaborative *Groth16* protocol from [51] and the previous two protocols. The resulting protocol involves the following steps:

Setup. In the Setup phase, the CRSs for the *Groth16* and $\mathsf{CP}_{\text{link}}$ protocols are generated. The resulting keys are sent to all parties.

Commit and Distribute. The input for the Commit algorithm is the commitment key ck, and a vector of field elements (Witnesses). Depending on whether the CtS or StC approach (see Section V-A) is used, the input vector will be either one party's part of the plain witness input or the shared witness input of all parties. The output is a commitment to the witness input. Then, in the $\mathsf{DistributeWitness}$ step of the protocol, all parties generate shares of their part of the witness and distribute the shares to all other parties.

Prove. The input for the collaborative prove algorithm Π is the proving key pk, the circuit's public input, the shared witness input, $\mathsf{CP}_{\text{link}}$'s evaluation key $\mathbf{ek}$, and the opening to the external commitment. This algorithm executes the MPC protocol to generate a proof share for each party. The proof contains the standard *Groth16* with the additional element D and the $\mathsf{CP}_{\text{link}}$ proof.

RevealProof. The input for the $\mathsf{RevealProof}$ algorithm is a vector of all proof shares generated in the proving phase. By exchanging the proof shares, the parties can reconstruct the full proof.

D. Collaborative Bulletproofs for arbitrary arithmetic circuits

In what follows, we introduce collaborative Bulletproofs, starting with an overview of regular Bulletproofs [16], followed by our collaborative version of Bulletproofs for arbitrary arithmetic circuits. As explained in Section V-B, this is then easily transformed into a CP-NIZK using $\mathsf{CP}_{\text{link}}$. This does however warrant a minor modification to the verification algorithm (see Appendix C).

Overview of Bulletproofs. Bulletproofs have several advantages over other proof systems, including being based only on standard assumptions, not requiring bilinear maps, and having linear prover time complexity and proof size logarithmic in the number of constraints. Additionally, it allows building constraint systems on the fly, without a trusted setup. Bulletproofs are constructed using inner product arguments and employ a recursive approach for efficiency. This efficiency makes it feasible to use bulletproof in blockchains and confidential transactions.

Bulletproofs provide an efficient zero-knowledge argument for arbitrary arithmetic circuits, whilst also generalizing to include committed values (witnesses) as inputs to the arithmetic circuit. Including committed input wires is important as it makes Bulletproofs naturally suitable for CP-NIZKs without the need to implement an in-circuit commitment algorithm, thus aligning with our definitions.

Our Protocol. We describe here how we constructed the collaborative bulletproof following the notation from [16], for ease of comparison. The prover begins by committing to its secret inputs $\mathbf{v} \in \mathbb{Z}_p^m$ using a blinding factor $\gamma \in \mathbb{Z}_p^m$ and generating a Pedersen commitment $\mathbf{V}$ where:

$$V_j = g^{v_j} h^{\gamma_j} \quad \forall j \in [1, m]$$

In our setting, $\mathbf{v}$ is distributed among the N provers. Each prover $\mathcal{P}_i$ has a share $[\![\mathbf{v}]\!]_i$ of the witness. Therefore, all provers collaboratively generate this commitment using either the CtS or the StC protocols as described earlier.

Subsequently, the provers build the constraint system, which allows them to perform a combination of operations to generate the constraints. Specifically, the rank-1 constraint system (R1CS) is used, which consists of two sets of constraints: (1) multiplication gates and (2) linear constraints in terms of the input variables.

(1) Multiplication gates simply take two input variables and multiply them to get an output. All multiplication gates in the constraint system can be expressed by the relation:

$$\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$$

Where $\mathbf{a}_L \in \mathbb{Z}_p^n$ is the vector of the first input to each gate, $\mathbf{a}_R \in \mathbb{Z}_p^n$ is the vector of the second input to each gate, and $\mathbf{a}_O \in \mathbb{Z}_p^n$ is the vector of multiplication results. All three vectors have size n representing the number of multiplication gates.

(2) the input variables are used to express Q Linear constraints in the form:

$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c}$$

Where $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}$ are the public constraints matrices representing the weights applied to the respective inputs and outputs. $\mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}$ is the matrix representing the weights for a commitment $\mathbf{V}$ and $\mathbf{c} \in \mathbb{Z}_p^Q$ is a constant public vector used in the linear constraints.

The provers' goal in collaborative bulletproofs is to jointly prove that there exists a $\mathbf{v}$ (in commitment $\mathbf{V}$) such that the linear constraints are satisfied while maintaining the secrecy of their respective $[\![\mathbf{v}]\!]_i$. The proof system can be summarized in the following relation.:

$$\left\{ \begin{array}{l} \left(\begin{array}{l} \mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{V} \in \mathbb{G}^m, g, h \in \mathbb{G}, \\ \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}, \\ \mathbf{c} \in \mathbb{Z}_p^Q; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n, \mathbf{v}, \gamma \in \mathbb{Z}_p^m \end{array} \right) : \\ V_j = g^{v_j} h^{\gamma_j} \forall j \in [1, m] \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \\ \wedge \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} \end{array} \right\}$$

In addition to proving the previous relation, the provers utilize $\mathsf{CP}_{\text{link}}$ to prove that $\mathbf{V}$ commits to the same witnesses $\mathbf{v}$ as an external commitment $\hat{V}$.

Specification. In Figure 1, we summarize the protocol for proving this relation collaboratively with N provers. The protocol relies on Π_{DBP}, a sub-protocol for collaboratively generating proofs for arbitrary arithmetic circuits. Π_{DBP} is our distributed construction of the regular bulletproofs from [16]. Unlike in [16], we show Π_{DBP} in its non-interactive form in Figure 2. We use the cryptographic hash function denoted as $\mathbf{H}$ to hash the transcript up to that point including the statements to be proven $\mathbf{st}$. The prover's input to the collaborative Bulletproof protocol in Figure 1 includes those required in the standard protocol, along with $\mathbf{ek}$ the $\mathsf{CP}_{\text{link}}$ evaluation key and $[\![\hat{o}]\!]$ the party's share of opening to the external commitment.

Additionally, the proving system employs the inner product argument (IPA) protocol by expressing all the constraints in terms of a single inner product and then running the IPA protocol. This reduces the communication cost (proof size) since IPA has logarithmic communication complexity. The IPA is not zero-knowledge itself [16], nor does it need to be, and therefore can be executed individually by each party on the revealed vectors $(\mathbf{l}, \mathbf{r})$ as shown in step 11 of Figure 2. This approach does not require any communication between parties, however, all parties must check that the proof π_{IPA} sent to the verifier is consistent with the one they generated locally. An alternative but costly approach is to run the IPA protocol in a distributed fashion. This ensures that all parties generate the same proof π_{IPA}. We present a distributed IPA protocol in Appendix B, and experimentally compare its performance to the non-distributed version.

