Non-Interactive Zero-Knowledge Proofs for Composite Statements

Shashank Agrawal[1], Chaya Ganesh[*2], and Payman Mohassel[3]

[1]Visa Research, shaagraw@visa.com
[2]Aarhus University, ganesh@cs.au.dk
[3]Visa Research, pmohasse@visa.com

Abstract

The two most common ways to design non-interactive zero-knowledge (NIZK) proofs are based on Sigma protocols and QAP-based SNARKs. The former is highly efficient for proving algebraic statements while the latter is superior for arithmetic representations.

Motivated by applications such as privacy-preserving credentials and privacy-preserving audits in cryptocurrencies, we study the design of NIZKs for composite statements that compose algebraic and arithmetic statements in arbitrary ways. Specifically, we provide a framework for proving statements that consist of ANDs, ORs and function compositions of a mix of algebraic and arithmetic components. This allows us to explore the full spectrum of trade-offs between proof size, prover cost, and CRS size/generation cost. This leads to proofs for statements of the form: knowledge of x such that $SHA(g^x) = y$ for some public y where the prover's work is 500 times fewer exponentiations compared to a QAP-based SNARK at the cost of increasing the proof size to 2404 group and field elements. In application to anonymous credentials, our techniques result in 8 times fewer exponentiations for the prover at the cost of increasing the proof size to 298 elements.

[*]Work done as an intern at Visa Research.

1 Introduction

Zero-knowledge proofs provide the ability to convince a verifier that a statement is true without revealing the secrets involved. Since their conception in the mid 1980s, zero-knowledge proofs have emerged as a fundamental object in modern cryptography, with connections to the theory of computation [GMW86, For87, BOGG+90, Vad99]. Zero-knowledge proofs (ZKPs) have found numerous applications as a building block in other cryptographic constructions such as identification schemes [FFS87], group signature schemes [CS97], public-key encryption [NY90], anonymous credentials [CL01], voting [CF85], and secure multi-party computation [GMW87]. Most recently, ZKPs have been used as a core component in digital cryptocurrencies such as ZCash and Monero to make the transactions private and anonymous [BCG+14, NMT].

Zero-knowledge proofs exist for all languages in NP [GMW86], but not all such constructions are efficiently implementable. Indeed, a large body of work has been devoted to the design and implementation of efficient ZKPs for a variety of statements. In case of Non-Interactive Zero-Knowledge (NIZK) proofs, which is the focus of this paper, the most practical approaches are based on (i) Sigma protocols (with the Fiat-Shamir transform), (ii) zk-SNARKs and (iii) "MPC-in-the-head" techniques, each with their own efficiency properties, advantages and shortcomings. While the MPC-in-the-head technique [IKOS07] has led to (Boolean) circuit-friendly NIZKs [GMO16, CDG+17, AHIV17], this line of work produces large proofs. In this paper we focus on Sigma protocols and zk-SNARKs, and elaborate on these next.

Sigma Protocols. Many of the statements we prove in cryptographic constructions are efficiently representable as algebraic functions over some group $\mathcal{G}$, such as an elliptic-curve group where the discrete-logarithm problem is hard. For example, Alice may want to convince Bob that she knows an x such that $g^x = y$ for publicly known values $g, y \in \mathcal{G}$ (knowledge of discrete log), or she may like to show that x lies between two public integers a and b (range proof).

Sigma protocol-based ZKPs are extremely efficient for such statements. They yield short proof sizes, require a constant number of public-key operations, and do not impose trusted common reference string (CRS) generation [GQ88, Sch91, CDS94, PS96, GMY06, GK15]. Moreover, they can be made non-interactive, i.e. only a single message from prover to verifier, using the efficient Fiat-Shamir transformation [FS87].

While Sigma protocols are efficient for algebraic statements, they are significantly slower when it comes to non-algebraic ones. Consider a cryptographic hash function or a block cipher represented by a Boolean or arithmetic circuit C, and suppose Alice wants to show that she knows an input x such that $C(x) = y$ for some public y. Alice can treat each gate of C as an algebraic function and provide a proof that the input and output wires of each gate satisfy the associated algebraic relation, to show that she indeed knows x, but this would be prohibitively expensive. In particular, both the proving/verification time and the proof size would grow linearly with the size of circuit which in case of hash functions and block-ciphers can be *tens of thousands* of exponentiations and group elements.

zk-SNARKS. There has been a series of works on constructing zero-knowledge *Succinct* Non-interactive ARguments of Knowledge (zk-SNARKs) [Gro10, Lip12, BCI+13, GGPR13, PHGR13, BCG+13, Lip13, BCTV14]. Starting with the construction of Kilian [Kil92] based on probabilistically checkable proofs (PCPs), made non-interactive by Micali [Mic00], there has been further works [GLR11, BCCT12, DFH12] that construct succinct arguments by removing interaction in Kilian's PCP-based protocol. Despite these advances, PCPs remain concretely expensive and current implementations along this line are not yet efficient. A more effective approach for proving statements about functions represented as Boolean or arithmetic circuits is based on Quadratic Arithmetic Programs (QAPs) [GGPR13] and throughout the paper, we will be concerned with QAP-based zk-SNARK proofs. Such proofs are very short and have fast verification time. More precisely, the proofs have constant size and can be verified in time that is linear in the length of the input x, rather than the length of the circuit C. Thus, zk-SNARKs are better suited for proving statements about hash functions or block ciphers than (non-interactive) Sigma protocols.

In principle, zk-SNARKs could also be used to prove algebraic statements, such as knowledge of discrete-log in a cyclic group by representing the exponentiation circuit as a QAP. The circuit for computing a single exponentiation is in the order of *thousands or millions* of gates depending on the group size. In zk-SNARKs based on QAP, the prover cost is linear in the size of circuit and an honestly generated common reference string (CRS) is needed, whose size also grows proportional to the circuit size. This makes them extremely inefficient for algebraic statements. In contrast, Sigma protocols can be used to prove knowledge of discrete-log with a constant number of exponentiations.

Another disadvantage of zk-SNARKs is that the CRS is generated with respect to a particular circuit C and, in the most efficient instantiations, needs to be regenerated when proving a new statement represented with a different circuit C'. This is not desirable since in current applications such as ZCash, where CRS is generated using an expensive secure multi-party computation (MPC) protocol in order to guarantee soundness of the proof system [parb]. In contrast, Sigma protocols have constant-size untrusted CRSs that can be used to prove arbitrary statements and can be generated inexpensively (without an MPC).

1.1 Composite Statements and Applications

Composite statements that include multiple algebraic and arithmetic components appear in various applications. We discuss three important cases here.

Proof of Solvency. Consider privacy-preserving proofs of solvency for Bitcoin exchanges [Wil, DBB+15]. Here an exchange wants to prove to its customers that it has enough reserves to cover its liabilities, or, in simple words, that it is solvent. A proof of reserves in the Bitcoin network amounts to showing that the exchange has control over certain Bitcoin addresses. A Bitcoin address is a 160-bit hash of the public portion of a public/private ECDSA keypair [bit], where the public portion is derived from the private key by doing an exponentiation operation on the secp256k1 curve [sec][1]. Thus the exchange wants to show that it knows the private keys corresponding to some hashed public keys available on the blockchain. Furthermore, the proof should not reveal the public keys themselves otherwise an adversary would be able to track the movement of exchange's funds.

