Split Prover Zero-Knowledge SNARKs

Sanjam Garg[*] Aarushi Goel[†] Dimitris Kolonelos[‡] Sina Shiehian[§] Rohit Sinha[¶]

Abstract

We initiate the study of *split prover zkSNARKs*, which allow Alice to offload part of the zkSNARK computation to her assistant, Bob. In scenarios like online transactions (e.g., zCash), a significant portion of the witness (e.g., membership proofs of input coins) is often available to the prover (Alice) before the transaction begins. This setup offers an opportunity to Alice to initiate the proof computation early, even before the entire witness is available. The remaining computation can then be delegated to Bob, who can complete it once the final witness (e.g., the transaction amount) is known.

To prevent Bob from generating proofs independently (e.g., initiating unauthorized transactions), it is essential that the data provided to him for the second phase of computation does not reveal the witness used in the first phase. Additionally, the verifier of the zkSNARK should be unable to determine whether the proof was generated solely by Alice or through this two-step process. To achieve this efficiently, we require this two-phase proof generation to only use cryptography in a black-box manner.

We propose a split prover zkSNARK based on the Groth16 zkSNARKs [Groth, EUROCRYPT 2016], meeting all these requirements. Our solution is also *asymptotically tight*, meaning it achieves the optimal second phase proof generation time for Groth16. Importantly, our split prover zkSNARK preserves the verification algorithm of the original Groth16 zkSNARK, enabling seamless integration into existing deployments of Groth16.

[*]UC Berkeley sanjamg@berkeley.edu
[†]Purdue University aarushi.goel794@gmail.com
[‡]UC Berkeley dimitris.kolonelos@berkeley.edu
[§]Snap Inc. shiayan@umich.edu
[¶]Swirlds Labs sinharo@gmail.com

Contents

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [Mic94, BCC+17] are cryptographic tools that allow a prover to generate a compact certificate validating the correctness of a potentially complex computation. These certificates are efficient to verify and protect any secrets used by the prover during the computation. zkSNARKs have found utility in various modern cryptographic applications. Investigating the feasibility of zkSNARKs in different models, under diverse security assumptions, and realizing them efficiently has been an active area of research in recent years.

In this work, we explore a new *prover model* for generating zkSNARKs. Consider a scenario where Alice wants to perform an online transaction (e.g., in zCash [BSCG+14]). She knows part of the witness (e.g., her private key, membership proof of input coins, upper bound on the transaction amount) needed to generate a zkSNARK for the transaction, but the exact transaction amount is not yet known. Additionally, Alice might be unavailable when the transaction amount becomes known. We ask whether Alice can initiate the zkSNARK computation using the available information and delegate the remaining computation to her assistant, Bob, who can complete it once the transaction amount is determined.

A similar application involves anonymous credentials. For instance, Alice needs electronic authorization for international travel and must prove, using a zkSNARK, that she holds a valid US passport. Can she start the zkSNARK computation using her passport and delegate the remaining computation to Bob, who can finalize it once the travel dates are confirmed?

In these applications, it is crucial to ensure that Bob cannot independently generate unauthorized proofs. The data sent to Bob for the second phase of proof computation must not disclose any part of the witness used in the first phase. Moreover, for seamless integration into existing systems, the verifier receiving the final zkSNARK should not be able to tell whether the proof was generated solely by Alice or through a delegated two-step process. Finally, for efficiency, we require the final proof to be succinct and the two-phase proof generation to only use cryptographic operations in a black-box manner.[1]

In other words, we aim to determine the following:

Is it possible to generate zkSNARKs in two-phases using cryptography in a black-box way, while ensuring that the output of the first phase preserves privacy?

1.1 Our Contributions

In this work, we answer the above question in the affirmative and present the following contributions.

Defining Split Prover zkSNARKs. We introduce the notion of split prover zkSNARKs which enable proof generation to be divided into two phases. Simply put, this means that the secret witness w associated with the statement being proven, can be divided into two segments – one for each phase. By utilizing the first segment to commence proof generation, the remaining zkSNARK computation can be delegated to an external entity, who only needs the second segment of the witness to finalize the proof.

A key requirement here is that even with this two-phase prover setup, the zkSNARK verifier algorithm should remain unchanged. We further require that the state that is generated in the first phase (and given as input to the external entity for delegation of the second phase) should reveal no information beyond the output of the relation circuit when partially evaluated using the first segment of the witness.

[1]We defer the reader to Section 1.3, for discussion on the disadvantages of a non-black box approach.

Split Prover zkSNARK Based on Groth16. Next, we present a split prover zkSNARK based on the widely used Groth16 zkSNARK [Gro16] (henceforth referred to as Groth16). More concretely, let C to be any circuit defining an NP relation $\mathcal{R}$ and let C_1 and C_2 be the subcircuits of C corresponding to the two segments of the witness. Then, for a witness $w = (w_{\mathrm{I}}, w_{\mathrm{II}})$ and statement $x = (x_{\mathrm{I}}, x_{\mathrm{II}})$ split into two segments, we can write $C = C_{\mathrm{II}}(x_{\mathrm{II}}, w_{\mathrm{II}}, C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}}))$. We obtain the following result:

Informal Theorem 1. *Groth16 admits a split prover, where,*

- *the first phase of proof generation runs in time* $O(|C_I| \cdot |C| \log |C|)$,[2]
- *the second phase of proof generation runs in time* $O(\mathsf{Min}\{|C_{II}|^2, |C| \log |C|\})$,[2] *and*
- *the verifier algorithm is identical*[3] *to the Groth16 proof system.*

In the above theorem, if $|C_{\mathrm{II}}| \in o(\sqrt{|C|})$, then the second phase of proof generation runs in time $O(|C_{\mathrm{II}}|^2)$. Else, if $|C_{\mathrm{II}}| \in \Omega(\sqrt{|C|})$, then the second phase of proof generation runs in time $O(|C| \log |C|)$.

Lower Bound for Split Prover Groth16. Since group operations are the main bottleneck in the generation of Groth16 SNARKs, we characterize the number of group operations that must be performed during the second phase of proof generation in any split prover variant of Groth16.

Informal Theorem 2. *In any split prover variant for Groth16, the second phase of proof generation must involve* $\Omega(\mathsf{Min}\{|C_{II}|^2, |C|\})$ *group operations.*

This shows that the number of group operations performed in the second phase of proof generation in our protocol from Informal Theorem 1 is asymptotically tight.

1.2 Application to Delegatable Payments and Beyond

As discussed earlier, our work is motivated by applications of zkSNARKs, where the witness can be partitioned into two segments – one accessible to the prover apriori, and the other disclosed later when the prover may be unavailable. This situation presents an opportunity for the prover to initiate the zkSNARK computation using available information and delegate the remaining tasks to an external entity. Now, we delve into how this witness division applies specifically to Zerocash [BSCG$^+$14] proofs for anonymous payments, enabling a prover to leverage our split prover zkSNARK to delegate a portion of the computation to an external entity.

Consider a simplified version of the zCash[4] [HBHW22] JoinSplit transaction. A JoinSplit transaction lets a payer consume two coins and create two new coins – typically, one output coin is issued to the payee, while the other output coin has the left-over change and is issued back to the payer. In Zerocash, a coin is spent (or nullified) by revealing its serial number, while a new coin is created by publishing a (randomized) commitment to a data structure containing the coin's value and the owner's public key. The payment is settled on-chain by submitting a transaction containing (sn_1, sn_2, cm_1', cm_2', π); here, sn_1 and sn_2 denote

[2]This includes both group and field operations. The total number of group operations performed by the prover in the first phase are $O(|C_{\mathrm{I}}| \cdot |C|)$ and in the second phase are $O(\mathsf{Min}\{|C_{\mathrm{II}}|^2, |C|\})$

[3]In Groth16, the common reference string (CRS) can be split into two parts – one for the prover and one for the verifier. While the verifier's part remains unchanged, our split prover adaptation of Groth16 requires the inclusion of some extra terms in the prover's section of the CRS.

[4]zCash [HBHW22] is a cryptocurrency that deploys the academic work Zerocash [BSCG$^+$14]. Although, prior versions of zCash were instantiating the zkSNARK component with Groth16 its current implementation has switched to a different SNARK [Zca]. Our work is still compatible with the cryptographic framework of Zerocash for anonymous transactions.

serial numbers for spent coins, while commitments cm_1' and cm_2' denote the new output coins. Finally, a zero-knowledge proof π attests to the transaction's validity, and it has the following basic form (using the notation and naming in [BSCG+14]):

- **public variables**: root, sn_1, sn_2, cm_1', cm_2'
- **secret witness**:

  $\mathsf{cm}_1, \mathsf{v}_1, \mathsf{r}_1, \mathsf{s}_1, \rho_1, \mathsf{apk}_1, \mathsf{ask}_1, \mathsf{h}_1^1, \ldots, \mathsf{h}_1^{31}$
  $\mathsf{cm}_2, \mathsf{v}_2, \mathsf{r}_2, \mathsf{s}_2, \rho_2, \mathsf{apk}_2, \mathsf{ask}_2, \mathsf{h}_2^1, \ldots, \mathsf{h}_2^{31}$
  $\mathsf{v}_1', \mathsf{r}_1', \mathsf{s}_1', \rho_1', \mathsf{apk}_1'$
  $\mathsf{v}_2', \mathsf{r}_2', \mathsf{s}_2', \rho_2', \mathsf{apk}_2'$

- **relation**: conjunction of the following five predicates:
  - membership proof that the spent coins were created previously on ledger:
    $\mathsf{MerkleVerify}(\mathsf{root}, \mathsf{cm}_1, \mathsf{h}_1^1, \ldots, \mathsf{h}_1^{31}) \wedge \mathsf{MerkleVerify}(\mathsf{root}, \mathsf{cm}_2, \mathsf{h}_2^1, \ldots, \mathsf{h}_2^{31})$
  - well-formedness of the data structures encoding the spent coins:
    $\mathsf{cm}_1 = \mathsf{Com}(\mathsf{v}_1, \mathsf{Com}(\mathsf{apk}_1, \rho_1; \mathsf{s}_1); \mathsf{r}_1) \ \wedge \ \mathsf{cm}_2 = \mathsf{Com}(\mathsf{v}_2, \mathsf{Com}(\mathsf{apk}_2, \rho_2; \mathsf{s}_2); \mathsf{r}_2)$
  - ownership of spent coins (via knowledge of openings to commitments):
    $\mathsf{sn}_1 = \mathsf{PRF}(\rho_1; \mathsf{ask}_1) \ \wedge \mathsf{apk}_1 = \mathsf{PRF}(0; \mathsf{ask}_1) \ \wedge$
    $\mathsf{sn}_2 = \mathsf{PRF}(\rho_2; \mathsf{ask}_2) \ \wedge \mathsf{apk}_2 = \mathsf{PRF}(0; \mathsf{ask}_2)$
  - well-formedness of the data structures encoding the new output coins:
    $\mathsf{cm}_1' = \mathsf{Com}(\mathsf{v}_1', \mathsf{Com}(\mathsf{apk}_1', \rho_1'; \mathsf{s}_1'); \mathsf{r}_1') \ \wedge$
    $\mathsf{cm}_2' = \mathsf{Com}(\mathsf{v}_2', \mathsf{Com}(\mathsf{apk}_2', \rho_2'; \mathsf{s}_2'); \mathsf{r}_2')$
  - conservation of value: $\mathsf{v}_1 + \mathsf{v}_2 = \mathsf{v}_1' + \mathsf{v}_2'$

For simplicity, we hide details such as range checks, viewership keys, etc. Above, we use blue to indicate values available and constraints that can be evaluated by the prover (i.e., delegator) before the transaction amount is known. We let the payer's device choose two of her coins to join for the transaction before she engages in a payment; in practice, this could be the two coins whose cumulative value is the largest, or at least exceeds some expected payment amount. Therefore, the delegator is able to evaluate the arithmetic circuit wires corresponding to the two Merkle verifications; the delegator can also perform the computation necesssary for proving well-formedness and ownership of those spent coins. The commitments to the new coins are determined in later, as are the constraints enforcing the value conservation. As a result, the computation needed for enforcing these constraints and for proving well-formedness of the data structures encoding the new output coins can be delegated to someone else.