VI. Implementation

To evaluate and experiment with our newly developed collaborative CP-NIZKs, a proof-of-concept system was implemented that allows multiple parties to generate collaborative CP-NIZK proofs.

Since implementing collaborative CP-NIZK relies heavily on finite fields and pairing-friendly curves, we used a library that abstracts the underlying cryptographic operations and offers a generic interface for working with finite fields and pairing-based cryptography.

$\mathcal{P}_i$'s input: $(g, h \in \mathbb{G}, [\![\mathbf{v}]\!]_i \in \mathbb{Z}_p^m, \mathbf{ek} \in \mathbb{G}^l, [\![\hat{o}]\!]_i \in \mathbb{Z}_p)$
Output: $(\pi, \mathbf{V}, \pi_{\mathsf{CP}_{\text{link}}})$

1. **Generate input commitment:**
   a. $\mathcal{P}_i$: $[\![\gamma]\!]_i \xleftarrow{\$} \mathbb{Z}_p^m$
   b. Parties use CtS or StC to commit to input $\mathbf{v}$, each using their share of the input $[\![\mathbf{v}]\!]_i$ and blinding factors $[\![\gamma]\!]_i$, resulting in commitment $\mathbf{V} \in \mathbb{G}^m$:
   $$V_j = g^{v_j} h^{\gamma_j} \quad \forall j \in [1, m]$$
2. **Build constraint system:**
   a. Parties build the constraint systems for the required statements to be proven, generating a public description of the circuit $\mathbf{st} = (g, h, \mathbf{g}, \mathbf{h}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_V, \mathbf{c}, \mathbf{V})$
   b. Each party $\mathcal{P}_i$ assign the left and right inputs to each multiplication gate using their shares of $[\![\mathbf{v}]\!]_i$ and obtains $([\![\mathbf{a}_L]\!]_i, [\![\mathbf{a}_R]\!]_i)$.
   c. Parties use MPC to compute: $[\![\mathbf{a}_O]\!]_i = [\![\mathbf{a}_L]\!]_i \circ [\![\mathbf{a}_R]\!]_i$
3. **Prove:** Each $\mathcal{P}_i$ runs Π_{DBP} and obtains the proof π
   $$\pi \leftarrow \Pi_{DBP}(g, h, \mathbf{g}, \mathbf{h}, \mathbf{c}, [\![\mathbf{a}_L]\!]_i, [\![\mathbf{a}_R]\!]_i, [\![\mathbf{a}_O]\!]_i, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_V, [\![\gamma]\!]_i)$$
4. **Link:**
   a. Each $\mathcal{P}_i$ computes:
   $$[\![\gamma']\!]_i = \sum_{j=1}^{m} [\![\gamma_j]\!]_i$$
   $$[\![\pi_{\mathsf{CP}_{\text{link}}}]\!]_i \leftarrow \mathsf{CP}_{\text{link}}.\mathsf{Prove}(\mathbf{ek}, [\![\gamma']\!]_i, [\![\hat{o}]\!]_i, [\![\mathbf{v}]\!]_i)$$
   b. $\mathcal{P}$: open $\pi_{\mathsf{CP}_{\text{link}}}$
5. **Output:** $(\pi, \mathbf{V}, \pi_{\mathsf{CP}_{\text{link}}})$

Fig. 1. Collaborative bulletproof protocol for arbitrary arithmetic circuits

Our implementation[2] is built on top of Arkworks [4], a Rust ecosystem for zk-SNARK programming. It provides libraries (crates) for working with finite fields and elliptic curves and several implementations of existing zk-SNARKs and other cryptographic primitives for implementing custom NIZKs.

MPC. A number of prior works [51], [34], [21] also base their solutions on Arkworks. For instance, [51] implements collaborative zk-SNARKs using the Arkworks library, including the MPCs primitives. However, extending their implementation to support the CP-NIZKs considered in this work proved to be a difficult task. The implementation in [51] uses version v0.2.0 of Arkworks, and since Arkworks is in active development, various breaking changes have occurred since. As a result, we opted to implement the MPC primitives using the recent version (v0.4.x). Although the implementation is made from scratch, some design decisions from these existing implementations are used. Similar to [51], introducing an MPC Pairing Wrapper is our primary implementation strategy, in addition to defining interfaces for shared field and group types. However, we limited our MPC implementation to the SPDZ framework.

Networking. Our collaborative CP-NIZK protocol requires communication between all parties. We implemented the `network` module to manage this communication. There are

[2]Our open-source implementation will be made available

$\mathcal{P}_i$**'s input:**

$$\left(\begin{array}{c} g, h \in \mathbb{G}, \mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{c} \in \mathbb{Z}_p^Q, [\![\mathbf{a}_L]\!]_i, [\![\mathbf{a}_R]\!]_i, [\![\mathbf{a}_O]\!]_i \in \mathbb{Z}_p^n, \\ \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}, [\![\gamma]\!]_i \in \mathbb{Z}_p^m \end{array}\right)$$

Output: proof π

1. Each $\mathcal{P}_i$ computes:

$$\begin{aligned} &[\![\alpha]\!]_i, [\![\beta]\!]_i, [\![\rho]\!]_i \xleftarrow{\$} \mathbb{Z}_p \\ &[\![A_I]\!]_i = h^{[\![\alpha]\!]_i} \mathbf{g}^{[\![\mathbf{a}_L]\!]_i} \mathbf{h}^{[\![\mathbf{a}_R]\!]_i} \in \mathbb{G} \\ &[\![A_O]\!]_i = h^{[\![\beta]\!]_i} \mathbf{g}^{[\![\mathbf{a}_O]\!]_i} \in \mathbb{G} \\ &[\![\mathbf{s}_L]\!]_i, [\![\mathbf{s}_R]\!]_i \xleftarrow{\$} \mathbb{Z}_p^n \\ &[\![S]\!]_i = h^{[\![\rho]\!]_i} \mathbf{g}^{[\![\mathbf{s}_L]\!]_i} \mathbf{h}^{[\![\mathbf{s}_R]\!]_i} \in \mathbb{G} \end{aligned}$$

2. $\mathcal{P}$: open (A_I, A_O, S)
3. $\mathcal{P}$: $y \leftarrow \mathbf{H}(\mathbf{st}, A_I, A_O, S), z \leftarrow \mathbf{H}(A_I, A_O, S, y) \in \mathbb{Z}_p^*$
4. Each $\mathcal{P}_i$ computes:

$$\begin{aligned} &\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1}) \in \mathbb{Z}_p^n \\ &\mathbf{z}_{[1:]}^{Q+1} = (z, z^2, \ldots, z^Q) \in \mathbb{Z}_p^Q \\ &\delta(y,z) = \langle \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R), \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L \rangle \end{aligned}$$