In particular, the exchange wants to show that it knows a secret x such that $H(g^x) = y$ where H is a hash function such as SHA-256. The statement has both algebraic (g^x) and Boolean (hash function H) parts. One can express the composite function (exponentiate then hash) as a purely algebraic or Boolean function and then use a Sigma protocol or zk-SNARK respectively, but, in the former case, the proof size and verification time will be quite large, while in the latter, the proof generation time will increase substantially and a much larger CRS is needed. Ideally, one would like to use a Sigma protocol for the algebraic part and a zk-SNARK for the Boolean part, and then combine the two proofs so that no extra information about x is revealed (beyond the fact that $H(g^x) = y$).

Thus any proof of solvency for a Bitcoin exchange must deal with a zero-knowledge proof that combines both Boolean and algebraic statements. Existing proposals for proofs of solvency

[1] Most cryptocurrencies generate public/private keys and define an address in a similar manner. Apart from Bitcoin and its fork Bitcoin Cash, Ethereum is another prominent example (see Appendix F in the yellow paper [Woo]).

get around this problem by assuming (incorrectly) that public keys themselves are available on the blockchain so that Sigma protocols alone suffice [DBB+15]. As we will see later, our efficient techniques allow designing NIZKs for proving knowledge of x given $H(g^x)$ that require roughly 500 times fewer exponentiations for the prover compared to proving the same statement using a QAP-based SNARK.

Privacy-Preserving Credentials. Digital certificates (X.509) are commonly used to identify entities over the Internet. They include a message m that may contain various identifying information about a user or a machine, and a digital signature (by a certificate authority) on the message attesting to its authenticity. The signature can then be verified by anyone who holds the public verification key. Typically, certificates reveal the message m and hence the identity of their owner. Anonymous credentials [Cha82] provide the same authentication guarantees without revealing the identifying message, and are widely studied due to their strong privacy guarantees. A main ingredient for making digital certificates anonymous is a ZKP of knowledge of a message m and a signature σ, where σ is a valid signature on message m with respect to the verification key vk. The ZKP ensures that we do not leak any information about m beyond the knowledge of a valid signature. A large body of work has studied anonymous credentials, but only a handful of techniques can turn commonly used X.509 certificates into anonymous credentials. The main challenge is that the ZKP statement being proven is a hybrid statement containing both algebraic (RSA or elliptic-curve operations) and Boolean functions (hashing), since the message is hashed before being algebraically signed. The work of Delignat-Lavaud et al. [DLFKP16] constructs a proof for such a hybrid statement using only zk-SNARKs which, as discussed earlier, is inefficient for the algebraic component, while the work of Chase et al. [CGM16] design such ZKP proofs in the interactive setting where the prover and verifier exchange multiple messages. Efficient NIZK for composite statements based on both zk-SNARKs and Sigma protocols would yield more efficient anonymous credential systems. Using our techniques for RSA signature results in prover's work that is about 8 times fewer group exponentiations compared to Cinderella [DLFKP16].

zk-SNARKs with composable CRSs. Anonymous decentralized digital crypto-currencies such as ZCash use zk-SNARKs to prove a massive statement containing many different smaller components. For example, at a high level, one of the statement being proven in ZCash is of the form: I have knowledge of x_i's such that $H(x_1||H(x_2||\ldots H(x_n))) = y$ for a large value of n. The CRS generated for proving this statement is extremely large (about a gigabyte for ZCash [para]) and cannot be reused to prove any other statement. A better alternative is to generate a much smaller CRS for proving a statement of the form: I have knowledge of x, y such that $H(x||H(y))$, combined with a technique for composing many such proofs. More generally, one can envision a general system with CRSs for small size statements $C_1, \ldots, C_n$ that enables NIZKs for arbitrary composition of these statements without having to generate new CRSs for each new composition. This yields a trade-off between proof size and the CRS size (and its reusability).

1.2 Contributions

Motivated by the above applications, we study the design of NIZKs for composite statements that compose algebraic and arithmetic statements in arbitrary ways. Specifically, we provide new protocols for statements that consist of ANDs, ORs and function compositions of a mix of algebraic and arithmetic components. In doing so, our goal is to maintain the invariant that algebraic components are proven using Sigma protocols, and arithmetic statements using QAP-based zk-SNARKs. This allows us to explore the full spectrum of trade-offs between proof size (verification cost), prover cost, and CRS size (and cost of generation) for composite statements.

More precisely, we propose new NIZKs for proof of knowledge of x, x_1, x_2, y_1, y_2 such that

- $f_1(x_1, f_2(x_2)) = z$,
- $f_1(x, y_1) = z_1$ *AND* $f_2(x, y_2) = z_2$,
- $f_1(x, y_1) = z_1$ *OR* $f_2(x, y_2) = z_2$,

for public values z, z_1, z_2, and where f_1 and f_2 can be either algebraic or arithmetic. Given our NIZKs for these compositions, it is easy to handle arbitrary composite statements. This is the first work that directly addresses the question of non-interactive proofs for composite statements and how disparate techniques can be used to prove them in zero-knowledge efficiently. We highlight two important technical ingredients that enable our NIZKs for composite statements below. We note that in this paper we primarily focus on elliptic curves as our algebraic group, as they are the most efficient for instantiating both zk-SNARKs and Sigma protocols.

Sigma protocols for statements on algebraically committed inputs and outputs: We show techniques for proving that the input/output used in a Sigma protocol for an algebraic statement are the same as input/output committed to by an algebraic commitment scheme, say Com. This enables using the output of an algebraic statement as an intermediate output in a composite statement. For instance, we can prove knowledge of h, x_1, x_2 such that $h = g_1^{x_1} g_2^{x_2^2}$ given g_1, g_2, $\mathsf{Com}(h)$, $\mathsf{Com}(x_1)$, $\mathsf{Com}(x_2)$. To enable such proofs, we commit to a point P on an elliptic curve $E(\mathbb{F}_t)$ by committing to its coordinates, i.e. $\mathsf{Com}(P) = (\mathsf{Com}_q(P_x), \mathsf{Com}_q(P_y))$ where $P = (P_x, P_y)$ and $q > t$.

- **Proof of addition of committed elliptic curve points.** We show efficient techniques for proving knowledge of two committed elliptic curve points P, Q such that $T = P + Q$ for a public point T. To do so, we expand the elliptic curve addition/subtraction operation $P + Q - T$ such that $T = P + Q$ holds if and only if two sets of equations of the form $L(\cdot) = R(\cdot)$ hold, where L and R are multivariate polynomials of degree 3 in the coordinates. Given commitments to the coordinate values and the output of polynomials L and R, we prove the corresponding relations between the committed values using Sigma protocols. For this to work, we address an additional technical subtlety that the addition operation over elliptic curve points is defined over $\mathbb{F}_t$, while the commitment scheme maybe over a different group of size q. While this may be addressed by using two different commitment schemes in groups of different orders, it would require performing the Complex Multiplication method to choose an elliptic curve group of a specific order which is a quite inefficient. The proof can be extended to the case where T is also private and committed to.