Other Applications. In addition to anonymous transaction, we observe this witness split in other classes of applications. In anonymous credentials, the user can prove validity of an issued credential on his own, before delegating the commputation necessary for proving additional properties about the credential to an external entity – we find [RPX+22] to be a system which can use the split prover Groth16 construction in this paper. Additionally, applications that need validity proofs for ciphertexts (e.g. [GAZ+22]), encrypted under a hybrid encryption scheme, can also benefit from our split prover zkSNARK, since the component of the circuit encoding the key encapsulation mechanism can be evaluated long before the message to be encrypted is determined.

1.3 Additional Discussion

Comparison with Recursive SNARKs. An astute reader might wonder how our notion of split prover zkSNARKs relates to the well-studied notion of incrementally verifiable computation (IVC) [Val08] and, whether recursive zkSNARKs [KST22, BCTV14, BCCT13] – which are used to construct IVCs – could also be utilized to design split prover zkSNARKs. We note that while the IVCs and split prover zkSNARKs bear some similarities, these are distinct notions.

Compared to our split prover zkSNARKs, IVCs have two advantages. First, IVCs allow the proof to be computed in any number of phases (potentially even greater than two). Second, in each phase of IVC, the prover's runtime is proportional to the portion of the computation being proven in that phase, whereas in our construction of split prover zkSNARK, the second phase of proof generation scales with the entire computation.[5]

However, these advantages come at the cost of providing only a theoretically questionable heuristic soundness guarantee, due to the use of idealized oracles such as the random oracle or the generic group model in a non-black-box manner. Such non-black-box use of the random oracle, in general, has recently been shown to be insecure [BCG24]. In contrast, our split prover zkSNARK inherits the same soundness guarantees that Groth16 provides, which can be established in well studied idealized models [Sho97, FKL18]. In contrast, our construction is black-box in the use of cryptography. Another advantage of our split prover zkSNARK is that the verifier algorithm does not depend on how the computation is split into the two phases. In comparison, in IVCs, verification depends on the specific splitting of the computation. Therefore, it is unclear how to use recursive proofs to design a zkSNARK that meets all the requirements of a split prover zkSNARK.

Comparison with other zkSNARK Delegation Frameworks. An orthogonal problem to ours involves delegating zkSNARK computation to third-party cloud servers to ease the burden of proof computation on provers. This topic has been explored in several prior works [BCG$^+$20b, WZC$^+$18, GGJ$^+$23, CLMZ23, GGW23, LZW$^+$24]. Unlike our model, in these works, the delegator possesses the entire witness at the time of delegating the computation. While some of these works [BCG$^+$20b, WZC$^+$18] do not focus on privacy-preserving delegation, others either [GGJ$^+$23, CLMZ23, LZW$^+$24] use MPC for privacy-preserving distributed delegation to multiple servers or rely [GGW23] on the heavy-hammer of fully-homomorphic encryptio (FHE) to ensure privacy.

Barriers for zkSNARKs in the Random Oracle Model. Given our construction for Groth16, a natural question arises as to whether we can extend our techniques to construct split prover versions of other zkSNARKs, namely those in the random oracle model [BCG$^+$17, BCG$^+$18, XZZ$^+$19, Set20, BCG20a, Lee21, KMP20, BCL22, CHM$^+$20, GWC19, ZLW$^+$21, COS20]. These zkSNARKs are popular because they have a universal setup and some of them (e.g. [GWC19]) also provide support for flexible gates. Most of these zkSNARKs are obtained by transforming an interactive public-coin protocol into its non-interactive counter-part using the Fiat-Shamir [FS87] transform. Unfortunately, this incorporation of the Fiat-Shamir transform in these zkSNARKs appears to present an obstacle for us, when it comes to applying our techniques.

Roughly speaking, the main problem is that when the delegator computes a part of the proof apriori, it is unclear how the verifier's challenge messages can be derived. In particular, when applying the Fiat-Shamir transform, the verifier's challenges are derived by querying the random oracle at inputs that depend

[5]Unless the the second segment of the witness is small, $|C_{\mathrm{II}}| = o(\sqrt{|C|})$, as indicated in Informal Theorem 1.

on the "entire computation" and not just a part of the witness. As such it is unclear what parts of the proof can be pre-computed, without knowledge of these challenge messages. We leave the exploration of new techniques to design split provers for such zkSNARKs as exciting future work.

2 Preliminaries

Notation. Throughout this work, we use $\lambda \in \mathbb{N}$ to denote the security parameter and we assume that each algorithm implicitly takes the security parameter as input. $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ will be used to denote polynomial and negligible functions respectively. We use "PPT" to refer to Probabilistic Polynomial-time Algorithms, and unless otherwise stated all the algorithms of our schemes are such. For any positive integer $n \in \mathbb{Z}$ $[n]$ denotes the set of integers $\{1, \ldots, n\}$ and, more generally, for any $A, B \in \mathbb{Z}$, $A \leq B$, $[A, B]$ denotes the set $\{A, \ldots, B\}$. $x \leftarrow\$ X$ is used to imply that x is being uniformly sampled from a finite set X.

We write vectors with bold small letters, e.g. $\boldsymbol{v}$ and with bold capital letters matrices, e.g. $\boldsymbol{A}$. We treat vectors as column matrices, e.g. $\boldsymbol{v} = \begin{pmatrix} v_1 & v_2 & \ldots & v_n \end{pmatrix}^\top$. We also sometimes write concisely $\boldsymbol{v} = (v_i)_{i \in [n]}$ for vectors or $\boldsymbol{A} = (a_{i,j})_{i \in [n], j \in [m]}$ for matrices.

By $\left\lfloor \frac{f(X)}{g(X)} \right\rfloor$ we denote the quotient polynomial of the division $f(X)/g(X)$. We denote the i-th coefficient of a polynomial $f(X)$ as $\tilde{f}_i$, e.g. $\kappa_5(X) = \tilde{\kappa}_{5,0} + \tilde{\kappa}_{5,1}X + \ldots + \tilde{\kappa}_{5,n}X^n$. By $\boldsymbol{f}(X)$ we denote a vector of polynomials, $\boldsymbol{f}(X) = (f_1(X), f_2(X), \ldots, f_n(X))^\top$. $\tilde{\boldsymbol{f}}_i$ denotes the vector of the corresponding i-th coefficients, i.e. $\tilde{\boldsymbol{f}}_i = (\tilde{f}_{1,i}, \tilde{f}_{2,i}, \ldots, \tilde{f}_{n,i})^\top$. Similarly with $\boldsymbol{F}(X)$ a matrix of polynomials. $L_i(X) = \prod_{j \in [n], j \neq i} \frac{X - \omega^j}{\omega^i - \omega^j}$ will be the lagrange polynomial over a group $\{\omega, \omega^2, \ldots, \omega^n\}$ and $V(x) = \prod_{i=1}^{\ell}(x - \omega^i)$ the vanishing polynomial.

2.1 Bilinear Groups

A bilinear group generator $\mathcal{BG}$ takes as input a security parameter 1^λ and outputs a description $\mathsf{bg} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, where p is a prime of $\Theta(\lambda)$ bits, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerate bilinear map. We require that the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and the bilinear map e are computable in deterministic polynomial time in λ. Let $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $g_T = e(g_1, g_2) \in \mathbb{G}_T$ be the respective generators. We employ the *implicit representation* of group elements: for a matrix $\boldsymbol{M}$ over $\mathbb{Z}_p$, we define $[\boldsymbol{M}]_1 := g_1^{\boldsymbol{M}}, [\boldsymbol{M}]_2 := g_2^{\boldsymbol{M}}, [\boldsymbol{M}]_T := g_T^{\boldsymbol{M}}$, where exponentiation is carried out component-wise.

2.2 Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge

Here we recall the definition of zkSNARKs.

Definition 1 (zkSNARKs). *A SNARK for a family of relations R_λ consists of three algorithms* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$*:*

$\underline{\mathsf{Setup}(\mathcal{R}) \rightarrow (\mathsf{srs})}$: *On input a relation $\mathcal{R} \in R_\lambda$ the setup algorithm outputs a structured reference string* srs.

$\underline{\mathcal{P}(\mathsf{srs}, x, w) \rightarrow \pi}$: *On input the structured reference string* srs*, a statement x and a witness w the prover algorithm outputs a proof π.*

$\underline{\mathcal{V}(\mathsf{srs}, x, \pi) \rightarrow 0/1}$: *On input the structured reference string* srs*, a statement x and a proof π the verifier algorithm outputs either* 1 *for accept or* 0 *for reject.*

It is further required that the following properties hold.

Correctness. *For each $\lambda \in \mathbb{N}$, each relation $\mathcal{R} \in R_\lambda$, and every statement-witness pair $(x, w) \in \mathcal{R}$:*

$$\Pr\left[\mathcal{V}(\mathsf{srs}, x, \pi) = 1 \;:\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{Setup}(\mathcal{R}) \\ \pi \leftarrow \mathcal{P}(\mathsf{srs}, x, w) \end{array} \right] = 1$$

Knowledge Soundness. *For every PPT adversarial prover P^*, there exists a PPT extractor $\mathcal{E}_{P^*}$ such that for every security parameter $\lambda \in \mathbb{N}$, every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and every relation $\mathcal{R} \in R_\lambda$:*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{srs}, x, \pi) = 1 \\ \wedge (x, w) \notin \mathcal{R} \end{array} \;:\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{Setup}(\mathcal{R}) \\ (x, \pi)^* \leftarrow \mathcal{P}^*(\mathsf{srs}, \mathsf{aux}) \\ w \leftarrow \mathcal{E}_{P^*}(\mathsf{srs}, \mathsf{aux}) \end{array} \right] = \mathsf{negl}(\lambda)$$

Succinctness. *There exists a universal polynomial $p(\cdot)$ such that, for every security parameter $\lambda \in \mathbb{N}$, every relation $\mathcal{R} \in R_\lambda$, and every statement-witness pair (x, w):*

- *An honestly generated proof π has size $p(\lambda + \log |w|)$.*
- *The verifier algorithm $\mathcal{V}(\mathsf{srs}, x, \pi)$ runs in time $p(\lambda + |x| + \log |w|)$.*

(Perfect) Zero-Knowledge. *For every security parameter $\lambda \in \mathbb{N}$ and every relation $(\mathcal{R}, \mathsf{aux}_R) \leftarrow R_\lambda$, there exists a simulator $\mathcal{S}$ such that, for every statement-witness pair $(x, w) \in \mathcal{R}$ and for every computationally unbounded adversary $\mathcal{A}$:*

$$\begin{aligned} &\Pr\left[\mathcal{A}(R, \mathsf{aux}_R, \mathsf{srs}, \pi) = 1 \;:\; \begin{array}{c} \mathsf{srs} \leftarrow \mathsf{Setup}(\mathcal{R}) \\ \pi \leftarrow \mathcal{P}(\mathsf{srs}, x, w) \end{array} \right] \\ =&\Pr\left[\mathcal{A}(R, \mathsf{aux}_R, \mathsf{srs}, \pi) = 1 \;:\; (\mathsf{srs}, \pi) \leftarrow \mathcal{S}(x, \mathcal{R}) \right] \end{aligned}$$

If the Zero-Knowledge property is not satisfied we call the proof system a SNARK (without zk).

2.3 The Groth16 zkSNARK

We recall the Groth16 proof system [Gro16].

2.3.1 Rank-1 constraint satisfiability (R1CS).

Groth16 works for relations encoded with the rank-1 constraint satisfiability (R1CS). Assume that we have n constraints and m variables. The constraint system consists of:

$$\begin{pmatrix} a_{1,1} & a_{2,1} & \dots & a_{m,1} \\ a_{1,2} & a_{2,2} & \dots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \dots & a_{m,n} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \circ \begin{pmatrix} b_{1,1} & b_{2,1} & \dots & b_{m,1} \\ b_{1,2} & b_{2,2} & \dots & b_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,n} & b_{2,n} & \dots & b_{m,n} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{2,1} & \dots & c_{m,1} \\ c_{1,2} & c_{2,2} & \dots & c_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1,n} & c_{2,n} & \dots & c_{m,n} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}$$

where the matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are fixed and $\boldsymbol{z}$ is what we call the 'extended witness', consisting of the witness and the statement. Informally speaking, a translation to arithmetic circuits would be that the n constraints are the multiplication gates, the m variables the wires and $\boldsymbol{z}$ the actual values of the wires.. Of course, R1CS generalizes arithmetic circuits and shall not necessarily be regarded as a translation of such.