5. Each $\mathcal{P}_i$ computes:

$$\begin{aligned} [\![l(X)]\!] &= [\![\mathbf{a}_L]\!] \cdot X + [\![\mathbf{a}_O]\!] \cdot X^2 \\ &\quad + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R) \cdot X \\ &\quad + [\![\mathbf{s}_L]\!] \cdot X^3 \in \mathbb{Z}_p^n[X] \\ [\![r(X)]\!] &= \mathbf{y}^n \circ [\![\mathbf{a}_R]\!] \cdot X - \mathbf{y}^n \\ &\quad + \mathbf{z}_{[1:]}^{Q+1} \cdot (\mathbf{W}_L \cdot X + \mathbf{W}_O) \\ &\quad + \mathbf{y}^n \circ [\![\mathbf{s}_R]\!] \cdot X^3 \in \mathbb{Z}_p^n[X] \\ [\![t(X)]\!]_i &= \langle [\![l(X)]\!]_i, [\![r(X)]\!]_i \rangle \\ &= \sum_{j=1}^{6} [\![t_j]\!]_i \cdot X^j \in \mathbb{Z}_p[X] \end{aligned}$$

$$\begin{aligned} &[\![\tau_j]\!]_i \xleftarrow{\$} \mathbb{Z}_p \quad \forall j \in [1,3,4,5,6] \\ &[\![T_j]\!]_i = g^{[\![t_j]\!]_i} \cdot h^{[\![\tau_j]\!]_i} \quad \forall j \in [1,3,4,5,6] \end{aligned}$$

6. $\mathcal{P}$: open $([\![T_j]\!]_i \quad \forall j \in [1,3,4,5,6])$
7. $\mathcal{P}$: $x \leftarrow \mathbf{H}(y, z, T_1, T_3, T_4, T_5, T_6) \in \mathbb{Z}_p^*$
8. Each $\mathcal{P}_i$ computes:

$$\begin{aligned} [\![\mathbf{l}]\!]_i &= [\![l(x)]\!]_i \in \mathbb{Z}_p^n \\ [\![\mathbf{r}]\!]_i &= [\![r(x)]\!]_i \in \mathbb{Z}_p^n \\ [\![\tau_x]\!]_i &= \sum_{j=1, j\neq 2}^{6} [\![\tau_j]\!]_i \cdot x^j \\ &\quad + x^2 \cdot \left(\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V \cdot [\![\gamma]\!]_i\right) \in \mathbb{Z}_p \\ [\![\mu]\!]_i &= [\![\alpha]\!]_i \cdot x + [\![\beta]\!]_i \cdot x^2 + [\![\rho]\!]_i \cdot x^3 \in \mathbb{Z}_p \end{aligned}$$

9. $\mathcal{P}$: open $(\mathbf{l}, \mathbf{r}, \tau_x, \mu)$
10. $\mathcal{P}$: $x_u \leftarrow \mathbf{H}(x, \tau_x, \mu) \in \mathbb{Z}_p^*$
11. Each $\mathcal{P}_i$ computes:

$$\begin{aligned} \hat{t} &= \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p \\ \pi_{IPA} &\leftarrow \Pi_{IPA}.\text{Prove}(\mathbf{g}, \mathbf{h}, g^{x_u}, \mathbf{l}, \mathbf{r}) \end{aligned}$$

12. **Output:**
$\pi = (A_I, A_O, S, T_1, T_3, T_4, T_5, T_6, \tau_x, \mu, \hat{t}, \pi_{IPA})$

Fig. 2. Sub-protocol Π_{DBP} for collaboratively generating bulletproofs for arbitrary arithmetic circuits

three types of communication between the parties: (1) exchanging witness shares; (2) communicating during the proving protocol to multiply two shared values; and (3) exchanging proof shares to get the final proof.

Collaborative LegoGro16. We based our LegoGro16 implementation on that of [17] and updated it to be compatible with our version of Arkworks. Additionally, we observed that the same MPC optimization applied to Groth16 in [51] can also be applied to its CP counterpart (LegoGro16), and therefore, these optimizations were implemented. The LegoGro16 implementation in this work does not support parallelization, and the existing multi-scalar multiplication implementation is replaced with a single-threaded alternative. Parallelization is disabled because the order in which the parties exchange and reveal values is essential. This problem could be solved by using a more complex communication protocol, but this is beyond the scope of this project. This decision is made similarly in prior works like [51].

Collaborative CP-Bulletproofs. Our starting point for the Bulletproof implementation is an existing Bulletproof implementation [30] that works over *Curve25519* and is implemented in Rust. We adapted this implementation to support distributed provers by modifying the underlying field and group elements to work over the MPC shared fields and group types we defined. Then, we applied the techniques described in Section V-D by building a wrapper for the prover functions to allow multiple parties to generate a proof. We use Merlin transcripts [29] to manage the generation of verifier challenges, ensuring that all provers have a consistent transcript to generate the correct challenge.

Summary. The main modules for the implementation can be summarized as follows:

- The `mpc` module contains all the functionality for implementing the MPC primitives on Arkworks;
- The `network` module implements all the networking and communication functionalities required for the parties to communicate with each other;
- The `col-LegoGro16` module contains the implementation for the collaborative LegoGro16;
- The `col-cp-BP` module contains the implementation for the collaborative CP-Bulletproofs.

VII. Experiments

Below, we present the main results of our extensive experimentation for evaluating the performance of our collaborative CP-NIZKs constructions. Additional experimental results are provided in Appendix D. Our evaluation was conducted on a consumer machine equipped with a 10-core Apple M2 Pro CPU and 16GB of RAM. We focused on two metrics: single-threaded runtime and communication cost. We report the average results from three runs. We vary the following parameters to examine their impact on performance:

- The number of R1CS constraints, varying from 2 to 2^{15};
- The number of parties, increasing from 2 to 2^6.

With our experiments we answer the following questions:

1) *How does the performance of both collaborative CP-NIZKs (LegoGro16 and Bulletproofs) compare to the single prover setting?*

2) *What is the overhead of composability (linking input commitments), i.e., how does collaborative CP-NIZKs compare to the non-CP variant such the ones in [51]?*
3) *Does the performance of using two different CP-NIZKs improves efficiency over using a single NIZK?*

A. Setup and commitments

The focus of this work is on distributing the prover in CP-NIZKs and evaluating the benefits of composability in this setting. Therefore, we omit the evaluation of the setup required for LegoGro16 to generate the CRS and any computations necessary to prepare the witness and public input to the circuit. This decision is common for this line of work, since these computations heavily depend on the type of circuit used and can be performed when spare computational resources and bandwidth are available. Additionally, we omit evaluating the verifier cost since we do not modify the verification algorithms and these costs are thus unchanged.

An essential part of the setup in our construction is that provers must collaboratively generate a commitment to the witness, which can then be linked to an external commitment using $\mathsf{CP}_{\text{link}}$. Figure 6 shows the performance of generating the commitment collaboratively using both sub-protocols: CtS and StC. The performance is measured with an increasing number of provers N, where we assume each prover has a distinct witness. The communication cost for both sub-protocols is the same, consisting of $N-1$ broadcast messages per prover. However, the total runtime for the CtS method is significantly less than the StC method, and this difference increases as the number of provers increases. This can be explained by the fact that the CtS method only requires each party to commit to their part of the witness, while the StC method requires each party to commit to the shared witness, a vector of N elements.