- **Double-discrete log proofs for elliptic-curve groups.** We show efficient techniques for committing to a group element g^x where g is a generator for an elliptic curve group, and proving knowledge of x such that $\mathsf{Com}(g^x) = y$ given a public y. Previous techniques for proving such statements are limited to RSA groups [CS97, MGGR13] and hence are not usable in many applications including privacy-preserving audits for Bitcoin which uses elliptic curve groups. We show how to securely reduce this problem to that of proving addition of committed EC points.

- **Proof of equality of committed values over different groups.** We show techniques for proving knowledge of x such that $\mathsf{Com}_p(x) = y$ and $\mathsf{Com}_q(x) = z$ for public values y, z where Com_p denotes an algebraic commitment over an elliptic curve group of size p (similarly, Com_q).

This allows us to easily move from proof systems in one group to another by committing to the shared values in both groups and invoking this proof. Existing techniques involve exponentiations in an RSA group and are fairly expensive as the group order is hidden.

zk-SNARKs for statements on algebraically committed inputs and outputs: We show efficient techniques for proving that the input/output used in a zk-SNARK for an arithmetic statement are the same as the input/output committed to by an algebraic commitment scheme. This enables efficient switching between the algebraic and arithmetic world, and helps hide intermediate outputs of an arithmetic statement (by committing to it), when used in a composition. For example, this enables proving knowledge of input x such that $\mathsf{Com}(x) = y$ and $\mathsf{Com}(H(x)) = z$ for public values y, z where H is SHA2 and Com is a Pedersen commitment over an elliptic-curve group, or prove knowledge of x such that $\mathsf{Com}(x) = y, \mathsf{Com}(H(x)) = z$, and $H(H(x)) = w$.

To design these new proofs, we dissect existing zk-SNARK constructions, and separately process private input and output wires of the statement circuit during CRS generation, proof generation and verification. We then ensure that the values for those input/output wires are consistent with corresponding algebraic commitments to the same values using customized Sigma protocols.

2 Preliminaries

Notation. Throughout the paper, we use κ to denote the security parameter or level. A function is negligible if for all large enough values of the input, it is smaller than the inverse of any polynomial. We use negl to denote a negligible function. Let $\{\mathcal{X}_\kappa\}_{\kappa\in\mathbb{N}}$ and $\{\mathcal{Y}_\kappa\}_{\kappa\in\mathbb{N}}$ be ensembles where $\mathcal{X}_\kappa$ and $\mathcal{Y}_\kappa$ are probability distributions over $\{0,1\}^{\mathsf{poly}(\kappa)}$ for some polynomial poly. We say $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable if for all PPT distinguishers $\mathcal{D}$, there exists a negligible function negl such that $|\Pr[\mathcal{D}(\mathcal{X}_\kappa) = 1] - \Pr[\mathcal{D}(\mathcal{Y}_\kappa) = 1]| \leq \mathsf{negl}(\kappa)$. We write $\mathcal{X}_\kappa \equiv \mathcal{Y}_\kappa$ to mean that the distributions $\mathcal{X}_\kappa$ and $\mathcal{Y}_\kappa$ are identical. We use $[1, n]$ to represent the set of numbers $\{1, 2, \ldots, n\}$. If Alg is a randomized algorithm, we use $y \leftarrow \mathsf{Alg}(x)$ to denote that y is the output of Alg on x. We write $x \stackrel{R}{\leftarrow} \mathcal{X}$ to mean sampling a value x uniformly from the set $\mathcal{X}$.

We denote an interactive protocol between two parties A and B by $\langle \mathsf{A}, \mathsf{B} \rangle$. $\langle \mathsf{A}(x), \mathsf{B}(y) \rangle (z)$ denotes a protocol where A has input x, B has input y and z is a common input. Also, view_A denotes the "view" of A in an interaction with B, which consists of the input to A, its random coins, and the messages sent by B (view_B is defined in a similar manner).

Bilinear groups. Let $\mathsf{GroupGen}$ be an asymmetric pairing group generator that on input 1^κ, outputs description of three cyclic groups $\mathbb{G}$, $\widetilde{\mathbb{G}}$, $\mathbb{G}_T$ of prime order $p = \Theta(2^\kappa)$ equipped with a non-degenerate efficiently computable bilinear map $e : \mathbb{G} \times \widetilde{\mathbb{G}} \to \mathbb{G}_T$, and generators g and $\tilde{g}$ for $\mathbb{G}$ and $\widetilde{\mathbb{G}}$ respectively. The discrete logarithm assumption is said to hold in $\mathbb{G}$ relative to $\mathsf{GroupGen}$ if for all PPT algorithms $\mathcal{A}$, $\Pr[x \leftarrow \mathcal{A}(\mathbb{G}, p, g, h) \mid (\mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T) \leftarrow \mathsf{GroupGen}; x \stackrel{R}{\leftarrow} \mathbb{Z}_p; h := g^x]$ is $\mathsf{negl}(\kappa)$.

In this paper, we primarily consider elliptic curves as our algebraic group. Let E be an elliptic curve defined over a field $\mathbb{F}_t$. The set of points on the curve form a group under the point addition operation, and we denote the group by $E(\mathbb{F}_t)$. For an element $P \in E(\mathbb{F}_t)$ of prime order p, P_x and P_y represent the x and y co-ordinates of the point P respectively. In some constructions, we use additive notation and write $Q = \alpha P$ for a scalar $\alpha \in \mathbb{F}_p$. The discrete logarithm assumption is believed to hold in well chosen elliptic curve groups where group elements are represented with $O(\kappa)$ bits. In our constructions, we use asymmetric bilinear groups where $\mathbb{G} \neq \widetilde{\mathbb{G}}$, and discrete logarithm is hard in $\mathbb{G}$. We also rely on q-type assumptions on bilinear maps, and describe them in Appendix G.

Zero-knowledge Proofs. Let R be an efficiently computable binary relation which consists of pairs of the form (s, w) where s is a statement and w is a witness. Let $\mathcal{L}$ be the language associated with R, i.e., $\mathcal{L} = \{s \mid \exists w \text{ s.t. } R(s, w) = 1\}$.

A zero-knowledge proof for $\mathcal{L}$ lets a prover P convince a verifier V that $s \in \mathcal{L}$ for a common input s without revealing w. A proof of knowledge captures not only the truth of a statement $s \in \mathcal{L}$, but also that the prover "possesses" a witness w to this fact. We are concerned with non-interactive proofs in this paper where P sends only one message to V, and V decides whether to accept or not based on its input, the message, and any public parameters. We define them formally below.

2.1 Non-interactive Zero-knowledge Proofs

Non-interactive zero-knowledge (NIZK) proofs are usually studied in the common reference string (CRS) model, wherein a string of a special structure is generated in a setup phase, and made available to everyone to prove/verify statements.