Formally an R1CS relation is of the form:

$$\mathcal{R} = \left\{ (\boldsymbol{x}; \boldsymbol{w}) : \boldsymbol{A}\boldsymbol{z} \circ \boldsymbol{B}\boldsymbol{z} = \boldsymbol{C}\boldsymbol{z} \wedge \boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w}) \right\}$$

where the relation is characterized by the matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{Z}_p^{n \times m}$ and $\boldsymbol{z} \in \mathbb{Z}_p^m$.

2.3.2 The Groth16 SNARK

For the proof system first each column of $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ is interpolated into polynomials as:

$$a_i(X) = \sum_{j=1}^{n} a_{i,j} L_j(X), \quad b_i(X) = \sum_{j=1}^{n} b_{i,j} L_j(X), \quad c_i(X) = \sum_{j=1}^{n} c_{i,j} L_j(X),$$

for each $i \in [m]$, where $L_j(x)$ the corresponding Lagrange polynomial. Then the prover should convince the verifier that

$$\left(\sum_{i=1}^{m} z_i a_i(X)\right) \cdot \left(\sum_{i=1}^{m} z_i b_i(X)\right) - \sum_{i=1}^{m} z_i c_i(X) = q(X)V(X)$$

where $V(X) = \prod_{i=1}^{n}(X - \omega^i)$ is the vanishing polynomial. This polynomial relation is essentially equivalent to the R1CS satisfiability (we refer to [GGPR13, PHGR13, Gro16] for more details).

The actual Groth16 SNARK is described below. Without loss of generality we assume that $\boldsymbol{x} = (z_1, \ldots, z_\ell)$ corresponds to the public statement.

$\underline{\mathsf{Setup}(\mathcal{R}) \to \mathsf{srs}:}$ Samples uniformly $\tau, \alpha, \beta, \gamma, \delta \leftarrow\!\!\$\ \mathbb{Z}_p$ and outputs:[6]

$$\mathsf{srs} = \Bigg\{ \{[\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_1, [\delta]_2, \left\{ \left[\tau^i\right]_1, \left[\tau^i\right]_2 \right\}_{i=0}^{n-1}, \left\{ \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 \right\}_{i=0}^{n-2},$$
$$\left\{[a_i(\tau)]_1, [b_i(\tau)]_1, [b_i(\tau)]_2\right\}_{i=1}^{m}, \left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma}\right]_1 \right\}_{i=1}^{\ell},$$
$$\left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1 \right\}_{i=\ell+1}^{m} \Bigg\}$$

$\underline{\mathcal{P}(\mathsf{srs}, \boldsymbol{x}, \boldsymbol{w}, \pi) \to \pi:}$ Sets $\boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w})$. Computes the quotient polynomial $q(X) = \left\lfloor \frac{(\sum_{i=1}^{m} z_i a_i(X)) \cdot (\sum_{i=1}^{m} z_i b_i(X)) - \sum_{i=1}^{m} z_i c_i(X)}{V(X)} \right\rfloor$. Then samples $r, s \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the group elements:

$$\pi_1 = [\alpha]_1 + \sum_{i=1}^{m} z_i [a_i(\tau)]_1 + r[\delta]_1$$
$$\pi_2 = [\beta]_2 + \sum_{i=1}^{m} z_i [b_i(\tau)]_2 + s[\delta]_2$$
$$\pi_3 = \sum_{i=\ell+1}^{m} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1 + s \sum_{i=1}^{m} z_i [a_i(\tau)]_1 + r \sum_{i=1}^{m} z_i [b_i(\tau)]_1$$
$$+ \sum_{i=0}^{n-2} \tilde{q}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 + s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1$$

Outputs $\pi = (\pi_1, \pi_2, \pi_3)$

[6] As noted in [Gro16], a_i, b_i, c_i are public polynomials and thus $\{[a_i(\tau)]_1, [b_i(\tau)]_1, [b_i(\tau)]_2\}_{i=1}^{m}$ can be publicly computed given $\{[\tau^i]_1, [\tau^i]_2\}_{i=0}^{n-1}$ without needing the trapdoor. Nevertheless, they are included in the srs for efficiency purposes.

$\underline{\mathcal{V}(\mathsf{srs}, \boldsymbol{x}, \pi) \to 0/1\text{:}}$ Outputs 1 iff:

$$e(\pi_1, \pi_2) = e\left([\alpha]_1, [\beta]_2\right) \cdot e\left(\sum_{i=1}^{\ell} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma}\right]_1, [\gamma]_2\right) \cdot e(\pi_3, [\delta]_2)$$

The proof system has knowledge soundness in the generic group model [Sho97, Mau05] and perfect zero-knowledge.

The prover's complexity is dominated by 7 Fast Fourier Transforms (FFTs) for polynomials of degree n over $\mathbb{Z}_p$, a Multi-Scalar Multiplication (MSM) of size m in $\mathbb{G}_1$, a Multi-Scalar Multiplication (MSM) of size m in $\mathbb{G}_2$ and another Multi-Scalar Multiplication (MSM) of size $3m - \ell + n$ in $\mathbb{G}_1$, overall $O(n \log n + m)$. Notably, in practice the dominant cost comes from the group operations (the MSMs) even when $n \log n > m$.

3 Defining Split Prover zkSNARKs

Here we formally define the notion of Split Prover zkSNARKs. The idea is that in an already well-defined SNARK one can replace the prover $\mathcal{P}$ with two phase provers $\mathcal{P}_{\mathrm{I}}$, $\mathcal{P}_{\mathrm{II}}$. For this we further allow for a new setup to possibly run, to generate a split common reference string. The verifier $\mathcal{V}$ should, nevertheless, remain the same.

Apart from the functionality, for the primitive to be meaningful we also define a zero-knowledge property for the outcome of the first-phase prover that is passed to the second-phase prover. We formalize this in the Split Zero-Knowledge property.

Definition 2. *Let $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ be (zk)SNARK, we say that Π admits a split prover if there exist algorithms $\Pi_{\mathsf{split}} = (\mathsf{Setup}_{\mathsf{split}}, \mathcal{P}_{I}, \mathcal{P}_{II})$ such that for any relation $\mathcal{R} \in R_\lambda$:*

- $\underline{\mathsf{Setup}_{\mathsf{split}}(\mathcal{R}, \mathcal{X}_{II}, \mathcal{W}_{II}) \to \widetilde{\mathsf{srs}}}$*: On input a relation $\mathcal{R}$ and sets of indices $\mathcal{X}_{II}$ and $\mathcal{W}_{II}$ specifying the portions of the statement and the witness of the second phase respectively, the split prover setup outputs a split prover structured reference string $\widetilde{\mathsf{srs}}$.*
- $\underline{\mathcal{P}_{I}(\widetilde{\mathsf{srs}}, x_I, w_I) \to \mathsf{aux}}$*: is a PPT algorithm that on input the split prover structure reference string $\widetilde{\mathsf{srs}}$, the part of the statement that is available in the first phase x_I and the part of the witness that is available in the first phase w_I outputs an auxiliary information for the prover of the second phase,* aux.
- $\underline{\mathcal{P}_{II}(\widetilde{\mathsf{srs}}, x_{II}, w_{II}, \mathsf{aux}) \to \pi}$*: is a PPT algorithm that on input the split prover structure reference string $\widetilde{\mathsf{srs}}$, the part of the statement that is available in the second phase x_{II}, the part of the witness that is available in the second phase w_{II} and the auxiliary information from $\mathcal{P}_I$,* aux, *outputs the proof π.*

We further consider the following properties:

Split Correctness. *We say that Π with Π_{split} has (perfect) split correctness if,*

$$\Pr\left[\mathcal{V}(\mathsf{srs}, \boldsymbol{x}, \pi) = \mathcal{V}(\widetilde{\mathsf{srs}}, \boldsymbol{x}, \pi') \;\middle|\; \begin{array}{c} x := (x_I \| x_{II});\; w := (w_I \| w_{II}) \\ \mathsf{srs} \leftarrow \mathsf{Setup}(\mathcal{R});\; \pi \leftarrow \mathcal{P}(\mathsf{srs}, x, w); \\ \widetilde{\mathsf{srs}} \leftarrow \mathsf{Setup}_{\mathsf{split}}(\mathcal{R}, \mathcal{X}_{II}, \mathcal{W}_{II}); \\ \mathsf{aux} \leftarrow \mathcal{P}_I(\widetilde{\mathsf{srs}}, x_I, w_I); \\ \pi' \leftarrow \mathcal{P}_{II}(\widetilde{\mathsf{srs}}, x_{II}, w_{II}, \mathsf{aux}) \end{array}\right] = 1,$$

for every set of possible indices $\mathcal{X}_{\text{II}}$ and $\mathcal{W}_{\text{II}}$, every statement x, and every witness w.

Split Zero-Knowledge. We now define the notion of split zero-knowledge. Formally, fix a relation $\mathcal{R}$ decided by a circuit C. Let $\mathcal{X}_{\text{II}}$ and $\mathcal{W}_{\text{II}}$ be sets of indices specifying the parts of the statement and the witness of phase II. Let C_{I} be the (maximal) subcircuit of C where all wires can be determined by the parts of the statement and the witness of phase I. We say Π_{split} is perfect split zero-knowledge for $\mathcal{R}$ with respect to $\mathcal{X}_{\text{II}}$ and $\mathcal{W}_{\text{II}}$, if there exists a simulator $\mathcal{S}$ such that for every security parameter $\lambda \in \mathbb{N}$, every statement-witness pair $(x = (x_{\text{I}}, x_{\text{II}}), w = (w_{\text{I}}, w_{\text{II}})) \in \mathcal{R}$, and for every computationally unbounded adversary $\mathcal{A}$:

$$\Pr\left[\ \mathcal{A}(\mathsf{aux}, \widetilde{\mathsf{srs}}) = 1 \ \middle|\ \begin{array}{c} \widetilde{\mathsf{srs}} \leftarrow \mathsf{Setup}_{\mathsf{split}}(\mathcal{R}, \mathcal{X}_{\text{II}}, \mathcal{W}_{\text{II}}); \\ \mathsf{aux} \leftarrow \mathcal{P}_{\text{I}}(\widetilde{\mathsf{srs}}, x_{\text{I}}, w_{\text{I}}) \end{array} \right]$$
$$= \Pr\left[\ \mathcal{A}(\mathsf{aux}, \widetilde{\mathsf{srs}}) = 1 \ \middle|\ (\widetilde{\mathsf{srs}}, \mathsf{aux}) \leftarrow \mathcal{S}(\mathcal{R}, x, \mathcal{X}_{\text{II}}, \mathcal{W}_{\text{II}}, C_{\text{I}}(x_{\text{I}}, w_{\text{I}}))\ \right]$$

To give an intuition of why $C_{\text{I}}(x_{\text{I}}, w_{\text{I}})$ cannot be avoided to be leaked, we elaborate on how an arithmetic circuit could be split into two parts. Assume that we have available some inputs of the circuit. We execute the circuit and obtain all the wires that can be possibly obtained, forming the first-phase extended witness z_{I}. Then at the second phase the $\mathcal{P}_{\text{II}}$ gets the rest of the input of the circuit. In order to even execute the circuit and compute the rest of the wires, forming the second-phase extended witness z_{II} they need the 'output' wires of the first phase, that we call $C_{\text{I}}(x_{\text{I}}, w_{\text{I}})$.

Remark 1. $\widetilde{\mathsf{srs}}$ *in fact consists of* $\widetilde{\mathsf{srs}}_I$ *that is inputed to the first-phase prover* $\mathcal{P}_I$ *and* $\widetilde{\mathsf{srs}}_{II}$ *taken as input by the second-phase prover* $\mathcal{P}_{II}$*. In order to avoid overwhelming the notation we write both as* $\widetilde{\mathsf{srs}}$.