B. Varying number of constraints

To demonstrate the scalability of collaborative CP-NIZKs, we evaluated both LegoGro16 and Bulletproof under varying numbers of constraints. The results are depicted in Figures 3 and 4, along with the performance of a single prover and non-collaborative variants of these NIZKs. The results show that even in distributed settings collaborative CP-NIZKs overhead is minimal and almost negligible in circuits with a large number of constraints, especially when compared to non-CP counterparts, such as the ones in [51].

C. Varying number of provers

In our second experiment, we fixed the number of constraints at 2^{10} and varied the number of provers to show how collaborative CP-NIZKs scale with an increasing number of provers. We show the results in Figure 5. Our results are consistent with that in [51] for Groth16. For bulletproofs the increase in the number of provers causes a significant decrease in performance. This is mainly because bulletproofs requires more communication compared to Groth16 as will be shown in Section VII-D.

As can be observed from our results, the performance of running two instances of collaborative CP-NIZKs with N provers compared to running one instance with $2N$ provers would result in 1.3–$2\times$ improved performance, highlighting the importance of composition. Splitting prover groups becomes beneficial in settings with a large number of provers (>8 provers). For instance, splitting 64 provers into 2 collaborative CP-NIZKs would improve the per-party runtime by $\approx 2\times$.

D. Communication Cost

In Figure 7, we report the communication cost per party for both collaborative CP-NIZKs (LegoGro16 and bulletproof). The results show that bulletproof requires more communication than LegoGro16.

E. Improving efficiency by composition

In our final experiment, we aim to answer our third question: whether using different collaborative CP-NIZKs would improve performance (for provers) compared to simply using a single general-purpose NIZK, such as those used in [51]. To answer this question, we first examined when using collaborative Bulletproofs or LegoGro16 would be less costly. For this, we plotted the runtime performance overhead of both protocols as we varied the number of constraints. Figure 8 shows our results. Based on these results, it can be observed that it is optimal to use LegoGro16 for large circuits (with a large number of constraints) and Bulletproofs for smaller circuits. Therefore, we conclude that it is indeed more efficient to use a combination of these two CP-NIZKs on the same input than to rely solely on a single one. In Section VIII we further support this conclusion by evaluating the performance improvement as a result of proof composition by means of an application scenario.

VIII. Application: private audits

As we discussed earlier in Section I-A, collaborative CP-NIZKs can be utilized to generate proofs in a setting where a mortgage applicant needs to demonstrate compliance with the bank's requirements. This setting is analogous to the proofs about net assets presented in [51]. Below, we discuss how collaborative CP-NIZKs can be constructed in this setting, evaluate the time required to construct the proof, and show how we significantly improve efficiency over the Collaborative SNARKs approach in [51].

Consider k transactions distributed among N banks, where each transaction is a 64-bit signed integer. Each bank publishes a Merkle tree commitment of all transactions. The applicant is required to prove that the net of their debits and credits across all banks exceeds a threshold T. This claim can be split into the following checks:

1. The transactions are in the committed Merkle tree.
2. The transactions are well-formed (represented as a 64-bit signed integer).
3. The sum of all transactions is computed correctly.
4. The computed sum is larger than the threshold T.

A straightforward approach is to include all checks in one collaborative zk-SNARK, requiring around $364 \cdot n$ as demonstrated in [51]. However, since the claim can be split into multiple checks, we can utilize the modularity of our proposed collaborative CP-NIZKs and use a combination of both collaborative Groth16 and Bulletproofs to gain efficiency.

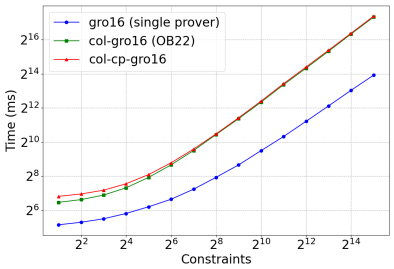

Fig. 3. Runtime per prover party for bulletproofs: (1) Collaborative with $\mathsf{CP}_{\text{link}}$ (col-cp-bp), (2) Collaborative (col-bp), (3) Single prover (bp).

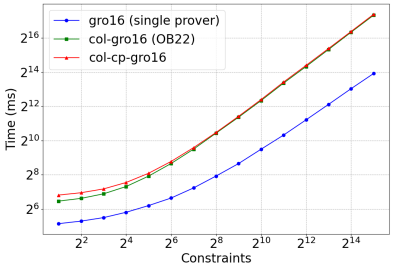

Fig. 4. Runtime per prover party for: (1) Collaborative LegoGro16 (col-cp-gro16), (2) Collaborative Groth16 (col-gro16), (3) Single prover Groth16 (gro16).

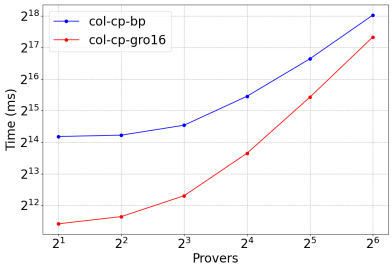

Fig. 5. Runtime per prover party for varying number of constraints and prover group sizes.

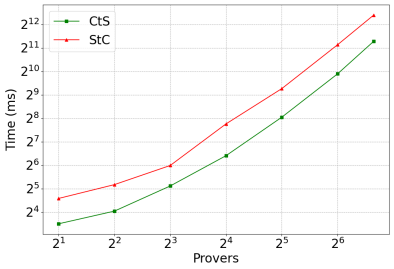

Fig. 6. Runtime per prover party for CtS and StC where each party provides a witness to the commitment.

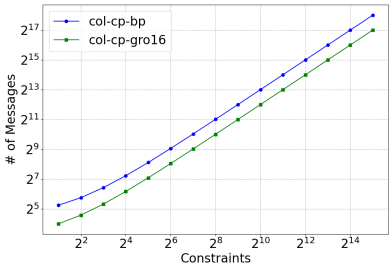

Fig. 7. Collaborative LegoGro16 and bulletproofs communication costs for varying circuit sizes. The number of provers is 2.

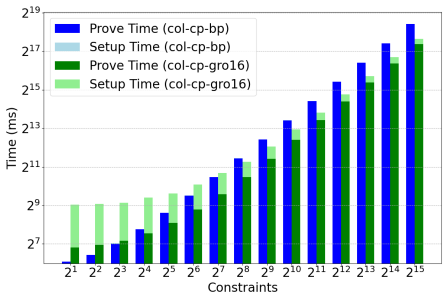

Fig. 8. Setup and prove runtime per prover party for Collaborative LegoGro16 (col-cp-gro16) and Collaborative bulletproofs with $\mathsf{CP}_{\text{link}}$ (col-cp-bp). The number of provers is 2.