Definition 2.1 (Non-interactive Zero-knowledge Argument [BFM88, FLS90])**.** *A NIZK argument for an NP relation R consists of a triple of polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *defined as follows.*

- $\mathsf{Setup}(1^\kappa)$ *takes a security parameter κ and outputs a CRS Σ.*
- $\mathsf{Prove}(\Sigma, s, w)$ *takes as input the CRS Σ, a statement s, and a witness w, and outputs an argument π.*
- $\mathsf{Verify}(\Sigma, s, \pi)$ *takes as input the CRS Σ, a statement s, and a proof π, and outputs either* 1 *accepting the argument or* 0 *rejecting it.*

The algorithms above should satisfy the following properties.

1. *Completeness. For all $\kappa \in \mathbb{N}$, $(s, w) \in R$,*

$$\Pr\left(\mathsf{Verify}(\Sigma, s, \pi) = 1 \;:\; \begin{array}{c}\Sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ \pi \leftarrow \mathsf{Prove}(\Sigma, s, w)\end{array}\right) = 1.$$

2. *Computational soundness. For all PPT adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\begin{array}{c}\mathsf{Verify}(\Sigma, \tilde{s}, \tilde{\pi}) = 1 \\ \wedge\, \tilde{s} \notin L\end{array} \;:\; \begin{array}{c}\Sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (\tilde{s}, \tilde{\pi}) \leftarrow \mathcal{A}(1^\kappa, \Sigma)\end{array}\right).$$

3. *Zero-knowledge. There exists a PPT simulator $(\mathcal{S}_1, \mathcal{S}_2)$ such that $\mathcal{S}_1$ outputs a simulated CRS Σ and trapdoor τ; $\mathcal{S}_2$ takes as input Σ, a statement s and τ, and outputs a simulated proof π; and, for all PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible in κ:*

$$\left|\Pr\left(\begin{array}{c}(s, w) \in R\, \wedge \\ \mathcal{A}_2(\pi, \mathsf{st}) = 1\end{array} \;:\; \begin{array}{c}\Sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (s, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\kappa, \Sigma) \\ \pi \leftarrow \mathsf{Prove}(\Sigma, s, w)\end{array}\right) - \Pr\left(\begin{array}{c}(s, w) \in R\, \wedge \\ \mathcal{A}_2(\pi, \mathsf{st}) = 1\end{array} \;:\; \begin{array}{c}(\Sigma, \tau) \leftarrow \mathcal{S}_1(1^\kappa) \\ (s, w, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\kappa, \Sigma) \\ \pi \leftarrow \mathcal{S}_2(\Sigma, \tau, s)\end{array}\right)\right|.$$

Definition 2.2 (Non-interactive Zero-knowledge Argument of Knowledge). *A NIZK argument of knowledge for a relation R is a NIZK argument for R with the following additional extractability property:*

- *Extraction. For any PPT adversary $\mathcal{A}$, random string $r \xleftarrow{R} \{0,1\}^*$, there exists a PPT algorithm* Ext *such that the following probability is negligible in κ:*

$$\Pr\left(\begin{array}{c}\mathsf{Verify}(\Sigma, \tilde{s}, \tilde{\pi}) = 1 \\ \wedge\, R(\tilde{s}, w') = 0\end{array} : \begin{array}{c}\Sigma \leftarrow \mathsf{Setup}(1^\kappa) \\ (\tilde{s}, \tilde{\pi}) \leftarrow \mathcal{A}(1^\kappa, \Sigma; r) \\ w' = \mathsf{Ext}(\Sigma, \tilde{s}, \tilde{\pi}; r)\end{array}\right).$$

Definition 2.3 (zero-knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARK)). *A zk-SNARK for a relation R is a non-interactive zero-knowledge argument of knowledge for R with the following additional property:*

- *Succinctness. For any s and w, the length of the proof π is given by $|\pi| = \mathsf{poly}(\kappa) \cdot \mathsf{polylog}(|s| + |w|)$.*

2.2 Sigma Protocols

Sigma protocols are two-party interactive protocols of a specific structure. Let P (the prover) and V (the verifier) be two parties with common input s and a private input w for P. In a Sigma protocol, P sends a message a, V replies with a random κ-bit string r, P then sends a message e, and V decides to accept or reject based on the transcript (a, r, e). If V accepts (outputs 1), then the transcript is called accepting.

Definition 2.4 (Sigma protocol [Dåm]). *An interactive protocol between a prover P and a verifier V is a Σ protocol for a relation R if the following properties are satisfied:*

1. *It is a three move public coin protocol.*
2. *Completeness: If P and V follow the protocol then $\Pr[\langle P(w), V\rangle(s) = 1] = 1$ whenever $(s, w) \in R$.*
3. *Special soundness: There exists a polynomial time algorithm called the extractor which when given s and two transcripts (a, r, e) and (a, r', e') that are accepting for s, with $r \neq r'$, outputs w' such that $(s, w') \in R$.*
4. *Special honest verifier zero knowledge: There exists a polynomial time simulator which on input s and a random r outputs a transcript (a, r, e) with the same probability distribution as that generated by an honest interaction between P and V on (common) input s.*

Fiat-Shamir transform. A Σ protocol can be efficiently compiled into a non-interactive zero-knowledge proof of knowledge (in the random oracle model) through the Fiat-Shamir transform [FS87]. Not only the transformation removes interaction from the protocol, but also makes it zero-knowledge against malicious verifiers. At a high level, the transform works by having the prover compute the verifier's message by applying an appropriate hash function, modeled as a random oracle in the security proof, to the prover's first message to obtain a random challenge.

OR composition of Σ-protocols. In Cramer et al. [CDS94], the authors devise an OR composition technique for Sigma protocols. Essentially, a prover can efficiently show $((x_0 \in \mathcal{L}) \vee (x_1 \in \mathcal{L}))$ without revealing which x_i is in the language. More generally, the OR transform can handle two different relations R_0 and R_1.

Theorem 2.5 (OR-composition [CDS94])**.** *If Π_0 is a Σ-protocol for R_0 and Π_1 a Σ-protocol for R_1, then there is a Σ-protocol Π_{OR} for the relation R_{OR} given by $\{((x_0, x_1), w) : ((x_0, w) \in R_0) \vee ((x_1, w) \in R_1)\}$.*

Pedersen commitment. Throughout the paper, we use algebraic commitment schemes that allow proving linear relationships among committed values. The Pedersen commitment scheme [Ped92] is one such example which gives unconditional hiding and computational binding properties based on the hardness of computing discrete logarithm in a group $\mathcal{G}$, say of order q. Given two random generators $g, h \in \mathcal{G}$ such that $\log_g h$ is unknown, a value $x \in \mathbb{Z}_q$ is committed to by choosing r randomly from $\mathbb{Z}_q$, and computing $g^x h^r$. We write $\mathsf{Com}_q(x)$ to denote a Pedersen commitment to x in a group of order q.