Remark 2. *We highlight that Split Zero-Knowledge does not imply 'conventional' Zero-Knowledge. Intuitively, Split Zero-Knoweledge is for* aux*, the information passed from* $\mathcal{P}_I$ *to* $\mathcal{P}_{II}$ *and 'conventional' Zero-Knowledge is for the final proof* π*. The final proof of a Split Prover (zk)SNARK may or may not satisfy 'conventional' zero-knowledge, following the initial (zk)SNARK and is orthogonal to our Split Prover definition.*

4 Split Prover for Groth16

Fix a relation $\mathcal{R}$ decided by a circuit C. Let $\mathcal{X}_{\text{II}}$ and $\mathcal{W}_{\text{II}}$ be sets of indices specifying the parts of the statement and the witness of the second prover, $\mathcal{P}_{\text{II}}$. Let C_{I} be the (maximal) subcircuit of C where all wires can be determined by the parts of the statement and the witness of the first prover, $\mathcal{P}_{\text{I}}$. We can write $C = C_{\text{II}}(x_{\text{II}}, w_{\text{II}}, C_{\text{I}}(x_{\text{I}}, w_{\text{I}}))$ for some circuit C_{II}. In this section we show that Groth16, as it is, admits a split prover which in addition to satisfying the split correctness notion it also satisfies split zero-knowledge.

For the rest of this section, instead of considering circuits we focus on R1CS instances. In this representation, the first and second components of the circuits correspond to the parts of the extended witness that can be computed from the phase I and phase II witnesses correspondingly and also the parts of the matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ that depend on the two parts of the circuit. Furthermore, when the sets of indices are implicit in the context we do not include them as an input to the algorithms.

4.1 Overview of the Protocol

W.l.o.g. let $\boldsymbol{z}_{\text{I}} = (z_1, \ldots, z_{m_1})$ be the part of the extended witness that is known to the prover during the first phase, i.e., $\boldsymbol{z}_{\text{I}}$ contains the known part of the statement and the witness.[7] We assume that the

[7]In terms of arithmetic circuit, this can be thought of as all the wires that can be computed without the unknown part.

first ℓ_1 positions of the extended witness contain the part of the statement that is known in the phase I, $x_{\mathrm{I}} = (x_1, \ldots, x_{\ell_1})$, i.e. $z_i = x_i$ for each $i \in \{1, \ldots, \ell_1\}$. Similarly $\boldsymbol{z}_{\mathrm{II}} = (z_{m_1+1}, \ldots, z_m)$ is the extended witness that cannot be initially computed and the positions $m_1 + 1, \ldots, m_1 + \ell_2$ contain x_{II}. We use $m_2 := m - m_1$ to denote the size of $\boldsymbol{z}_{\mathrm{II}}$. Precisely, we write:

$$\boldsymbol{z} = (\boldsymbol{z}_{\mathrm{I}}, \boldsymbol{z}_{\mathrm{II}}) = (\overbrace{\underbrace{x_1, \ldots, x_{\ell_1}}_{\boldsymbol{x}_{\mathrm{I}}}, z_{\ell_1+1}, \ldots, z_{m_1}}^{\boldsymbol{z}_{\mathrm{I}}}, \overbrace{\underbrace{x_{m_1+1}, \ldots, x_{m_1+\ell_2}}_{\boldsymbol{x}_{\mathrm{II}}}, z_{m_1+\ell_2+1}, \ldots, z_m}^{\boldsymbol{z}_{\mathrm{II}}})$$

and we define the corresponding sets of indices:

$$\begin{array}{lll} \mathcal{Z}_{\mathrm{I}} = [1, m_1], & \mathcal{X}_{\mathrm{I}} = [1, \ell_1], & \mathcal{W}_{\mathrm{I}} = [\ell_1 + 1, m_1] \\ \mathcal{Z}_{\mathrm{II}} = [m_1 + 1, m], & \mathcal{X}_{\mathrm{II}} = [m_1 + 1, m_1 + \ell_2], & \mathcal{W}_{\mathrm{II}} = [m_1 + \ell_2 + 1, m] \end{array}$$

Intuitively the first row is the set of indices of the phase I and the second row the set of indices of the phase II.

4.1.1 Split-R1CS.

Assume a rank-1-constraint-satisfiability system $\boldsymbol{Az} \circ \boldsymbol{Bz} = \boldsymbol{Cz}$. The R1CS can be written accordingly:

$$\left(\begin{array}{c|c} \boldsymbol{A}_{11} & 0 \\ 0 & \boldsymbol{A}_{22} \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} \end{array}\right) \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \end{array}\right) \circ \left(\begin{array}{c|c} \boldsymbol{B}_{11} & 0 \\ 0 & \boldsymbol{B}_{22} \\ \boldsymbol{B}_{31} & \boldsymbol{B}_{32} \end{array}\right) \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \end{array}\right) = \left(\begin{array}{c|c} \boldsymbol{C}_{11} & 0 \\ 0 & \boldsymbol{C}_{22} \\ \boldsymbol{C}_{31} & \boldsymbol{C}_{32} \end{array}\right) \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \end{array}\right)$$

where $\boldsymbol{A}_{11} \in \mathbb{Z}_p^{n_1 \times m_1}$ are the constraints on $\boldsymbol{z}_{\mathrm{I}}$ but not on $\boldsymbol{z}_{\mathrm{II}}$, conversely $\boldsymbol{A}_{22} \in \mathbb{Z}_p^{n_2 \times m_2}$ are the constraints on $\boldsymbol{z}_{\mathrm{II}}$ but not on $\boldsymbol{z}_{\mathrm{I}}$ and $\boldsymbol{A}_{31} \in \mathbb{Z}_p^{n_3 \times m_1}$, $\boldsymbol{A}_{32} \in \mathbb{Z}_p^{n_3 \times m_2}$ involve both. Similarly, for $\boldsymbol{B}_{11} \in \mathbb{Z}_p^{n'_1 \times m_1}$, $\boldsymbol{B}_{22} \in \mathbb{Z}_p^{n'_2 \times m_2}$, $\boldsymbol{B}_{31} \in \mathbb{Z}_p^{n'_3 \times m_1}$, $\boldsymbol{B}_{32} \in \mathbb{Z}_p^{n'_3 \times m_2}$ and $\boldsymbol{C}_{11} \in \mathbb{Z}_p^{n''_1 \times m_1}$, $\boldsymbol{C}_{22} \in \mathbb{Z}_p^{n''_2 \times m_2}$, $\boldsymbol{C}_{31} \in \mathbb{Z}_p^{n''_3 \times m_1}$, $\boldsymbol{C}_{32} \in \mathbb{Z}_p^{n''_3 \times m_2}$. We note that n_i, n'_i, n''_i may not necessarily be the same.

We can re-write the above system as:

$$\left[\overbrace{\begin{pmatrix} \boldsymbol{A}_{11} \\ 0 \\ \boldsymbol{A}_{31} \end{pmatrix}}^{\boldsymbol{A}_{\mathrm{I}}} \boldsymbol{z}_{\mathrm{I}} + \overbrace{\begin{pmatrix} 0 \\ \boldsymbol{A}_{22} \\ \boldsymbol{A}_{32} \end{pmatrix}}^{\boldsymbol{A}_{\mathrm{II}}} \boldsymbol{z}_{\mathrm{II}}\right] \circ \left[\overbrace{\begin{pmatrix} \boldsymbol{B}_{11} \\ 0 \\ \boldsymbol{B}_{31} \end{pmatrix}}^{\boldsymbol{B}_{\mathrm{I}}} \boldsymbol{z}_{\mathrm{I}} + \overbrace{\begin{pmatrix} 0 \\ \boldsymbol{B}_{22} \\ \boldsymbol{B}_{32} \end{pmatrix}}^{\boldsymbol{B}_{\mathrm{II}}} \boldsymbol{z}_{\mathrm{II}}\right] =$$

$$= \left[\overbrace{\begin{pmatrix} \boldsymbol{C}_{11} \\ 0 \\ \boldsymbol{C}_{31} \end{pmatrix}}^{\boldsymbol{C}_{\mathrm{I}}} \boldsymbol{z}_{\mathrm{I}} + \overbrace{\begin{pmatrix} 0 \\ \boldsymbol{C}_{22} \\ \boldsymbol{C}_{32} \end{pmatrix}}^{\boldsymbol{C}_{\mathrm{II}}} \boldsymbol{z}_{\mathrm{II}}\right]$$

or equivalently

$$\begin{aligned} \big(\boldsymbol{C}_{\mathrm{I}} \cdot \boldsymbol{z}_{\mathrm{I}}\big) + \big(\boldsymbol{C}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{II}}\big) =& \big(\boldsymbol{A}_{\mathrm{I}} \cdot \boldsymbol{z}_{\mathrm{I}}\big) \circ \big(\boldsymbol{B}_{\mathrm{I}} \cdot \boldsymbol{z}_{\mathrm{I}}\big) + \big(\boldsymbol{A}_{\mathrm{I}} \cdot \boldsymbol{z}_{\mathrm{I}}\big) \circ \big(\boldsymbol{B}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{II}}\big) \\ &+ \big(\boldsymbol{A}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{II}}\big) \circ \big(\boldsymbol{B}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{I}}\big) + \big(\boldsymbol{A}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{II}}\big) \circ \big(\boldsymbol{B}_{\mathrm{II}} \cdot \boldsymbol{z}_{\mathrm{II}}\big) \end{aligned}$$

We refer to this form as the 'Split-R1CS'.

4.1.2 Split Proof Computation.

To begin with, from the available statement x_{I} and witness w_{I} the first prover can compute their extended witness $\boldsymbol{z}_{\mathrm{I}}$ by computing all the wires of the circuit that are possible with x_{I} and w_{I}. Then the second prover having x_{II} and w_{II} can compute their extended witness, i.e. the rest of the wires of the circuit, given $C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}})$ which corresponds to the output wires of the subcircuit that was computed by $\mathcal{P}_{\mathrm{I}}$ (see the discussion at the beginning of the section). Therefore, the first part of the auxiliary information that needs to be passed from $\mathcal{P}_{\mathrm{I}}$ to $\mathcal{P}_{\mathrm{II}}$ is $\mathsf{aux}_0 = C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}})$.

As discussed in Section 2.3, a Groth16 proof consists of three group elements π_1, π_2, π_3.

The first group element of the proof, π_1, can be written as:

$$\pi_1 = [\alpha]_1 + \overbrace{\sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [a_i(\tau)]_1}^{\mathsf{aux}_1} + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1 + r[\delta]_1.$$

Therefore the value $\mathsf{aux}_1 := \sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [a_i(\tau)]_1$ can be fully computed in the phase I, as it depends only on $\boldsymbol{z}_{\mathrm{I}}$. Given this, the final π_1 can be computed in phase II as $\pi_1 = \mathsf{aux}_1 + [\alpha]_1 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1 + r[\delta]_1$, once $\boldsymbol{z}_{\mathrm{II}}$ becomes available.

The same argument holds for π_2:

$$\pi_2 = [\beta]_2 + \overbrace{\sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_2}^{\mathsf{aux}_2} + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_2 + s[\delta]_2.$$

where $\mathsf{aux}_2 := \sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_2$

For the third group element π_3 in the proof, we have:

$$\begin{aligned}
\pi_3 = & \overbrace{\sum_{i \in \mathcal{W}_{\mathrm{I}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta} \right]_1}^{\mathsf{aux}_3} + \sum_{i \in \mathcal{W}_{\mathrm{II}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta} \right]_1 \\
& + s \left(\overbrace{\sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [a_i(\tau)]_1}^{\mathsf{aux}_1} + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1 \right) + r \left(\overbrace{\sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_1}^{\mathsf{aux}_4} + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_1 \right) \\
& + \sum_{i=0}^{n-2} \tilde{q}_i \left[\frac{V(\tau)\tau^i}{\delta} \right]_1 + s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1.
\end{aligned}$$

For this, we need to compute the quotient polynomial $q(X) := \left\lfloor \frac{\left(\sum_{i=1}^m z_i a_i(X)\right) \cdot \left(\sum_{i=1}^m z_i b_i(X)\right) - \sum_{i=1}^m z_i c_i(X)}{V(X)} \right\rfloor$ that depends on both $\boldsymbol{z}_{\mathrm{I}}$ and $\boldsymbol{z}_{\mathrm{II}}$. To this end, our first observation is that, following the split R1CS described above, the quotient polynomial can be re-written as:

$$q(X) = \Bigg\lfloor \frac{\left(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i b_i(X)\right)}{V(X)} + \\ + \frac{\left(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i b_i(X)\right)}{V(X)} + \\ + \frac{\left(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i b_i(X)\right)}{V(X)} + \\ + \frac{\left(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i b_i(X)\right)}{V(X)} + \\ - \frac{\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i c_i(X)}{V(X)} - \frac{\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i c_i(X)}{V(X)} \Bigg\rfloor$$

Our second observation is that the quotient polynomial q is equal to the sum of the six partial quotients (i.e., we can ignore the partial remainders in the above six terms). We formally present this claim in the following lemma.