In this setting, checking that each transaction is well-formed and is included in the Merkle tree can be done individually by each bank, as each bank can generate such proofs for their set of transactions. Then, computing the sum and checking it against the threshold T can be done collaboratively using our construction of collaborative CP-NIZKs, which would link this collaborative proof with the individually generated proofs. We exploit this modularity even further by employing Groth16 for large circuits such as checks 1 and 4, and Bulletproofs for checks 2 and 3.

For evaluating the two approaches (on our consumer machine) we set the number of transactions to 90 ($\approx 2^{15}$ constraints) and the number of banks to 2. Generating a proof would take ≈ 165 seconds and require $\approx 2^{17}$ messages exchanged using the approach in [51]. Using our construction, the two parties each with 45 transactions will run 2^{14} constraints on their set of transactions locally (without MPC) using legoGro16 in ≈ 8.3 seconds. Then the two banks will collaboratively run CP-NIZKs to compute the sum and compare to T requiring $64 + log_2 k \approx 71$ constraints in ≈ 0.5 seconds. Thus, generating the proof using our construction is done in ≈ 9 seconds and requires $\approx 2^8$ messages exchanged. This results in an $18\times$ improvement in per-party runtime and only 0.2% of the required communication. We note that if the number of transactions is fixed at 90 but the number of parties (banks) increases, the per-party runtime decreases since each party will locally prove k/N transactions ($2^{15}/N$ constraints). We evaluated this for 4 and 8 banks, finding that the improvement in runtime becomes $33\times$ and $55\times$, respectively. Overall, our protocol is 18–$55\times$ faster in this application setting compared to [51] with only a fraction of the required communication between parties. We estimate that this improvement in performance increases with a higher number of transactions (constraints). Processing 2^{20} constraints using [51] approach would be infeasible on a consumer machine, taking ≈ 400 minutes. In contrast, using our approach the time taken would be ≈ 9.5 minutes. Therefore, we project that for 2^{20} constraints, the performance enhancement would be in the range 40–$200\times$.

IX. Conclusion

We have formally defined collaborative CP-NIZKs and showed how to generically construct these from existing NIZKs and MPCs frameworks, by taking advantage of the modularity of our construction. Moreover, we present two commitment paradigms that allow the adoption of CP-NIZKs in varying settings, both adaptive and on-the-fly.

By combining the strengths of collaborative proofs with the composability of the commit-and-prove paradigm, we achieve significant efficiency improvements over non-composable counterparts while maintaining the flexibility of collaborative proofs. Our implementation in Arkworks demonstrates the practical feasibility and modularity of our approach, paving the way for future work and extensions to easily integrate novel NIZKs and MPCs schemes. Our experiments show that our approach incurs minimal overhead and provides substantial performance benefits, including efficiency improvements of 18–$55\times$ compared to existing constructions in realistic application scenarios. This makes collaborative CP-NIZKs a valuable tool for a wide range of applications.

REFERENCES

[1] S. Agrawal, C. Ganesh, and P. Mohassel, "Non-Interactive Zero-Knowledge Proofs for Composite Statements," in *Advances in Cryptology – CRYPTO 2018*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 643–673.

[2] R. Alvarez and M. Nojoumian, "Comprehensive survey on privacy-preserving protocols for sealed-bid auctions," *Computers & Security*, vol. 88, p. 101502, Jan. 2020.

[3] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 2087–2104.

[4] arkworks contributors, "arkworks zksnark ecosystem," arkworks, 2022. [Online]. Available: https://arkworks.rs

[5] T. Attema and R. Cramer, "Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics," in *Advances in Cryptology – CRYPTO 2020*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 513–543.

[6] C. Baum, I. Damgård, and C. Orlandi, "Publicly Auditable Secure Multi-Party Computation," in *Security and Cryptography for Networks*, ser. Lecture Notes in Computer Science, M. Abdalla and R. De Prisco, Eds. Cham: Springer International Publishing, 2014, pp. 175–196.

[7] C. Baum, R. Jadoul, E. Orsini, P. Scholl, and N. P. Smart, "Feta: Efficient Threshold Designated-Verifier Zero-Knowledge Proofs," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 293–306.

[8] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," in *Advances in Cryptology — CRYPTO '91*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer, 1992, pp. 420–432.

[9] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE, May 2015, pp. 287–304.

[10] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent Succinct Arguments for R1CS," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 103–128.

[11] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive Oracle Proofs," in *Theory of Cryptography*, M. Hirt and A. Smith, Eds. Berlin, Heidelberg: Springer, 2016, pp. 31–60.

[12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, Jan. 2012, pp. 326–349. [Online]. Available: https://dl.acm.org/doi/10.1145/2090236.2090263

[13] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ser. STOC '88. New York, NY, USA: Association for Computing Machinery, Jan. 1988, pp. 103–112. [Online]. Available: https://dl.acm.org/doi/10.1145/62212.62222

[14] T. Bontekoe, M. Everts, and A. Peter, "Balancing privacy and accountability in digital payment methods using zk-SNARKs," in *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*. Fredericton, NB, Canada: IEEE, Aug. 2022, pp. 1–10.

[15] T. Bontekoe, D. Karastoyanova, and F. Turkmen, "Verifiable Privacy-Preserving Computing," Apr. 2024.

[16] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," in *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2018, pp. 315–334.

[17] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2075–2092.

[18] M. Chase, C. Ganesh, and P. Mohassel, "Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials," in *Advances in Cryptology – CRYPTO 2016*, M. Robshaw and J. Katz, Eds. Berlin, Heidelberg: Springer, 2016, pp. 499–530.

[19] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections," *IEEE Security & Privacy*, vol. 2, no. 1, pp. 38–47, Jan. 2004.

[20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward, "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS," 2019, publication info: A major revision of an IACR publication in EUROCRYPT 2020. [Online]. Available: https://eprint.iacr.org/2019/1047

[21] A. Chiesa, R. Lehmkuhl, P. Mishra, and Y. Zhang, "Eos: Efficient Private Delegation of zkSNARK Provers," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA, USA: USENIX Association, 2023, pp. 6453–6469. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa

[22] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and Transparent Recursive Proofs from Holography," in *Advances in Cryptology – EUROCRYPT 2020*, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 769–793.

[23] J. D. Cohen and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme," in *26th Annual Symposium on Foundations of Computer Science (Sfcs 1985)*. Portland, OR, USA: IEEE, Oct. 1985, pp. 372–382.

[24] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, Jan. 2012, pp. 90–112.

[25] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile Verifiable Computation," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE, May 2015, pp. 253–270.

[26] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer, 2012, pp. 643–662.

[27] P. Dayama, A. Patra, P. Paul, N. Singh, and D. Vinayagamurthy, "How to prove any NP statement jointly? Efficient Distributed-prover Zero-Knowledge Protocols," 2021. [Online]. Available: https://eprint.iacr.org/2021/1599

[28] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai, "Robust Non-interactive Zero Knowledge," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer, 2001, pp. 566–598.