Sigma protocols are known in literature to prove knowledge of a committed value, equality of two committed values, and so on, and these protocols can be combined in natural ways. In particular, linear relationships between Pedersen commitments can be shown through existing techniques [Sch91, FO97, CS97, CM99]. For example, one could show that $y = ax + b$ for some public values a and b, given $\mathsf{Com}_q(x)$ and $\mathsf{Com}_q(y)$.

We use $\mathsf{PK}\{(x, y, \ldots) :$ *statements* about $x, y, \ldots\}$ to denote a proof of knowledge of $x, y, \ldots$ that satisfies *statements* [CS97]. Other values in *statements* are public.

2.3 SNARK Construction from QAP

The work of Gennaro et al. [GGPR13] showed how to encode computations as quadratic programs. They show how to convert any Boolean circuit into a Quadratic Span Program (QSP) and any arithmetic circuit into a Quadratic Arithmetic Program (QAP). In this work, we will only use the latter definition. Even though QSPs are designed for Boolean circuits, arithmetic *split gates* defined in Parno et al. [PHGR13] translate an arithmetic wire into binary output wires, and Boolean functions may be computed using arithmetic gates. Parno et al. also note that such an arithmetic embedding results in a smaller QAP compared to the QSP of the original Boolean circuit. In the rest of the paper, we assume that Boolean functions are computed by a QAP defined over an arithmetic field, and hence will only be concerned with QAP.

Definition 2.6 (Quadratic Arithmetic Program [GGPR13])**.** *A quadratic arithmetic program (QAP) Q over a field $\mathbb{F}$ consists of three sets of polynomials $V = \{v_k(x) : k \in \{0, \ldots, m\}\}, W = \{w_k(x) : k \in \{0, \ldots, m\}\}, Y = \{y_k(x) : k \in \{0, \ldots, m\}\}$ and a target polynomial $t(x)$, all in $\mathbb{F}[X]$.*

Let $f : \mathbb{F}^n \to \mathbb{F}^{n'}$ be a function with input variables labeled $1, \ldots, n$ and output variables labeled $m-n'+1, \ldots, m$. A QAP Q is said to compute f if the following holds: $a_1, \ldots, a_n, a_{m-n'+1}, \ldots, a_m \in \mathbb{F}^{n+n'}$ is a valid assignment to the input and output variables of f (i.e., $f(a_1, \ldots, a_n) = (a_{m-n'+1}, \ldots, a_m)$) iff there exist $(a_{n+1}, \cdots, a_{m-n'}) \in \mathbb{F}^{m-n-n'}$ such that $t(x)$ divides $p(x)$, where

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} a_k v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} a_k w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} a_k y_k(x)\right).$$

The size of the QAP Q is m, and degree is deg$(t(x))$.

The polynomials $v_k(x), w_k(x), y_k(x)$ have degree at most $\deg(t(x))-1$, since they can be reduced modulo $t(x)$ without affecting the divisibility check. We review the QAP-based SNARK construction of Parno et al. [PHGR13] in Appendix H.

3 NIZK on Committed IO for Algebraic Statements

In this section, we design Sigma protocols for knowledge of inputs and outputs of algebraic statements where the inputs and outputs are committed to. In other words, we enable proof of knowledge of x_i given commitments $\mathsf{Com}(x_i)$ to inputs and a commitment $\mathsf{Com}(\Pi g_i^{P_i(x_i)})$ to the output of an algebraic function where g_is are public generators in an elliptic curve group and P_is are public single-variable polynomials. An important ingredient in this is a proof of knowledge of double discrete log which we elaborate on next.

3.1 Proof of Knowledge of Double Discrete Logarithm

Our goal is to prove the equality of a committed value and the discrete logarithm of another committed value. When the commitments are in elliptic curve groups, the known techniques for double discrete logarithm proofs will not work [CS97, MGGR13]. This is because a group element cannot be naturally interpreted as a field element, as can be done in integer groups. Towards this end, we first describe a protocol to prove that the sum of two elliptic curve points that are committed to, is another public point on the curve.

In this section, we consider the family of curves E given by

$$y^2 = x^3 + ax + b, \tag{1}$$

where $a, b \in \mathbb{F}_t$, but the techniques we describe below would extend to other curve families like Edwards [Edw07]. The curve sec256k1 used by Bitcoin has the form of equation 1 with $a = 0, b = 7$.

The point addition relation is defined by the point addition equation specific to the curve family. Let $P = (x_1, y_1), Q = (x_2, y_2), P, Q \in E(\mathbb{F}_t)$ for the family E above. For distinct P, Q, $P \neq -Q$, $(x_3, y_3) = P + Q$ is given by

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \tag{2}$$

$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1. \tag{3}$$

We use $\mathsf{addFormula}(P, Q)$ to denote (x_3, y_3) computed in this way. When $P = Q$, the operation is doubling of the point P, denoted by $\mathsf{doubleFormula}(P)$. In this case, (x_3, y_3) is given by

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \tag{4}$$

$$y_3 = \frac{3x_1^2 + a}{2y_1}(x_1 - x_3) - y_1. \tag{5}$$

We could prove the above relations for committed x_1, x_2, y_1, y_2 using known Sigma protocol techniques. But since the point addition computation is over $\mathbb{F}_t$, the commitments to the coordinates have to be in a group of order t, which is not necessarily the same as p, the order of the group $E(\mathbb{F}_t)$. The Complex Multiplication (CM) method could be used to find elliptic curve groups of a specific order. However, it is quite inefficient for large orders and would make our protocols impractical . We avoid the CM method by proposing a protocol that does not need to find a group of a given order.

We rewrite the point addition formula (equations 2 and 3) as

$$x_3x_2^2 + x_3x_1^2 + x_1^3 + x_2^3 + 2y_1y_2 = y_2^2 + y_1^2 + x_1^2x_2 + x_1x_2^2 + 2x_1x_2x_3, \tag{6}$$
$$x_2y_3 + x_3y_2 + x_2y_1 = x_1y_2 + x_3y_1 + x_1y_3. \tag{7}$$

Let L_x and R_x denote the left-hand side and right-hand side respectively of equation 6, and L_y and R_y of equation 7. That is:

$$\begin{aligned}
L_x(x_1, y_1, x_2, y_2) &= x_3x_2^2 + x_3x_1^2 + x_1^3 + x_2^3 + 2y_1y_2,\\
R_x(x_1, y_1, x_2, y_2) &= y_2^2 + y_1^2 + x_1^2x_2 + x_1x_2^2 + 2x_1x_2x_3,\\
L_y(x_1, y_1, x_2, y_2) &= x_2y_3 + x_3y_2 + x_2y_1,\\
R_y(x_1, y_1, x_2, y_2) &= x_1y_2 + x_3y_1 + x_1y_3.
\end{aligned}$$

We use Sigma protocols to prove that L_x, R_x, L_y and R_y satisfy the above relations using committed intermediate values. To do so, in addition to linear relationships, our protocol needs to prove that a committed value is the product of two committed values: given $C_1 = \mathsf{Com}(a) = g^a h^{r_1}, C_2 = \mathsf{Com}(b) = g^b h^{r_2}, C_3 = \mathsf{Com}(c) = g^c h^{r_3}$, prove $c = ab$. This can be done by proving knowledge of b such that the discrete logarithm of C_4 with respect to C_1 is equal to the committed value in C_2, and the equality of committed values in C_4 and C_3, where $C_4 = C_1^b$. The prover computes and sends $C_4 = C_1^b$ with the following proof: $\mathsf{PK}\{(a, b, c, b', c', r_1, r_2, r_3, r_4) : C_1 = g^a h^{r_1} \wedge C_2 = g^b h^{r_2} \wedge C_3 = g^c h^{r_3} \wedge C_4 = C_1^{b'} \wedge C_4 = g^{c'} h^{r_4} \wedge b' = b \wedge c' = c\}$. In general, Sigma protocols for polynomial relationships among committed values were given by Camenisch and Michels [CM99]. For completeness, we sketch the ideas in Appendix E.