Lemma 1. *Let $f_1(X), \ldots, f_t(X), g(X)$ be univariate polynomials in $\mathbb{Z}_p[X]$ and $k_1, \ldots, k_t$ be field elements in $\mathbb{Z}_p$. Then $\left\lfloor \frac{\sum_{i=1}^{t} k_i f_i(X)}{g(X)} \right\rfloor = \sum_{i=1}^{t} k_i \left\lfloor \frac{f_i(X)}{g(X)} \right\rfloor$.*

Proof. Let the euclidean division of f_i by g be $f_i(X) = q_i(X)g(X) + r_i(X)$, where $\deg(r_i) < \deg(g)$, for each $i \in [t]$. Similarly $f(X) = q(X)g(X) + r(X)$, where $\deg(r) < \deg(g)$. Then,

$$\sum_{i=1}^{t} k_i f_i(X) = \sum_{i=1}^{t} [k_i q_i(X) g(X) + k_i r_i(X)] \\ = \left[\sum_{i=1}^{t} k_i q_i(X)\right] g(X) + \left[\sum_{i=1}^{t} k_i r_i(X)\right]$$

where, $\deg(\sum_{i=1}^{t} k_i r_i) < \deg(g)$, since $\deg(r_i) < \deg(g)$ for each $i \in [t]$ and k_i's are constants (degree 0). Therefore, $q(X) = \sum_{i=1}^{t} k_i q_i(X)$. □

Finally, notice that $\deg(c_i) < \deg(V)$ for each $i \in [m]$, hence $\left\lfloor \frac{\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i c_i(X)}{V(X)} \right\rfloor = \left\lfloor \frac{\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i c_i(X)}{V(X)} \right\rfloor = 0$.

Therefore, the quotient polynomial is actually a sum of four terms:

$$\begin{aligned}
q(X) =& \left\lfloor \frac{\Big(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i a_i(X)\Big)\cdot\Big(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i b_i(X)\Big)}{V(X)} \right\rfloor + \\
&+ \left\lfloor \frac{\Big(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i a_i(X)\Big)\cdot\Big(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i b_i(X)\Big)}{V(X)} \right\rfloor + \\
&+ \left\lfloor \frac{\Big(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i a_i(X)\Big)\cdot\Big(\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i b_i(X)\Big)}{V(X)} \right\rfloor + \\
&+ \left\lfloor \frac{\Big(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i a_i(X)\Big)\cdot\Big(\sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i b_i(X)\Big)}{V(X)} \right\rfloor \\
:=& q_1(X) + q_2(X) + q_3(X) + q_4(X)
\end{aligned}$$

For notational convenience and to make the dependence on $\boldsymbol{z}_{\mathrm{I}}$ and $\boldsymbol{z}_{\mathrm{II}}$ clear we re-write the sums in the above as inner products:

$$\begin{aligned}
q(X) =& \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\big\rangle}{V(X)} \right\rfloor + \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{b}_{\mathrm{II}}(X)\big\rangle}{V(X)} \right\rfloor + \\
&+ \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{a}_{\mathrm{II}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\big\rangle}{V(X)} \right\rfloor + \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{a}_{\mathrm{II}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{b}_{\mathrm{II}}(X)\big\rangle}{V(X)} \right\rfloor
\end{aligned}$$

where $\boldsymbol{a}(X) = (\boldsymbol{a}_{\mathrm{I}}(X) \| \boldsymbol{a}_{\mathrm{II}}(X))^\top = (a_1(X), \ldots, a_{m_1}(X), a_{m_1+1}(X), \ldots, a_m(X))^\top$ and $\boldsymbol{b}(X) = (\boldsymbol{b}_{\mathrm{I}}(X) \| \boldsymbol{b}_{\mathrm{II}}(X))^\top = (b_1(X), \ldots, b_{m_1}(X), b_{m_1+1}(X), \ldots, b_m(X))^\top$.

Now, notice that since the first term is entirely computable in phase I, the first prover $\mathcal{P}_{\mathrm{I}}$ computes the first quotient polynomial $q_1(X)$ and sets $\mathsf{aux}_5 = \sum_{i=0}^{n-2} \tilde{q}_{1,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]$. The second and third terms depend on both $\boldsymbol{z}_{\mathrm{I}}$ and $\boldsymbol{z}_{\mathrm{II}}$. We re-write these terms as:

$$\begin{aligned}
q_2(X) =& \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{b}_{\mathrm{II}}(X)\big\rangle}{V(X)} \right\rfloor = \left\lfloor \frac{\Big\langle \boldsymbol{z}_{\mathrm{II}}, \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\big\rangle \cdot \boldsymbol{b}_{\mathrm{II}}(X)\Big\rangle}{V(X)} \right\rfloor \\
\overset{\text{Lemma 1}}{=}& \left\langle \boldsymbol{z}_{\mathrm{II}}, \overbrace{\left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\big\rangle \cdot \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor}^{\boldsymbol{\mu}_2(X)} \right\rangle := \big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{\mu}_2(X)\big\rangle \\
q_3(X) =& \left\lfloor \frac{\big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{a}_{\mathrm{II}}(X)\big\rangle \cdot \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\big\rangle}{V(X)} \right\rfloor = \left\lfloor \frac{\Big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{a}_{\mathrm{II}}(X) \cdot \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\big\rangle\Big\rangle}{V(X)} \right\rfloor \\
\overset{\text{Lemma 1}}{=}& \left\langle \boldsymbol{z}_{\mathrm{II}}, \overbrace{\left\lfloor \frac{\boldsymbol{a}_{\mathrm{II}}(X) \cdot \big\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\big\rangle}{V(X)} \right\rfloor}^{\boldsymbol{\mu}_3(X)} \right\rangle := \big\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{\mu}_3(X)\big\rangle.
\end{aligned}$$

Now prover I proceeds as follows: Computes the vectors of m_2 polynomials $\boldsymbol{\mu}_2(X) = \left\lfloor \frac{\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}_{\mathrm{I}}(X)\rangle \cdot \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor$ and $\boldsymbol{\mu}_3(X) = \left\lfloor \frac{\boldsymbol{a}_{\mathrm{II}}(X)\cdot \langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}_{\mathrm{I}}(X)\rangle}{V(X)} \right\rfloor$ and sets $\mathbf{aux}_6 = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{2,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ and $\mathbf{aux}_7 = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{3,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$, each consisting of m_2 group elements. In the second phase $\mathcal{P}_{\mathrm{II}}$ computes the multi-exponentiations $\langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_6\rangle$ and $\langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_7\rangle$ to reconstruct $\left[\frac{q_2(\tau)V(\tau)}{\delta}\right]_1$ and $\left[\frac{q_3(\tau)V(\tau)}{\delta}\right]_1$ respectively.

The fourth term of $q_4(X)$ can be fully computed in the second phase by $\mathcal{P}_{\mathrm{II}}$.

In conclusion the final π_3 can be computed in the phase II as follows:[8]

$$\begin{aligned}\pi_3 = &\left(\mathsf{aux}_3 + \sum_{i\in\mathcal{W}_{\mathrm{II}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1\right)\\ &+ s\left(\mathsf{aux}_1 + \sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1\right) + r\left(\mathsf{aux}_4 + \sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_1\right)\\ &+\left(\mathsf{aux}_5 + \langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_6\rangle + \langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_7\rangle + \sum_{i=0}^{n-2} \tilde{q}_{4,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1\right) + s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1.\end{aligned}$$

4.1.3 Optimizing $\mathcal{P}_{\mathrm{II}}$ for small witnesses, $m_2 = o(\sqrt{n\log n + m})$.

$\mathcal{P}_{\mathrm{II}}$'s running time, as described above, is dominated by the computation of $q_4(X)$ which, being a polynomial division of degree n, requires $O(n\log n)$ time.

We observe that if the most significant portion of the witness is in the first phase, i.e. the phase II extended witness is small, then there is a more efficient mechanism for $\mathcal{P}_{\mathrm{II}}$. In concrete, if $m_2 = o(\sqrt{n\log n + m})$, then we preprocess the polynomials as follows:

$$\begin{aligned}q_4(X) =&\left\lfloor \frac{\langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{a}_{\mathrm{II}}(X)\rangle \cdot \langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{b}_{\mathrm{II}}(X)\rangle}{V(X)} \right\rfloor = \left\lfloor \frac{\boldsymbol{z}_{\mathrm{II}} \cdot (\boldsymbol{a}_{\mathrm{II}}^\top(X) \otimes \boldsymbol{b}_{\mathrm{II}}(X)) \cdot \boldsymbol{z}_{\mathrm{II}}^\top}{V(X)} \right\rfloor\\ &\overset{\text{Lemma 1}}{=} \boldsymbol{z}_{\mathrm{II}} \cdot \left\lfloor \frac{\boldsymbol{a}_{\mathrm{II}}^\top(X) \otimes \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor \cdot \boldsymbol{z}_{\mathrm{II}}^\top.\end{aligned}$$

Let $\boldsymbol{T}(X) = \left\lfloor \frac{\boldsymbol{a}_{\mathrm{II}}^\top(X)\otimes \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor$ be a $(m_2 \times m_2)$-size matrix of polynomials. In the split setup phase we compute the matrix containing $(m_2 \times m_2)$ group elements $\boldsymbol{H} = \sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ and publish it in $\widetilde{\mathsf{srs}}$. Thereafter, in the second phase the prover II computes $\boldsymbol{z}_{\mathrm{II}}\boldsymbol{H}\boldsymbol{z}_{\mathrm{II}}^\top$ to reconstruct $\left[\frac{q_4(\tau)V(\tau)}{\delta}\right]_1$.

[8] Note that a for a concrete improvement on the size of aux we can merge $\mathbf{aux}_6$ and $\mathbf{aux}_7$ into $\mathbf{aux}_6 + \mathbf{aux}_7$. For more intuitive presentation of our protocol we stick to separate $\mathbf{aux}_6$ and $\mathbf{aux}_7$.

The the final π_3 can be alternatively computed by $\mathcal{P}_{\mathrm{II}}$ as:

$$\begin{aligned}\pi_3 = &\left(\mathsf{aux}_3 + \sum_{i \in \mathcal{W}_{\mathrm{II}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1\right) \\ &+ s\left(\mathsf{aux}_1 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1\right) + r\left(\mathsf{aux}_4 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_1\right) \\ &+ \left(\mathsf{aux}_5 + \langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_6 \rangle + \langle \boldsymbol{z}_{\mathrm{II}}, \mathbf{aux}_7 \rangle + \boldsymbol{z}_{\mathrm{II}} \boldsymbol{H} \boldsymbol{z}_{\mathrm{II}}^\top\right) + s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1,\end{aligned}$$

taking time $O(m_2^2)$, which is less than $O(n \log n)$.

4.1.4 Split Zero-Knowledge.

Until now we have seen how to obtain a split prover for Groth16 that satisfies correctness, ignoring the split zero-knowledge property. In order to add split zero-knowledge to the above we proceed as follows: Assume that we want to build a split prover for the R1CS relation

$$\mathcal{R} = \left\{(\boldsymbol{x}; \boldsymbol{w}) : \boldsymbol{A}\boldsymbol{z} \circ \boldsymbol{B}\boldsymbol{z} = \boldsymbol{C}\boldsymbol{z} \wedge \boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w})\right\},$$

then, we show a construction with split zero-knowledge for the relation

$$\mathcal{R}' = \left\{(\boldsymbol{x}; (\boldsymbol{w}, \boldsymbol{r})) : \boldsymbol{A}'\boldsymbol{z} \circ \boldsymbol{B}'\boldsymbol{z} = \boldsymbol{C}'\boldsymbol{z} \wedge \boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w} \| \boldsymbol{r})\right\},$$

where $\mathcal{R}'$ defined as follows:

$$\forall \boldsymbol{x}, \boldsymbol{w}, \boldsymbol{r} : (\boldsymbol{x}; (\boldsymbol{w}, \boldsymbol{r})) \in \mathcal{R}' \iff (\boldsymbol{x}; \boldsymbol{w}) \in \mathcal{R}.$$

Therefore, the two relations are functionally equivalent as for any $\boldsymbol{x}$, $\boldsymbol{w}$, $\boldsymbol{r}$ can be seen as a dummy witness that is present solely to achieve the zero-knowledge property.