[29] H. de Valence, "Merlin," https://github.com/dalek-cryptography/merlin, 2024, accessed: 2024-05-17.

[30] H. de Valence, C. Yun, and O. Andreev, "Bulletproofs," https://github.com/dalek-cryptography/bulletproofs, 2024, accessed: 2024-03-17.

[31] M. Dutta, C. Ganesh, S. Patranabis, and N. Singh, "Compute, but Verify: Efficient Multiparty Computation over Authenticated Inputs," 2022, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2022/1648

[32] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Advances in Cryptology — CRYPTO' 86*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer, 1987, pp. 186–194.

[33] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge," 2019, publication info: Preprint. [Online]. Available: https://eprint.iacr.org/2019/953

[34] S. Garg, A. Goel, A. Jain, G.-V. Policharla, and S. Sekar, "zkSaaS: Zero-Knowledge SNARKs as a Service," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA, USA:

USENIX Association, 2023, pp. 4427–4444. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/garg

[35] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge: Cambridge University Press, 2004. [Online]. Available: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html

[36] S. Goldwasser and Y. Kalai, "On the (In)security of the Fiat-Shamir paradigm," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* Cambridge, MA, USA: IEEE, Oct. 2003, pp. 102–113.

[37] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, Feb. 1989.

[38] J. Groth, "Non-interactive Zero-Knowledge Arguments for Voting," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Berlin, Heidelberg: Springer, 2005, pp. 467–482.

[39] ——, "Short Pairing-Based Non-interactive Zero-Knowledge Arguments," in *Advances in Cryptology - ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, M. Abe, Ed. Berlin, Heidelberg: Springer, 2010, pp. 321–340.

[40] ——, "On the Size of Pairing-Based Non-interactive Arguments," in *Advances in Cryptology – EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer, 2016, pp. 305–326.

[41] J. Y. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM*, vol. 37, no. 3, pp. 549–587, Jul. 1990.

[42] D. E. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash Protocol Specification, Version 2023.4.0 [NU5]," Dec. 2023, Protocol specification. [Online]. Available: https://zips.z.cash/protocol/protocol.pdf

[43] K. Jurek, "What is a zero-knowledge proof?" Oct. 2023. [Online]. Available: https://www.aleo.org/post/what-is-a-zero-knowledge-proof/

[44] S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller, "Publicly Auditable MPC-as-a-Service with succinct verification and universal setup," Jul. 2021, arXiv:2107.04248 [cs]. [Online]. Available: http://arxiv.org/abs/2107.04248

[45] H. Kikuchi, M. Hakavy, and D. Tygar, "Multi-round anonymous auction protocols," *IEICE Transactions on Information and Systems*, vol. 82, no. 4, pp. 769–777, 1999.

[46] M. Kohlweiss, M. Maller, J. Siim, and M. Volkhov, "Snarky Ceremonies," in *Advances in Cryptology – ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds. Cham: Springer International Publishing, 2021, pp. 98–127.

[47] J. Lee, J. Choi, J. Kim, and H. Oh, "SAVER: SNARK-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization," 2019, publication info: Preprint. MINOR revision. [Online]. Available: https://eprint.iacr.org/2019/1270

[48] Y. Lindell, "Secure Multiparty Computation (MPC)," 2020. [Online]. Available: https://eprint.iacr.org/2020/300

[49] H. Lipmaa, "Prover-Efficient Commit-and-Prove Zero-Knowledge SNARKs," in *Progress in Cryptology – AFRICACRYPT 2016*, D. Pointcheval, A. Nitaj, and T. Rachidi, Eds. Cham: Springer International Publishing, 2016, pp. 185–206.

[50] B. Marrika, van, "ING launches Zero-Knowledge Range Proof solution, a major addition to blockchain technology," Nov. 2017. [Online]. Available: https://www.ingwb.com/en/insights/distributed-ledger-technology/ing-launches-major-addition-to-blockchain-technology

[51] A. Ozdemir and D. Boneh, "Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 4291–4308. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir

[52] S. Panja and B. Roy, "A secure end-to-end verifiable e-voting system using blockchain and cloud server," *Journal of Information Security and Applications*, vol. 59, p. 102815, Jun. 2021.

[53] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2856449

[54] T. P. Pedersen, "Distributed Provers with Applications to Undeniable Signatures," in *Advances in Cryptology — EUROCRYPT '91*, D. W. Davies, Ed. Berlin, Heidelberg: Springer, 1991, pp. 221–242.

[55] ——, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Advances in Cryptology — CRYPTO '91*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer, 1992, pp. 129–140.

[56] K. Ramchen, C. Culnane, O. Pereira, and V. Teague, "Universally Verifiable MPC and IRV Ballot Counting," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds. Cham: Springer International Publishing, 2019, pp. 301–319.

[57] B. Schoenmakers, "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting," in *Advances in Cryptology — CRYPTO' 99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed. Berlin, Heidelberg: Springer, 1999, pp. 148–164.

[58] B. Schoenmakers, M. Veeningen, and N. de Vreede, "Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation," in *Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, M. Manulis, A.-R. Sadeghi, and S. Schneider, Eds. Cham: Springer International Publishing, 2016, pp. 346–366.

[59] M. Veeningen, "Pinocchio-Based Adaptive zk-SNARKs and Secure/Correct Adaptive Function Evaluation," 2017, publication info: Published elsewhere. Minor revision. Proceedings AFRICACRYPT 2017. [Online]. Available: https://eprint.iacr.org/2017/013

[60] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-Efficient zkSNARKs Without Trusted Setup," in *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, May 2018, pp. 926–943.

[61] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "DIZK: A Distributed Zero Knowledge Proof System," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD, USA: USENIX Association, 2018, pp. 675–692. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/wu

APPENDIX

A. Additional Related Work

PA-MPC from distributed proofs. The research on PA-MPC [6] is closely related to the context of our work. However, constructions for PA-MPC are often very specific to one scheme and do not consider composability, contrary to our generic, modular approach. In PA-MPC, a group of parties, each holding some private values, collaboratively evaluates a function, in such a way that correctness of the result can be verified by any external party. PA-MPC is often proposed for the outsourcing setting, in which data parties secret share there data to multiple servers, who compute the proof in a distributed fashion. Since data parties themselves can also be computation servers, it can also be used in the setting of VPPCs [15]. Correctness is, generally, derived from commitments to the parties' inputs.

In the initial construction for PA-MPC [6] a verifier had to assert correctness of a full transcript. However, later it was shown by Veeningen that PA-MPC can be made more efficient by creating a distributed ZKP on committed data. Veeningen showed how to construct PA-MPC from an adaptive CP-SNARK based on Pinocchio [53]. Here, adaptive means that the scheme is secure even when the relation is chosen after the input commitments are generated. PA-MPC is realized by applying the CP-SNARK approach to Trinocchio [58], a distributed version of [53].