Let G_2 be an elliptic-curve group of order q such that $q > 2t^3$, and P', Q' be points in G_2. We commit to the coordinates and the intermediate values necessary for the proof in G_2, and since the largest intermediate value in equations 6 and 7 is cubic, the choice of q ensures there is no wrap around when the computation is modulo q. Since all computation on committed values will now be modulo q, and the addition equations are to be computed modulo t, we use division with remainder. We prove equality of L_x and R_x modulo q, divide them by t taking away multiples of t, and prove that the remainders are equal. When used together with appropriate range proofs to prove that the remainder does not exceed the divisor, and that the committed coordinates are in the desired range, we get equality modulo t. (There are several known techniques to build range proofs [Bou00, CCs08], that is, to prove that $x \in [0, S]$ for a public S and committed x, including the recent, very efficient technique called Bulletproof [BBB+17].)

The protocol addition given in Figure 1 proves that the addition formula holds for committed points P, Q and their sum T. We show that addition is secure in Appendix C.2. The protocol's cost is dominated by the range proofs in steps 4, 5, 6 and the proof for polynomial relationships in steps 2 and 3. addition roughly has a proof size of $75 + \log\log t$ elements, and prover's work $60 + \log t$ exponentiations.

Let $\mathsf{C}_P = \mathsf{Com}_q(P) = (\mathsf{Com}_q(P_x), \mathsf{Com}_q(P_y))$ denote a commitment to a point $P = (P_x, P_y)$.

Theorem 3.1. *Let $E(\mathbb{F}_t)$ be an elliptic curve given by equation 1, $T \in E$ and $q > 2t^3$. Then,* addition *in Figure 1 is a Σ-protocol for the relation $R = \{((T, \mathsf{C}_P, \mathsf{C}_Q), (P, Q)) : \mathsf{C}_P = \mathsf{Com}_q(P) \wedge \mathsf{C}_Q = \mathsf{Com}_q(Q) \wedge T = \mathsf{addFormula}(P, Q) \wedge P, Q \in E\}$.*

Using techniques similar to the above protocol addition, we obtain a protocol double to prove that doubling formula holds, i.e. $T = \mathsf{doubleFormula}(P)$. Now, we can handle all cases of point addition

Given $T = (T_x, T_y), \mathsf{C}_1 = \mathsf{Com}_q(P_x), \mathsf{C}_2 = \mathsf{Com}_q(P_y), \mathsf{C}_3 = \mathsf{Com}_q(Q_x), \mathsf{C}_4 = \mathsf{Com}_q(Q_y)$, prove that $T = P + Q$, where $P = (P_x, P_y), Q = (Q_x, Q_y), T \in E(\mathbb{F}_t)$ and $q > 2t^3$.

1. Let $L_x(P_x, P_y, Q_x, Q_y) = k_1 t + r_1, R_x(P_x, P_y, Q_x, Q_y) = k'_1 t + r'_1, L_y(P_x, P_y, Q_x, Q_y) = k_2 t + r_2, R_y(P_x, P_y, Q_x, Q_y) = k'_2 t + r'_2$, for $k_1, k'_1, k_2, k'_2 < \frac{q}{t}$ and $r_1, r'_1, r_2, r'_2 < t$.

   Compute and send commitments $\mathsf{C}_4 = \mathsf{Com}_q(L_x), \mathsf{C}_5 = \mathsf{Com}_q(R_x), \mathsf{C}_6 = \mathsf{Com}_q(L_y), \mathsf{C}_7 = \mathsf{Com}_q(R_y), \mathsf{C}_8 = \mathsf{Com}_q(k_1), \mathsf{C}_9 = \mathsf{Com}_q(r_1), \mathsf{C}_{10} = \mathsf{Com}_q(k'_1), \mathsf{C}_{11} = \mathsf{Com}_q(r'_1), \mathsf{C}_{12} = \mathsf{Com}_q(k_2), \mathsf{C}_{13} = \mathsf{Com}_q(r_2), \mathsf{C}_{14} = \mathsf{Com}_q(k'_2), \mathsf{C}_{15} = \mathsf{Com}_q(r'_2)$.

2. Prove that $(P_x, P_y), (Q_x, Q_y)$ and (T_x, T_y) satisfy the addition equation for the x-coordinate.

   $\pi_1 : \mathsf{PK}\{(P_x, P_y, Q_x, Q_y, L_x, R_x) : \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y) \wedge \mathsf{C}_4 = \mathsf{Com}_q(L_x) \wedge \mathsf{C}_5 = \mathsf{Com}_q(R_x) \wedge L_x = T_x Q_x^2 + T_x P_x^2 + P_x^3 + P_y^3 + 2P_y Q_y \wedge R_x = Q_y^2 + P_y^2 + P_x^2 Q_x + P_x Q_x^2 + 2P_x Q_x T_x\}$

3. Prove that $(P_x, P_y), (Q_x, Q_y)$ and (T_x, T_y) satisfy the addition equation for the y-coordinate.

   $\pi_2 : \mathsf{PK}\{(P_x, P_y, Q_x, Q_y, L_y, R_y) : \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y) \wedge \mathsf{C}_6 = \mathsf{Com}_q(L_y) \wedge \mathsf{C}_7 = \mathsf{Com}_q(R_y) \wedge L_y = Q_x T_y + T_x Q_y + Q_x P_y \wedge R_y = P_x Q_y + T_x P_y + P_x T_y\}$

4. Prove that the coordinates are in the correct range.

   $\pi_3 : \mathsf{PK}\{(P_x, P_y, Q_x, Q_y) : \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y) \wedge Q_x < t \wedge Q_y < t \wedge P_x < t \wedge P_y < t\}$

5. Prove that L_x and R_x are equal modulo t, by dividing each side by t, showing correct range for the quotients and the remainders, and proving the remainders are equal.