The idea is to carefully add some extra constraints in the R1CS and wires in the extended witness. Then the extra wires are going to be sampled uniformly at random from $\mathcal{P}_{\mathrm{I}}$ in order to 'mask' the auxiliary information. Similar approaches for achieving zero-knowledge can be found in the literature (e.g. [AHIV17]).

In more detail, the new R1CS matrices will be:

$$\overbrace{\left(\begin{array}{c|c|cccc} \boldsymbol{A}_{11} & \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \boldsymbol{A}_{22} & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} & 0 & 0 & 0 & 0 \\ \hline 0\ldots0 & 0\ldots0 & 1 & 0 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 0 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 0 & 1 & 0 \end{array}\right)}^{\boldsymbol{A}'} \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \\ \hline r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\right) \circ \overbrace{\left(\begin{array}{c|c|cccc} \boldsymbol{B}_{11} & \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \boldsymbol{B}_{22} & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{B}_{31} & \boldsymbol{B}_{32} & 0 & 0 & 0 & 0 \\ \hline 0\ldots0 & 0\ldots0 & 0 & 0 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 1 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 0 & 1 & 0 \end{array}\right)}^{\boldsymbol{B}'} \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \\ \hline r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\right)$$

$$= \overbrace{\left(\begin{array}{c|c|cccc} \boldsymbol{C}_{11} & \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \boldsymbol{C}_{22} & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{C}_{31} & \boldsymbol{C}_{32} & 0 & 0 & 0 & 0 \\ \hline 0\ldots0 & 0\ldots0 & 0 & 0 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 0 & 0 & 0 \\ 0\ldots0 & 0\ldots0 & 0 & 0 & 0 & 1 \end{array}\right)}^{\boldsymbol{C}'} \left(\begin{array}{c} \boldsymbol{z}_{\mathrm{I}} \\ \hline \boldsymbol{z}_{\mathrm{II}} \\ \hline r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}\right)$$

Equivalently for the corresponding polynomials we get:

- $\boldsymbol{a}'_{\mathrm{I}}(X) = \big(a_1(X), \ldots, a_{m_1}(X), L_{n+1}(X), 0, L_{n+3}(X), 0\big)^\top, \quad \boldsymbol{a}'_{\mathrm{II}}(X) = \boldsymbol{a}_{\mathrm{II}}$
- $\boldsymbol{b}'_{\mathrm{I}}(X) = \big(b_1(X), \ldots, b_{m_1}(X), 0, L_{n+2}(X), L_{n+3}(X), 0\big)^\top, \quad \boldsymbol{b}'_{\mathrm{II}}(X) = \boldsymbol{b}_{\mathrm{II}}$

In the modfied R1CS $n' = n + 3$ and $m' = m + 4$.

We note that this approach is not compatible with an already existing Groth16 srs for $\mathcal{R}$ and one should run a new setup, $\mathsf{Setup}(\mathcal{R}')$ for $\mathcal{R}'$, where $\mathcal{R}'$ is characterized by the above R1CS. The latter is in fact happening in $\mathsf{Setup}_{\mathsf{split}}$.

4.2 The protocol

Here we describe our protocol formally. We recall the notation:

- $a_i(X), b_i(X), c(X)$ are (publicly) known polynomial that interpolate the i-th column of the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ R1CS matrices respectively.
- $\boldsymbol{a}(X) := (\boldsymbol{a}_{\mathrm{I}}(X) \| \boldsymbol{a}_{\mathrm{II}}(X))^\top = (a_1(X), \ldots, a_{m_1}(X), a_{m_1+1}(X), \ldots, a_m(X))^\top$
- $\boldsymbol{b}(X) := (\boldsymbol{b}_{\mathrm{I}}(X) \| \boldsymbol{b}_{\mathrm{II}}(X))^\top = (b_1(X), \ldots, b_{m_1}(X), b_{m_1+1}(X), \ldots, b_m(X))^\top$
- $\boldsymbol{a}'_{\mathrm{I}}(X) = \big(a_1(X), \ldots, a_{m_1}(X), L_{n+1}(X), 0, L_{n+3}(X), 0\big)^\top, \quad \boldsymbol{a}'_{\mathrm{II}}(X) = \boldsymbol{a}_{\mathrm{II}}$
- $\boldsymbol{b}'_{\mathrm{I}}(X) = \big(b_1(X), \ldots, b_{m_1}(X), 0, L_{n+2}(X), L_{n+3}(X), 0\big)^\top, \quad \boldsymbol{b}'_{\mathrm{II}}(X) = \boldsymbol{b}_{\mathrm{II}}$
- $n' = n + 3$ and $m' = m + 4$.

$\underline{\mathsf{Setup}_{\mathsf{split}}(\mathcal{R}, \mathcal{X}_{\mathbf{II}}, \mathcal{W}_{\mathbf{II}}) \to \mathsf{srs}}$: First it runs Groth16's setup for the relation $\mathcal{R}'$. That is, samples uniformly $\tau, \alpha, \beta, \gamma, \delta \leftarrow\!\!\$\ \mathbb{Z}_p$ and outputs:

$$\widetilde{\mathsf{srs}} = \Bigg\{ \Big\{ [\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_1, [\delta]_2, \Big\{ \left[\tau^i\right]_1, \left[\tau^i\right]_2 \Big\}_{i=0}^{n'-1}, \left\{ \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 \right\}_{i=0}^{n'-2},$$
$$\Big\{ [a_i(\tau)]_1, [b_i(\tau)]_1, [b_i(\tau)]_2 \Big\}_{i=1}^{m'}, \left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma}\right]_1 \right\}_{i \in \mathcal{X}_{\mathrm{I}} \cup \mathcal{X}_{\mathrm{II}}},$$
$$\left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1 \right\}_{i \in \mathcal{W}_{\mathrm{I}} \cup \mathcal{W}_{\mathrm{II}}} \Bigg\}$$

Then if $m_2 = o(\sqrt{n \log n + m})$: First computes the matrix of polynomials:

$$\boldsymbol{T}(X) = \left\lfloor \frac{\boldsymbol{a}_{\mathrm{II}}^\top(X) \otimes \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor := \big(t_{i,j}(X)\big)_{i,j \in \mathcal{Z}_{\mathrm{II}}}$$

and using the $\left\{ \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 \right\}_{i=0}^{n-2}$ of the srs outputs the corresponding matrix of group elements:

$$\boldsymbol{H} = \left(\sum_{k=0}^{n-2} \tilde{t}_{i,j,k} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 \right)_{i,j \in \mathcal{Z}_{\mathrm{II}}}$$

and appends it to the structured reference string, i.e. $\widetilde{\mathsf{srs}} \leftarrow \widetilde{\mathsf{srs}} \cup \boldsymbol{H}$.

$\underline{\mathcal{P}_{\mathbf{I}}(\widetilde{\mathsf{srs}}, \boldsymbol{x}_{\mathbf{I}}, \boldsymbol{w}_{\mathbf{I}}) \to \mathsf{aux}}$**:** The prover I samples $r_1, r_2, r_3 \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $r_4 = r_3^2$ and computes:

0. $\mathsf{aux}_0 = C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}})$,
1. $\mathsf{aux}_1 = \sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [a_i(\tau)]_1 + r_1 [a_{m+1}(\tau)]_1 + r_3 [a_{m+3}(\tau)]_1$,
2. $\mathsf{aux}_2 = \sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_2 + r_2 [b_{m+2}(\tau)]_2 + r_3 [b_{m+3}(\tau)]_1$,
3.
$$\begin{aligned}\mathsf{aux}_3 = \sum_{i \in \mathcal{W}_{\mathrm{I}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1 + r_1 \left[\frac{\beta a_{m+1}(\tau)}{\delta}\right]_1 + r_3 \left[\frac{\beta a_{m+3}(\tau)}{\delta}\right]_1 \\ + r_2 \left[\frac{\alpha b_{m+2}(\tau)}{\delta}\right]_1 + r_3 \left[\frac{\alpha b_{m+3}(\tau)}{\delta}\right]_1 + r_4 \left[\frac{c_{m+3}(\tau)}{\delta}\right]_1,\end{aligned}$$
4. $\mathsf{aux}_4 = \sum_{i \in \mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_1 + r_2 [b_{m+2}(\tau)]_1 + r_3 [b_{m+3}(\tau)]_1$,
5. $\mathsf{aux}_5 = \sum_{i=0}^{n-2} \tilde{q}_{1,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ where $q_1(X) = \left\lfloor \frac{\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}'_{\mathrm{I}}(X)\rangle \cdot \langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}'_{\mathrm{I}}(X)\rangle}{V(X)} \right\rfloor$,
6. $\boldsymbol{\mathsf{aux}}_6 = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{2,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ where $\boldsymbol{\mu}_2(X) = \left\lfloor \frac{\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{a}'_{\mathrm{I}}(X)\rangle \boldsymbol{b}_{\mathrm{II}}(X)}{V(X)} \right\rfloor$,
7. $\boldsymbol{\mathsf{aux}}_7 = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{3,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ where $\boldsymbol{\mu}_3(X) = \left\lfloor \frac{\langle \boldsymbol{z}_{\mathrm{I}}, \boldsymbol{b}'_{\mathrm{I}}(X)\rangle \boldsymbol{a}_{\mathrm{II}}(X)}{V(X)} \right\rfloor$

and outputs $\mathsf{aux} := \{\mathsf{aux}_1, \mathsf{aux}_2, \mathsf{aux}_3, \mathsf{aux}_4, \mathsf{aux}_5, \boldsymbol{\mathsf{aux}}_6, \boldsymbol{\mathsf{aux}}_7\}$

$\underline{\mathcal{P}_{\mathbf{II}}(\widetilde{\mathsf{srs}}, \boldsymbol{x}_{\mathbf{II}}, w_{\mathbf{II}}, \mathsf{aux}) \to \pi}$**:** The prover II first computes $\boldsymbol{z}_{\mathrm{II}}$ given $\boldsymbol{x}_{\mathrm{II}}$, w_{II} and aux_0. Then uniformly samples $r, s \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes:

1. $\pi_1 = [\alpha]_1 + \mathsf{aux}_1 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1 + r[\delta]_1$,
2. $\pi_2 = [\beta]_2 + \mathsf{aux}_2 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_2 + s[\delta]_2$,
3.
$$\begin{aligned}\pi_3 = \left(\mathsf{aux}_3 + \sum_{i \in \mathcal{W}_{\mathrm{II}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1\right) \\ + s\left(\mathsf{aux}_1 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [a_i(\tau)]_1\right) + r\left(\mathsf{aux}_4 + \sum_{i \in \mathcal{Z}_{\mathrm{II}}} z_i [b_i(\tau)]_1\right) \\ + \left(\mathsf{aux}_5 + \langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{\mathsf{aux}}_6\rangle + \langle \boldsymbol{z}_{\mathrm{II}}, \boldsymbol{\mathsf{aux}}_7\rangle + K\right) + s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1,\end{aligned}$$

where $K = \boldsymbol{z}_{\mathrm{II}} \boldsymbol{H} \boldsymbol{z}_{\mathrm{II}}^{\top}$ if $m_2 = o(\sqrt{n \log n + m})$ otherwise $K = \sum_{i=0}^{n-2} \tilde{q}_{4,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$.

Finally, outputs $\pi := \{\pi_1, \pi_2, \pi_3\}$

Theorem 1. *The above scheme has perfect split correctness and perfect split zero-knowledge.*

Proof. Split Correctness follows by construction. To avoid repetition we point to Section 4.1 where we extensively unveiled the protocol details.