Kanjalkar et al. [44] improve upon [59], by constructing PA-MPC from a new CP-SNARKs based on Marlin [20], which only requires a single trusted setup ceremony that can be used for any statement, rather than the relation-specific trusted setup of Pinocchio. More recently, Dutta et al. [31] consider the notion of *authenticated MPC*, on signed inputs. They construct several distributed proofs of knowledge for opening Pedersen commitments and compressed Σ-protocols [5].

B. Distributed Inner-Product Argument.

A core building block of the bulletproof system is the inner product argument (IPA). It allows the prover to convince a verifier that a scalar $c \in \mathbb{Z}_p$ is the correct inner-product of two vectors a and b where $a, b \in \mathbb{Z}_p^n$. Bulletproofs employs a recursive argument to demonstrate the validity of an inner product relation under cryptographic commitments. In the bulletproofs setup, the IPA provides a structured proof for the relation:

$$\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) \colon P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \cdot u^{\langle \mathbf{a}, \mathbf{b} \rangle}\}.$$

Here, it is assumed that n is a power of 2, ensuring the inputs align correctly; if not, zero padding may be applied to the vectors. To prove the above relation, the prover conducts $k = \log_2 n$ rounds, where in each round, the prover sends a commitment to the left half of the vector L and a commitment to the right half of the vector R to the verifier, receives a challenge x, and refines the vectors $\mathbf{a}, \mathbf{b}, \mathbf{G}, \mathbf{H}$ for the next round. This reduction process is crucial for enhancing the efficiency of the protocol. If the reduced commitment maintains the prescribed relation at any round, it implies, with overwhelming probability, that the original, un-reduced commitment also satisfies the relation. The interactive protocol is described in [16], and by employing the Fiat-Shamir heuristic, the protocol can be made non-interactive, substituting rounds of interaction with cryptographic hashes of the commitments as we do in our implementation.

When extended to multiple provers, the protocol can be adapted to a distributed setting where each prover holds parts of the vectors a and b. Each prover can then create shares of their parts and distribute them to all parties, resulting in each prover having shares of vectors a and b, denoted by $[\![\mathbf{a}]\!]_i, [\![\mathbf{b}]\!]_i$ for $i \in [N]$ where N is the number of provers. These provers collaboratively generate parts of the overall proof without revealing their individual vector shares to the verifier or each other. To achieve this, each prover runs the IPA protocol with their respective shares in parallel and broadcasts the resulting commitments $[\![L]\!]_i$ and $[\![R]\!]_i$ to obtain the challenge x. At the conclusion of the protocol, each party broadcasts their partial proof, which is then combined to make the proof that the verifier can use. The non-interactive distributed protocol is summarized in Figure 9 for the prover and Figure 10 for the verifier. We note that the verification in Figure 10 is done through a single multi-exponentiation as described in [16, Section 3.1]. This method enhances verification speed, and we have adopted it in our implementation as well.

Additionally, in Figure 12 we compare the performance of using the distributed IPA compared to running the IPA on revealed $\mathbf{a}$ and $\mathbf{b}$ vectors. This clearly shows that the distributed version incurs a noticeable overhead over the regular version.

$\mathcal{P}_i$'s input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u \in \mathbb{G}, [\![\mathbf{a}]\!]_i, [\![\mathbf{b}]\!]_i \in \mathbb{Z}_p^n)$
Output: π

1. Each $\mathcal{P}_i$ computes:
   $n' \leftarrow n$, $k \leftarrow \log_2 n$
   $[\![\mathbf{a}']\!]_i \leftarrow [\![\mathbf{a}]\!]_i$, $[\![\mathbf{b}']\!]_i \leftarrow [\![\mathbf{b}]\!]_i$
   $\mathbf{g}' \leftarrow \mathbf{g}$, $\mathbf{h}' \leftarrow \mathbf{h}$
   $x_0 \leftarrow 0$
   for $j = 1, \ldots, k$
   $n' = \frac{n}{2}$
   $[\![c_{L,j}]\!]_i = \langle [\![\mathbf{a}'_{[:n']}]\!]_i, [\![\mathbf{b}'_{[n':]}]\!]_i \rangle \in \mathbb{Z}_p$
   $[\![c_{R,j}]\!]_i = \langle [\![\mathbf{a}'_{[n':]}]\!]_i, [\![\mathbf{b}'_{[:n']}]\!]_i \rangle \in \mathbb{Z}_p$
   $[\![L_j]\!]_i = \mathbf{g}_{[n':]}^{[\![a'_{[:n']}]\!]_i} \mathbf{h}_{[:n']}^{[\![b'_{[n':]}]\!]_i} u^{[\![c_{L,j}]\!]_i} \in \mathbb{G}$
   $[\![R_j]\!]_i = \mathbf{g}_{[:n']}^{[\![a'_{[n':]}]\!]_i} \mathbf{h}_{[n':]}^{[\![b'_{[:n']}]\!]_i} u^{[\![c_{R,j}]\!]_i} \in \mathbb{G}$
   $\mathcal{P}$: open (L_j, R_j)
   $\mathcal{P} : x_j \leftarrow \mathbf{H}(x_{j-1}, L_j, R_j) \in \mathbb{Z}_p$
   $[\![\mathbf{a}']\!]_i = [\![\mathbf{a}'_{[:n']}]\!]_i \cdot x_j + [\![\mathbf{a}'_{[n':]}]\!]_i \cdot x_j^{-1} \in \mathbb{Z}_p^{n'}$
   $[\![\mathbf{b}']\!]_i = [\![\mathbf{b}'_{[:n']}]\!]_i \cdot x_j^{-1} + [\![\mathbf{b}'_{[n':]}]\!]_i \cdot x_j \in \mathbb{Z}_p^{n'}$
   $\mathbf{g}' = \mathbf{g}'^{x_j^{-1}}_{[:n']} \circ \mathbf{g}'^{x_j}_{[n':]} \in \mathbb{G}^{n'}$
   $\mathbf{h}' = \mathbf{h}'^{x_j}_{[:n']} \circ \mathbf{h}'^{x_j^{-1}}_{[n':]} \in \mathbb{G}^{n'}$
2. $\mathcal{P}$: open (a', b')
3. **Output**: $(\mathbf{L}, \mathbf{R} \in \mathbb{G}^k, a', b' \in \mathbb{Z}_p)$

Fig. 9. Distributed IPA prover protocol Π_{DIPA}.Prove.