   $\pi_4 : \mathsf{PK}\{(L_x, R_x, k_1, k'_1, r_1, r'_1) : \mathsf{C}_4 = \mathsf{Com}_q(L_x) \wedge \mathsf{C}_5 = \mathsf{Com}_q(R_x) \wedge \mathsf{C}_8 = \mathsf{Com}_q(k_1) \wedge \mathsf{C}_9 = \mathsf{Com}_q(r_1) \wedge \mathsf{C}_{10} = \mathsf{Com}_q(k'_1) \wedge \mathsf{C}_{11} = \mathsf{Com}_q(r'_1) \wedge L_x = k_1 t + r_1 \wedge R_x = k'_1 t + r'_1 \wedge r_1 < t \wedge r'_1 < t \wedge k_1 < \frac{q}{t} \wedge k'_1 < \frac{q}{t} \wedge r_1 - r'_1 = 0\}$

6. Prove that L_y and R_y are equal modulo t, by dividing each side by t, showing correct range for the quotients and the remainders, and proving the remainders are equal.

   $\pi_5 : \mathsf{PK}\{(L_y, R_y, k_2, k'_2, r_2, r'_2) : \mathsf{C}_6 = \mathsf{Com}_q(L_y) \wedge \mathsf{C}_7 = \mathsf{Com}_q(R_y) \wedge \mathsf{C}_{12} = \mathsf{Com}_q(k_2) \wedge \mathsf{C}_{13} = \mathsf{Com}_q(r_2) \wedge \mathsf{C}_{14} = \mathsf{Com}_q(k'_2) \wedge \mathsf{C}_{15} = \mathsf{Com}_q(r'_2) \wedge L_y = k_2 t + r_2 \wedge R_y = k'_2 t + r'_2 \wedge r_2 < t \wedge r'_2 < t \wedge k_2 < \frac{q}{t} \wedge k'_2 < \frac{q}{t} \wedge r_2 - r'_2 = 0\}$

Figure 1: $\mathsf{addition} : \mathsf{PK}\{(P = (P_x, P_y), Q = (Q_x, Q_y)) : T = (T_x, T_y) = \mathsf{addFormula}(P, Q) \wedge \mathsf{C}_1 = \mathsf{Com}_q(P_x) \wedge \mathsf{C}_2 = \mathsf{Com}_q(P_y) \wedge \mathsf{C}_3 = \mathsf{Com}_q(Q_x) \wedge \mathsf{C}_4 = \mathsf{Com}_q(Q_y)\}$

through the following statement:

$$(P \neq Q \wedge P \neq -Q \wedge T = \mathsf{addFormula}(P,Q)) \vee \\ (P = Q \wedge T = \mathsf{doubleFormula}(P)) \vee (P = -Q \wedge T = 0)\,.$$

This statement can be proved using OR composition of Sigma protocols: protocol addition for the first part of the OR statement, protocol double for the second, and simple Sigma protocols for the last component. We denote the proof of point addition of two committed points by pointAddition.

$$\mathsf{pointAddition} : \mathsf{PK}\{(P,Q) : \mathsf{C}_P = \mathsf{Com}_q(P) \wedge \mathsf{C}_Q = \mathsf{Com}_q(Q) \wedge P, Q \in E \wedge \\ ((P \neq Q \wedge P \neq -Q \wedge T = \mathsf{addFormula}(P,Q)) \vee \\ (P = Q \wedge T = \mathsf{doubleFormula}(P)) \vee (P = -Q \wedge T = 0))\}$$

For curves with a complete formula like Edwards, a point addition proof will not have different cases based on the relationship between P and Q.

Theorem 3.2. *Let $E(\mathbb{F}_t)$ be an elliptic curve given by equation 1, $T \in E$ and $q > 2t^3$. Then,* pointAddition *is a Σ-protocol for the relation $R = \{((T, \mathsf{C}_P, \mathsf{C}_Q), (P,Q)) : \mathsf{C}_P = \mathsf{Com}_q(P) \wedge \mathsf{C}_Q = \mathsf{Com}_q(Q) \wedge T = P + Q \wedge P, Q \in E\}$.*

We note that the protocol addition may be modified to prove point addition for a committed point T in the following way. The proofs π_1 and π_2 are on committed coordinates (T_x, T_y), and the range proof π_3 also includes proving the range of coordinates of T. We denote the point addition proof $\mathsf{PK}\{(P,Q,T) : C_P = \mathsf{Com}_q(P) \wedge C_Q = \mathsf{Com}_q(Q) \wedge C_T = \mathsf{Com}_q(T) \wedge T = P + Q \wedge P, Q, T \in E\}$ on all committed inputs by comPointAddition.

We now construct a protocol to prove the equality of a committed value and the discrete logarithm of another committed value using the point addition proof. The double discrete logarithm proof is given in Figure 2. (See Appendix C.3 for a proof of security.) While the prover's work is dominated by the protocol pointAddition, we note that the range proofs for each challenge bit may be batched [BBB+17]. For soundness 2^{-60}, the protocol ddlog incurs proof size of about $2370 + \log\log t$ elements and prover's work of $1800 + 30\log t$ exponentiations.

Theorem 3.3. *Let $E(\mathbb{F}_t)$ be an elliptic curve given by equation 1, and $P \in E$ be an element of prime order p. Then,* ddlog *is a Σ-protocol for the relation $R = \{(P, \mathsf{C}, \mathsf{C}_h, (\lambda, h)) : \mathsf{C} = \mathsf{Com}(\lambda) \wedge \mathsf{C}_h = \mathsf{Com}(h) \wedge h = \lambda P, 0 < \lambda < p\}$ with soundness $1/2$.*

3.2 Sigma Protocols on Committed Outputs

In this section, we construct Sigma protocols for committed output. First, we note a simpler construction when the output is a single bit. (This simpler variant is used in our OR compositions.) In particular, given an algebraic commitment to private input x, public y and an efficient Sigma protocol to prove that $f(x,y) = 1$, we show how to construct an efficient Sigma protocol to prove $f(x,y) = b$, for a committed bit b. Let $f : \mathbb{Z}_q^{n+m} \to \{0,1\}$, and let C be a commitment to the input x. Let f_{com} be the relation, $f_{\mathsf{com}} = \{(y,(x,b)) : ((x,y) \in \mathcal{L}_f \wedge b = 1) \vee (b = 0)\}$. The Sigma protocol for the relation f_{com} is given by the proof $\mathsf{PK}\{(b,x) : f(x,y) = b \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\}$. Let $\mathcal{G}$ be a group of order q, g a generator of $\mathcal{G}$, and h a random element of $\mathcal{G}$ such that the discrete logarithm of h with respect to g is unknown to the prover. Let Π be a Σ-protocol for the relation f. The Σ-protocol for f_{com} is shown in Figure 3.

Theorem 3.4. *If Π is a Σ-protocol for f, then* comBitSigma *is a Σ-protocol for f_{com}.*

To generalize the above to the case where output is a group element and not a single bit, we need one more building block.

Given $\mathsf{C}_1 = \mathsf{Com}_p(\lambda), \mathsf{C}_2 = \mathsf{Com}_q(x), \mathsf{C}_3 = \mathsf{Com}_q(y)$, for $q > 2t^3$, prove that $(x, y) = \lambda P$, where $P \in E$ is an element of prime order p, $0 < \lambda < p$, P', Q', points in G_2 of order q.