We now show that our construction achieves perfect split zero knowledge. The simulator $\mathcal{S}$ works as follows: It samples two random polynomials of degree $n+2$, $\hat{x}(X), \hat{y}(X)$, and a random scalar $\hat{\omega} \leftarrow\!\!\$\, \mathbb{Z}_p$ and sets

- $\hat{\mathsf{aux}}_1 = [\hat{x}(\tau)]_1$,
- $\hat{\mathsf{aux}}_2 = [\hat{y}(\tau)]_2$,
- $\hat{\mathsf{aux}}_3 = \left[\frac{\beta\hat{x}(\tau) - \sum_{i\in\mathcal{X}_{\mathrm{I}}} z_i\beta a_i(\tau) + \alpha\hat{y}(\tau) - \sum_{i\in\mathcal{X}_{\mathrm{I}}} z_i\alpha b_i(\tau)}{\delta} + \hat{\omega}\right]_1$,
- $\hat{\mathsf{aux}}_4 = [\hat{y}(\tau)]_1$,
- $\hat{\mathsf{aux}}_5 = \left[\frac{\hat{x}(\tau)\hat{y}(\tau) - R_5(\tau)}{\delta}\right]_1$ where $R_5(X)$ is the remainder such that $\hat{x}(X)\hat{y}(X) = q_5(X)V(X) + R_5(X)$ and $q_5(X) := \left\lfloor \frac{\hat{x}(X)\hat{y}(X)}{V(X)} \right\rfloor$,
- $\hat{\mathbf{aux}}_6 = \left[\frac{\hat{x}(\tau)\boldsymbol{b}_{\mathrm{II}}(\tau) - \boldsymbol{R}_6(\tau)}{\delta}\right]_1$ where $\boldsymbol{R}_6(X)$ is the vector of remainders such that $\hat{x}(X)\boldsymbol{b}_{\mathrm{II}}(X) = \boldsymbol{q}_6(X)V(X) + \boldsymbol{R}_6(X)$ and $\boldsymbol{q}_6(X) := \left\lfloor \frac{\hat{x}(\tau)\boldsymbol{b}_{\mathrm{II}}(\tau)}{V(X)} \right\rfloor$,
- $\hat{\mathbf{aux}}_7 = \left[\frac{\hat{y}(\tau)\boldsymbol{a}_{\mathrm{II}}(\tau) - \boldsymbol{R}_7(\tau)}{\delta}\right]_1$ where $\boldsymbol{R}_7(X)$ is the vector of remainders such that $\hat{y}(X)\boldsymbol{a}_{\mathrm{II}}(X) = \boldsymbol{q}_7(X)V(X) + \boldsymbol{R}_7(X)$ and $\boldsymbol{q}_7(X) := \left\lfloor \frac{\hat{y}(\tau)\boldsymbol{a}_{\mathrm{II}}(\tau)}{V(X)} \right\rfloor$.

Recall that $\mathcal{S}$ samples itself the $\widetilde{\mathsf{srs}}$ so has access to the trapdoors $\alpha, \beta, \delta, \tau$. Finally aux_0 is trivially simulated, since $C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}})$ is part of its input.

Regarding correctness of the simulation, the distribution of the simulated $\hat{\mathsf{aux}}$ is identical to the one generated by the protocol. $\mathsf{aux}_1, \hat{\mathsf{aux}}_1, \mathsf{aux}_2, \hat{\mathsf{aux}}_2, \mathsf{aux}_3, \hat{\mathsf{aux}}_3$ are all uniformly distributed, since the groups $\mathbb{G}_1, \mathbb{G}_2$ are cyclic and $r_1, r_2, r_3, \hat{x}, \hat{y}, \hat{\omega}$ are uniformly random (r_4 is implicitly r_3^2). The rest of the auxiliary values are uniquely determined based on the $\widetilde{\mathsf{srs}}$ and aux_1, aux_2 in the real world. In the simulated world also, they are chosen accordingly using the $\widetilde{\mathsf{srs}}$ trapdoors and $\hat{\mathsf{aux}}_1$, $\hat{\mathsf{aux}}_2$. Hence, these are also identically distributed. □

4.3 Efficiency

Here we provide a concrete analysis of the efficiency of our scheme, namely the computational and communication complexities.

4.3.1 Computational Complexity of the algorithms

First, we define metrics for the two operations that are dominant, Fast Fourier Transforms (FFTs) and Multi-Scalar-Multiplications (MSMs).[9] $\mathsf{MSM}_i(n)$ and $\mathsf{FFT}(n)$ denote an MSM in $\mathbb{G}_i$ and FFT of n elements respectively. For ease of presentation, in MSM and FFT we ignore the additive constants that have insignificant contribution, for example for $\sum_{i=1}^{m} d_i[x_i]_1 + e[y]$ we would write $\mathsf{MSM}(m)$ instead of $\mathsf{MSM}(m+1)$.

Our first prover, $\mathcal{P}_{\mathrm{I}}$ requires $O(1)\mathsf{FFT}(n)$, $O(m_2)\mathsf{FFT}(n)$ and $O(m_2)\mathsf{FFT}(n)$ to compute the corresponding quotient polynomials $q_1(X)$, $\boldsymbol{\mu}_2(X)$, $\boldsymbol{\mu}_3(X)$ and then $(2m_2+1)\mathsf{MSM}(n)$ to compute aux_5, $\mathbf{aux}_6$, $\mathbf{aux}_7$ respectively. Additionally, $2\mathsf{MSM}_1(m_1) + \mathsf{MSM}_1(m_1 - \ell_1) + \mathsf{MSM}_2(m_1)$ to compute $\mathsf{aux}_1, \mathsf{aux}_2, \mathsf{aux}_3, \mathsf{aux}_4$. Then our second prover is performing as follows: If $m_2 = o(\sqrt{n\log n + m})$ then $(m_2+1)\mathsf{MSM}_1(m_2) + \mathsf{MSM}_1(5m_2 - \ell_2) + \mathsf{MSM}_2(m_2)$ to compute π_1, π_2 and π_3, the dominant cost being the computation of $\boldsymbol{z}_{\mathrm{II}}\boldsymbol{H}\boldsymbol{z}_{\mathrm{II}}^{\top}$, otherwise $5\mathsf{FFT}(n) + \mathsf{MSM}_1(5m_2 - \ell_2) + \mathsf{MSM}_2(m_2)$ to compute π_1, π_2 and π_3, the dominant cost being the FFTs to compute polynomial division for $q_4(X)$.

[9]MSMs are also referred to as multi-exponentiations.

4.3.2 Communication Complexity in group elements.

The sizes of the elements of our protocol in group elements precisely are: the size of the auxiliary information (ignoring $\mathsf{aux}_0 = C_{\mathrm{I}}(x_{\mathrm{I}}, w_{\mathrm{I}})$) passed to the second prover $|\mathsf{aux}| = (2m_2+4)\,|\mathbb{G}_1| + 1\,|\mathbb{G}_2|$. For srs and π, they are, again, the same as in Groth16: $|\mathsf{srs}| = (2n+3m+1)\,|\mathbb{G}_1| + (n+m+3)\,|\mathbb{G}_2| = O(n+m)$ and $|\pi| = 2\,|\mathbb{G}_1| + 1\,|\mathbb{G}_2| = O(1)$.

Remark 3. *In fact, we can consider an optimization where the second prover time and $\boldsymbol{H}$-size are both $O(\mathsf{rank}(\boldsymbol{A}_{II}) \times \mathsf{rank}(\boldsymbol{B}_{II}))$ instead of $O(m_2^2)$, where* rank *denotes the column rank. For simplicity we describe our protocols assuming that $\boldsymbol{A}_{II}$ and $\boldsymbol{B}_{II}$ are both full rank.*

5 Lower Bound on the second Prover Time in Groth16

In this section, we sketch a lower bound on the best achievable phase II prover time in any split prover scheme for the Groth16 [Gro16] proof system, thereby demonstrating that our constructions from Section 4 is asymptotically tight. Let $\mathsf{rank}(\boldsymbol{M})$ denote the column rank of matrix $\boldsymbol{M}$. At a high-level, we show that $\mathcal{P}_{\mathrm{II}}$ in any split prover variant of Groth16 must receive $\Omega\left(\mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{\mathrm{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\mathrm{II}}))\right\}\right)$ group elements as auxiliary information from the split structured reference string and the first prover. This also implicitly puts a bound on the smallest possible runtime for $\mathcal{P}_{\mathrm{II}}$ in Groth16. In particular, it shows that the second prover must perform $\Omega\left(\mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{\mathrm{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\mathrm{II}}))\right\}\right)$ group operations. Before proving our main impossibility result, we find it useful to prove the following helper lemma.

Lemma 2. *Let $m_1, m_2, \ell_1, \ell_2, n \in \mathbb{N}$ and let $(\boldsymbol{A}_I \| \boldsymbol{A}_{II}, \boldsymbol{B}_I \| \boldsymbol{B}_{II}, \boldsymbol{C}_I \| \boldsymbol{C}_{II})$ be any split RICS instance (as described in Section 4.1) for these parameters. For any phase I extended witness $\boldsymbol{z}_I$ and any k-sized set of phase II witnesses $\{\boldsymbol{z}_{II,i}\}_{i\in[k]}$, let $\{\pi_{1,i}, \pi_{2,i}, \pi_{3,i}\}_{i\in[k]}$ be the honestly computed Groth16 proofs for $\{\boldsymbol{z}_I \| \boldsymbol{z}_{II,i}\}_{i\in[k]}$. If $k \geq m_2^2 + 3m_2 + 4$, and vectors $\{[z_{II,i}, (z_{II,i}^\top \otimes z_{II,i})]\}_{i\in[k]}$ are linearly independent, then there exists a polynomial time algorithm $\mathcal{M}$ such that with high probability*

$$\mathcal{M}\left(\{z_{II,i}, \pi_{1,i}, \pi_{2,i}, \pi_{3,i}\}_{i\in[k]}\right) \to \sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1,$$

where $\boldsymbol{T}(X)$ is the vector of m_2^2 polynomials computed as $\left\lfloor \frac{\boldsymbol{a}_{II}^\top(X) \otimes \boldsymbol{b}_{II}(X)}{V(X)} \right\rfloor$.

Proof. Recall that the third group element π_3 in Groth16 is of the form

$$\begin{aligned}\pi_3 =& \overbrace{\sum_{i\in\mathcal{W}_{\mathrm{I}}} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1}^{\mathsf{var}_1} + \sum_{i\in\mathcal{W}_{\mathrm{II}}} z_i \overbrace{\left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\right]_1}^{\mathsf{var}_{2,i}} \\ &+ s\left(\overbrace{\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i [a_i(\tau)]_1}^{\mathsf{var}_3} + \sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i \overbrace{[a_i(\tau)]_1}^{\mathsf{var}_{4,i}}\right) + r\left(\overbrace{\sum_{i\in\mathcal{Z}_{\mathrm{I}}} z_i [b_i(\tau)]_1}^{\mathsf{var}_5} + \sum_{i\in\mathcal{Z}_{\mathrm{II}}} z_i \overbrace{[b_i(\tau)]_1}^{\mathsf{var}_{6,i}}\right) \\ &+ \sum_{i=0}^{n-2} \tilde{q}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 + s\overbrace{[\alpha]_1}^{\mathsf{var}_7} + r\overbrace{[\beta]_1}^{\mathsf{var}_8} + rs\overbrace{[\delta]_1}^{\mathsf{var}_9}.\end{aligned}$$

Here the group elements colored in blue remain constant across all $\pi_{3,j}$. Furthermore, $\sum_{i=0}^{n-2} \tilde{q}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1 = \sum_{u\in[4]} \sum_{i=0}^{n-2} \tilde{q}_{u,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$, where

1. $\tilde{q}_{1,i}$'s are the coefficients in the quotient polynomial $\left\lfloor \frac{\left(\sum_{i\in\mathcal{Z}_\mathrm{I}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_\mathrm{I}} z_i b_i(X)\right)}{V(X)} \right\rfloor$. Therefore, $\mathsf{var}_{10} = \sum_{i=0}^{n-2} \tilde{q}_{1,i} \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$ also remains constant across all $\pi_{3,j}$.