$\mathcal{V}$'s input:
$(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}, \pi = (\mathbf{L}, \mathbf{R} \in \mathbb{G}^k, a, b \in \mathbb{Z}_p))$
Output: $\{\mathcal{V}$ accepts, or $\mathcal{V}$ rejects$\}$

1. $\mathcal{V}$ computes challenges $\mathbf{x} \in \mathbb{Z}_p^k$:
   $x_0 \leftarrow 0$
   for $j = 1, \ldots, k$
   $x_j \leftarrow \mathbf{H}(x_{j-1}, L_j, R_j)$
2. $\mathcal{V}$ computes $\mathbf{s} \in \mathbb{Z}_p^n$:
   $s_i = \prod_{j=1}^k x_j^{b(i,j)} \quad \forall i \in [1, n]$
   where: $b(i,j) = \begin{cases} 1 & \text{the } j\text{th bit of } i-1 \text{ is 1} \\ -1 & \text{otherwise} \end{cases}$
3. $\mathcal{V}$ checks:
   $\mathbf{g}^{a \cdot \mathbf{s}} \cdot \mathbf{h}^{b \cdot \mathbf{s}^{-1}} \cdot u^{a \cdot b} \stackrel{?}{=} P \cdot \prod_{j=1}^k L_j^{(x_j^2)} \cdot R_j^{(x_j^{-2})}$
   - If yes, $\mathcal{V}$ accepts; otherwise $\mathcal{V}$ rejects.

Fig. 10. Distributed IPA verifier protocol Π_{DIPA}.Verify.

C. Bulletproofs Verifier Protocol

In Figure 11, we present the non-interactive version of the Bulletproof verification algorithm, adapted for the commit-and-prove setting. At a high level, this algorithm mirrors the standard verification protocol with the addition of $\mathsf{CP}_{\mathsf{link}}$ verification. Besides the proof and public parameters, the verification protocol requires $\hat{V}$, an external commitment to the same witnesses in $\mathbf{V}$, and $\mathbf{vk}_{\mathsf{CP}_{\mathsf{link}}}$, the verification key used for $\mathsf{CP}_{\mathsf{link}}$ verification. This ensures that the witnesses in $\mathbf{V}$ satisfy the arithmetic circuit and are linked to the external commitment $\hat{V}$. Additionally, this Bulletproof verification uses IPA to reduce communication costs, as detailed in Figure 10.

D. Additional Experimental Evaluations

$\mathsf{CP}_{\mathsf{link}}$ overhead. In this experiment, we measured the overhead of $\mathsf{CP}_{\mathsf{link}}$, a technique, as described previously, that allows

$\mathcal{V}$**'s input:**

$$\left(\begin{array}{c} g, h \in \mathbb{G}, \mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{c} \in \mathbb{Z}_p^Q, \mathbf{V} \in \mathbb{G}^m, \hat{V} \in \mathbb{G}, \\ \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}, \pi_{\mathsf{CP}_{\mathsf{link}}}, \mathbf{vk}_{\mathsf{CP}_{\mathsf{link}}}, \\ \pi = (A_I, A_O, S, T_1, T_3, T_4, T_5, T_6, \tau_x, \mu, \hat{t}, \pi_{IPA}) \end{array}\right)$$

Output: $\{\mathcal{V}$ accepts,$\mathcal{V}$ rejects$\}$

1. $\mathcal{V}$ computes challenges from π: $\{y, z, x, x_u\}$
2. $\mathcal{V}$ computes and checks:

$$\begin{aligned} \mathbf{y}^n &= (1, y, y^2, \ldots, y^{n-1}) \in \mathbb{Z}_p^n \\ \mathbf{z}_{[1:]}^{Q+1} &= (z, z^2, \ldots, z^Q) \in \mathbb{Z}_p^Q \\ h_i' &= h_i^{y^{-i+1}} \quad \forall i \in [1, n] \\ W_L &= \mathbf{h}'^{\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L} \\ W_R &= \mathbf{g}^{\mathbf{y}^{-n} \circ \left(\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R\right)} \\ W_O &= \mathbf{h}'^{\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O} \\ P &= A_I^x \cdot A_O^{(x^2)} \cdot \mathbf{h}'^{-\mathbf{y}^n} \cdot W_L^x \cdot W_R^x \cdot W_O \cdot S^{(x^3)} \\ P' &= P \cdot h^{-\mu} \cdot g^{x_u \cdot \hat{t}} \\ \{0,1\} &\leftarrow \Pi_{IPA}.\mathsf{Verify}(\mathbf{g}, \mathbf{h}', g^{x_u}, P', \pi_{IPA}) \end{aligned}$$

3. $\mathcal{V}$ computes and checks:

$$\begin{aligned} V' &= \sum_{j=0}^{m} V_j \\ \{0,1\} &\leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Verify}(\mathbf{vk}_{\mathsf{CP}_{\mathsf{link}}}, V', \hat{V}, \pi_{\mathsf{CP}_{\mathsf{link}}}) \end{aligned}$$

4. If all checks succeed, $\mathcal{V}$ accepts, else $\mathcal{V}$ rejects.

Fig. 11. Protocol Π_{DBP}.Verify for verifying bulletproofs in the commit-and-prove setting (with $\mathsf{CP}_{\mathsf{link}}$).

linking the commitment to the witnesses in both LegoGro16 and Bulletproofs to an external commitment. The results provide insight into the cost associated with linking proofs in a distributed setting compared to a single prover setting. While $\mathsf{CP}_{\mathsf{link}}$ does come with a cost (overhead), it is certainly far more efficient than linking the commitments inside the circuit. Figure 13 shows the performance of $\mathsf{CP}_{\mathsf{link}}$ for an increasing number of provers where each prover contributes a witness to the commitment.

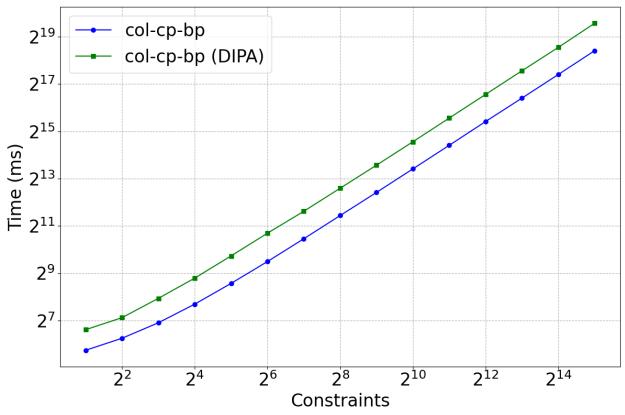

Fig. 12. Runtime of a single prover for: (1) IPA on opened $\mathbf{a}$ and $\mathbf{b}$ input vectors (col-cp-bp); (2) on shared input (col-cp-bp(DIPA)). The number of provers is fixed at 2.

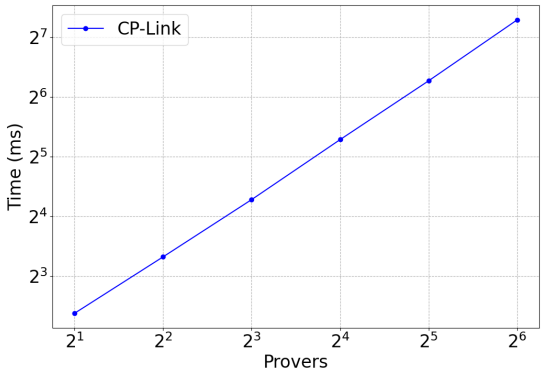

Fig. 13. Runtime per prover for $\mathsf{CP}_{\mathsf{link}}$ with a varying number of provers.