2. $\tilde{q}_{2,i}$'s are the coefficients in the quotient polynomial $\left\lfloor \frac{\left(\sum_{i\in\mathcal{Z}_\mathrm{I}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_\mathrm{II}} z_i b_i(X)\right)}{V(X)} \right\rfloor$ = $\left\langle \boldsymbol{z}_\mathrm{II}, \overbrace{\left\lfloor \frac{\langle \boldsymbol{z}_\mathrm{I}, \boldsymbol{a}_\mathrm{I}(X)\rangle \cdot \boldsymbol{b}_\mathrm{II}(X)}{V(X)} \right\rfloor}^{\boldsymbol{\mu}_2(X)} \right\rangle$. The following m_2 group elements also remain constant across all $\pi_{3,j}$'s: $\boldsymbol{\mathsf{var}_{11}} = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{2,i}[\frac{V(\tau)\tau^i}{\delta}]_1$.

3. $\tilde{q}_{3,i}$'s are the coefficients in the quotient polynomial $\left\lfloor \frac{\left(\sum_{i\in\mathcal{Z}_\mathrm{II}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_\mathrm{I}} z_i b_i(X)\right)}{V(X)} \right\rfloor$ = $\left\langle \boldsymbol{z}_\mathrm{II}, \overbrace{\left\lfloor \frac{\boldsymbol{a}_\mathrm{II}(X) \cdot \langle \boldsymbol{z}_\mathrm{I}, \boldsymbol{b}_\mathrm{I}(X)\rangle}{V(X)} \right\rfloor}^{\boldsymbol{\mu}_3(X)} \right\rangle$. The following m_2 group elements also remain constant across all $\pi_{3,j}$'s: $\boldsymbol{\mathsf{var}_{12}} = \sum_{i=0}^{n-2} \tilde{\boldsymbol{\mu}}_{3,i}[\frac{V(\tau)\tau^i}{\delta}]_1$.

4. $\tilde{q}_{4,i}$'s are the coefficients in the quotient polynomial $\left\lfloor \frac{\left(\sum_{i\in\mathcal{Z}_\mathrm{II}} z_i a_i(X)\right)\cdot\left(\sum_{i\in\mathcal{Z}_\mathrm{II}} z_i b_i(X)\right)}{V(X)} \right\rfloor$ = $\left\langle \overbrace{\left\lfloor \frac{\boldsymbol{a}_\mathrm{II}^\top(X) \otimes \boldsymbol{b}_\mathrm{II}(X)}{V(X)} \right\rfloor}^{\boldsymbol{T}(X)}, \boldsymbol{z}_\mathrm{II}^\top \otimes \boldsymbol{z}_\mathrm{II} \right\rangle$. The following m_2^2 group elements also remain constant across all $\pi_{3,j}$'s: $\boldsymbol{\mathsf{var}_{13}} = \sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i[\frac{V(\tau)\tau^i}{\delta}]_1$

   In other words, for each $j \in [k]$, we can re-write $\pi_{3,j}$ as

$$\begin{aligned}\pi_{3,j} =& \mathsf{var}_1 + \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_2}\rangle + s_j\left(\mathsf{var}_3 + \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_4}\rangle + \mathsf{var}_7\right)\\ &+ r_j\left(\mathsf{var}_5 + \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_6}\rangle + \mathsf{var}_8\right) + r_j s_j \mathsf{var}_9 + \mathsf{var}_{10} + \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_{11}}\rangle\\ &+ \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_{12}}\rangle + \langle \boldsymbol{z}_{\mathrm{II},j}^\top \otimes \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_{13}}\rangle\end{aligned}$$

After rearranging we get,

$$\begin{aligned}\pi_{3,j} =& \mathsf{var}_1 + \mathsf{var}_1 0\\ &+ s_j(\mathsf{var}_3 + \mathsf{var}_7)\\ &+ r_j(\mathsf{var}_5 + \mathsf{var}_8)\\ &+ r_j s_j \mathsf{var}_9\\ &+ \langle \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_2} + \boldsymbol{\mathsf{var}_{11}} + \boldsymbol{\mathsf{var}_{12}}\rangle\\ &+ \langle s_j \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_4}\rangle\\ &+ \langle r_j \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_6}\rangle\\ &+ \langle \boldsymbol{z}_{\mathrm{II},j}^\top \otimes \boldsymbol{z}_{\mathrm{II},j}, \boldsymbol{\mathsf{var}_{13}}\rangle\end{aligned}$$

As a result, $\{\boldsymbol{z}_{\text{II},j}, \pi_{3,j}\}_{j\in[k]}$ can be used to obtain a system of k linear equations in $m_2^2+3m_2+4$ unknown group elements. If $k \geq m_2^2 + 3m_2 + 4$, then this system of equations can be solved in polynomial time, to learn all the unknown group elements. This includes $\textbf{var}_{\textbf{13}}$ which is the m_2^2-length vector of group elements $\sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i[\frac{V(\tau)\tau^i}{\delta}]_1$. This completes the proof of this lemma. □

We now present a formal proof for our main impossibility result.

Theorem 2 (Main Lower-Bound). *Let $m_1, m_2, \ell_1, \ell_2, n \in \mathbb{N}$ and let $(\boldsymbol{A}_I \| \boldsymbol{A}_{II}, \boldsymbol{B}_I \| \boldsymbol{B}_{II}, \boldsymbol{C}_I \| \boldsymbol{C}_{II})$ be any split RICS instance (as described in Section 4.1) for these parameters. There does not exist a split prover (see Definition 2) for Groth16 [Gro16] in the generic group model, where the phase I prover outputs a group element that has the same form as π_3 in Groth16 and where $\widetilde{\mathsf{srs}}_{II}, \mathsf{aux}$ contain $o\left(\mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{II}) \times \mathsf{rank}(\boldsymbol{B}_{II}))\right\}\right)$ group elements.*

Proof. Let $\boldsymbol{K}$ be an $(n-1) \times m_2^2$ sized matrix defined by the evaluation representation of the following m_2^2 quotient polynomials of degree $(n-1)$ each

$$\boldsymbol{T}(X) = \left\lfloor \frac{\boldsymbol{a}_{\text{II}}^\top(X) \otimes \boldsymbol{b}_{\text{II}}(X)}{V(X)} \right\rfloor,$$

i.e., the columns in $\boldsymbol{K}$ correspond to the evaluations of the polynomials in $\boldsymbol{T}(X)$ on the n^{th} roots of unity. It is easy to see that the maximum column rank of this matrix is $\mathsf{rank}(\boldsymbol{K}) = \mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{\text{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\text{II}}))\right\}$, where $\boldsymbol{A}_{\text{II}}$ and $\boldsymbol{B}_{\text{II}}$ are $n \times m_2$ sized matrices defined by the vector of polynomials $\boldsymbol{a}_{\text{II}}(X)$ and $\boldsymbol{b}_{\text{II}}(X)$ respectively.

Let us now assume for the sake of contradiction that there exists a split prover for Groth16, where $\widetilde{\mathsf{srs}}_{\text{II}}$, aux contain $o\left(\mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{\text{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\text{II}}))\right\}\right)$ group elements.

Claim 1. *An adversarial $\mathcal{P}_{II}$ in this split prover variant for Groth16 can recover the following m_2^2 group elements*

$$\sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1.$$

Proof. The adversary samples $k = m_2^2 + 3m_2 + 4$ random phase II extended-witness $\{\boldsymbol{z}_{\text{II},j}\}_{j\in[k]}$, such that the vectors $\{[\boldsymbol{z}_{\text{II},i}, (\boldsymbol{z}_{\text{II},i}^\top \otimes \boldsymbol{z}_{\text{II},i})]\}_{i\in[k]}$ are linearly independent. It then uses the given $\widetilde{\mathsf{srs}}_{\text{II}}, \mathsf{aux}$ on these phase II extended witness to generate a Groth16 proof for each of them, i.e., it computes $\{\pi_{1,j}, \pi_{2,j}, \pi_{3,j}\}_{j\in[k]}$. Observe that each of these Groth16 proofs rely on the same *phase I extended-witness* (this follows from Definition 2). Given these Groth16 proofs and the corresponding set of phase I extended-witnesses, the adversary can then use Lemma 2 to recover the desired m_2^2 group elements. □

We know that out of the m_2^2 group elements $\sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$, $\mathsf{rank}(\boldsymbol{K})$ of them are linearly independent. However, since the generic group model only allows linear operations of the group elements, $|\widetilde{\mathsf{srs}}_{\text{II}}| + |\mathsf{aux}| \in o(\mathsf{rank}(\boldsymbol{K}))$ group elements should not have sufficed to compute all of the m_2^2 group elements $\sum_{i=0}^{n-2} \tilde{\boldsymbol{T}}_i \left[\frac{V(\tau)\tau^i}{\delta}\right]_1$. Hence, our assumption was incorrect and no such split prover for Groth16 exists, where $\widetilde{\mathsf{srs}}_{\text{II}}$, aux contain $o\left(\mathsf{Min}\left\{n-1, (\mathsf{rank}(\boldsymbol{A}_{\text{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\text{II}}))\right\}\right)$ group elements. This completes the proof of this theorem. □

As discussed in remark 3, the phase II proof generation time in our protocols from Section 4 can be optimized to have the second prover perform only $O\left(\mathsf{Min}\left\{n-1, \left(\mathsf{rank}(\boldsymbol{A}_{\mathrm{II}}) \times \mathsf{rank}(\boldsymbol{B}_{\mathrm{II}})\right)\right\}\right)$ group operations. Therefore, our lower bound from Theorem 2 helps demonstrate that the number of group operations performed by the second prover in our protocols is asymptotically tight.

Acknowledgements

We thank Tohru Kohrita for spotting an inconsistency in the proof of Theorem 1 in a previous version of this work. This work is supported in part by the AFOSR Award FA9550-24-1-0156 and research grants from the Bakar Fund, J. P. Morgan Faculty Research Award, Supra Inc., Sui Foundation, and the Stellar Development Foundation. Dimitris Kolonelos is also supported in part by a Berkeley Center for Responsible, Decentralized Intelligence (RDI) Fellowship. Part of this work was done while the second author was a postdoc at NTT Research.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. 17

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017. 3

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013. 6

[BCG+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 336–365. Springer, Heidelberg, December 2017. 6

[BCG+18] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Heidelberg, December 2018. 6

[BCG20a] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 19–46. Springer, Heidelberg, November 2020. 6

[BCG+20b] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press, May 2020. 6

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the rom do not exist. Cryptology ePrint Archive, Paper 2024/728, 2024. https://eprint.iacr.org/2024/728. 6

[BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu. Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 275–304. Springer, Heidelberg, May / June 2022. 6

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014. 6

[BSCG$^+$14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. 3, 4, 5

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. 6

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zksnark provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 6453–6469. USENIX Association, 2023. 6

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020. 6

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. 6

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 6

[GAZ$^+$22] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. Zero-knowledge middleboxes. In *USENIX Security Symposium*, pages 4255–4272. USENIX Association, 2022. 5

[GGJ$^+$23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zksaas: Zero-knowledge snarks as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4427–4444. USENIX Association, 2023. 6

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. 9

[GGW23] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? *IACR Cryptol. ePrint Arch.*, page 1609, 2023. 6

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. 4, 8, 9, 21, 23

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`. 6

[HBHW22] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, 2022. 4

[KMP20] Abhiram Kothapalli, Elisaweta Masserova, and Bryan Parno. A direct construction for asymptotically optimal zksnarks. *IACR Cryptol. ePrint Arch.*, page 1318, 2020. 6

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Heidelberg, August 2022. 6

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Heidelberg, November 2021. 6

[LZW$^+$24] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-snark: Fully distributed proof generation and malicious security. *IACR Cryptol. ePrint Arch.*, page 143, 2024. 6

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. 10

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 3

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. 9

[RPX$^+$22] Deevashwer Rathee, Guru Vamsi Policharla, Tiancheng Xie, Ryan Cottone, and Dawn Song. Zebra: Snark-based anonymous credentials for practical, private and accountable on-chain access control. Cryptology ePrint Archive, Paper 2022/1286, 2022. `https://eprint.iacr.org/2022/1286`. 5

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020. 6

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 6, 10

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008. 6

[WZC$^+$18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018. 6

[XZZ$^+$19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Heidelberg, August 2019. 6

[Zca] Zcash. The halo2 book. `https://zcash.github.io/halo2/index.html`. 4

[ZLW$^+$21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 159–177. ACM Press, November 2021. 6