1. The prover computes the following values: $a_1 = \mathsf{Com}_p(\alpha) = \alpha P + \beta_1 Q, a_2 = \mathsf{Com}_q(\gamma_1) = \gamma_1 P' + \beta_2 Q', a_3 = \mathsf{Com}_q(\gamma_2) = \gamma_2 P' + \beta_3 Q'$ where $\alpha \in \mathbb{F}_p$ is chosen at random, and $(\gamma_1, \gamma_2) = \alpha P$. It sends a_1, a_2, a_3 to the verifier.
2. The verifier chooses a random challenge bit c and sends it to the prover.
3. For challenge c,
   - If $c = 0$, compute $z_1 = \alpha, z_2 = \beta_1, z_3 = \beta_2, z_4 = \beta_3$. Send the tuple (z_1, z_2, z_3, z_4)
   - If $c = 1$, compute $z_1 = \alpha - \lambda$. Let $T = z_1 P = (t_1, t_2)$. The prover uses $\mathsf{pointAddition}$ (Figure 1) to prove that $T = (\gamma_1, \gamma_2) - (x, y)$. Let π be $\mathsf{PK}\{(x, y, \gamma_1, \gamma_2) : T = (\gamma_1, \gamma_2) - (x, y)\}$. Send (z_1, π).
4. Verification: Compute $(t_1, t_2) = z_1 P$. If $c = 0$, check if $a_1 = z_1 P + z_2 Q, a_2 = t_1 P' + z_3 Q', a_3 = t_2 P' + z_4 Q'$. If $c = 1$, verify proof π.

Figure 2: ddlog : $\mathsf{PK}\{(\lambda, x, y, r, r_1, r_2) : \mathsf{Com}_p(\lambda) = \lambda P + rQ \wedge \mathsf{Com}_q(x) = xP' + r_1 Q' \wedge \mathsf{Com}_q(y) = yP' + r_2 Q' \wedge (x, y) = \lambda P\}$

Given $y, C = \mathsf{Com}(x), D_b = \mathsf{Com}(b)$, prove that $f(x, y) = b$.

- The prover uses the protocol Π for f, Σ-protocol for proving knowledge of committed values, and the OR-transform to prove the following statement:

$$\begin{aligned}\mathsf{PK}\big\{(b, x) : \big(f(x, y) = 1 \wedge b = 1 \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\big) \\ \vee \big(b = 0 \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\big)\big\}\end{aligned}$$

Figure 3: $\mathsf{comBitSigma}$: $\mathsf{PK}\{(b, x) : f(x, y) = b \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\}$

Proof of Point Addition and Discrete Log on Committed Points. Suppose we want to prove that a committed point is the sum of two group elements. But the challenge is that the input group elements are secret and are committed to, hence the prover also needs to prove knowledge of discrete logarithms of the input points with respect to a public base. Specifically, our goal is to design a protocol to prove knowledge of discrete logarithms of two committed points such that their sum is another committed point which we do using $\mathsf{comPointAddition}$. Let E be an elliptic curve defined over $\mathbb{F}_t$, and let $P \in E$ be an element of prime order p. Let $q > 2t^3$ be a prime. The protocol comSum : $\mathsf{PK}\{(\gamma, \alpha, \beta, x_1, x_2) : \gamma = \alpha + \beta \wedge \alpha = x_1 P \wedge \beta = x_2 P\}$ for $0 < x_1, x_2 < p$ is shown in Figure 4.

When Committed Output is a Group Element. In the following discussion, similar to before, for a group element $\alpha = (\alpha_x, \alpha_y)$, where α_x, α_y are the two coordinates of the elliptic curve point, the commitment to the point is performed by committing to its two coordinates in the proper group, i.e. $\mathsf{Com}(\alpha) = (\mathsf{Com}(\alpha_x), \mathsf{Com}(\alpha_y))$.

- The prover computes commitments $c_1 = \mathsf{Com}_p(x_1), c_2 = \mathsf{Com}_p(x_2), c_3 = \mathsf{Com}_q(\alpha), c_4 = \mathsf{Com}_q(\beta), c_5 = \mathsf{Com}_q(\gamma)$
- The prover uses ddlog to give the following proof.
  $\mathsf{PK}\{(x_1, \alpha) : \alpha = x_1 P \wedge c_3 = \mathsf{Com}_q(\alpha) \wedge c_1 = \mathsf{Com}_p(x_1)\}$
- The prover uses ddlog to give the following proof.
  $\mathsf{PK}\{(x_2, \beta) : \beta = x_2 P \wedge c_4 = \mathsf{Com}_q(\beta) \wedge c_2 = \mathsf{Com}_p(x_2)\}$
- The prover uses comPointAddition to give the following proof, given the commitments $c_3 = (\mathsf{Com}_q(\alpha_x), \mathsf{Com}_q(\alpha_y)), c_4 = (\mathsf{Com}_q(\beta_x), \mathsf{Com}_q(\beta_y)), c_5 = (\mathsf{Com}_q(\gamma_x), \mathsf{Com}_q(\gamma_y))$ and the point addition formula for the elliptic curve that defines the group (Equations 6,7).
  $\mathsf{PK}\{(\gamma, \alpha, \beta) : \gamma = \alpha + \beta \wedge c_3 = \mathsf{Com}_q(\alpha) \wedge c_4 = \mathsf{Com}_q(\beta) \wedge c_5 = \mathsf{Com}_q(\gamma)\}$

Figure 4: comSum : $\mathsf{PK}\{(\gamma, \alpha, \beta, x_1, x_2) : \gamma = \alpha + \beta \wedge \alpha = x_1 P \wedge \beta = x_2 P\}$

We observe that given the above-mentioned building blocks i.e. ddlog and comSum, we can construct Sigma protocol on a committed output group element for algebraic statements of the form $f(x_1, \ldots, x_n) = \Pi g_i^{P_i(x_i)}$. We sketch the ideas at a high-level for some simple functions. Let $f : \mathbb{Z}_p^n \to \mathcal{G}$, where $\mathcal{G}$ is a group $E(\mathbb{F}_t)$ of order p. When $f(x) = g^x$, then this reduces to the ddlog proof. For $f(x_1, x_2) = g_1^{x_1} g_2^{x_2}$, it suffices to commit to $g_1^{x_1}$ and $g_2^{x_2}$ separately and call the comSum proof. To consider higher degree polynomials in the exponent let us consider $f(x) = g^{x^2}$. To construct a proof $\mathsf{PK}\{(x, y) : g^{x^2} = y \wedge \mathsf{C}_1 = \mathsf{Com}(x) \wedge \mathsf{C}_2 = \mathsf{Com}(y)\}$, the prover computes the commitments $C_1 = \mathsf{Com}_p(x)$, $C_2 = \mathsf{Com}_p(x^2)$ and $C_3 = \mathsf{Com}_q(k) = (\mathsf{Com}_q(k_x), \mathsf{Com}_q(k_y))$, where $k = g^{x^2} = (k_x, k_y)$, for the choice of q as discussed in Section 3.1. Now, the prover gives the following proofs. $\mathsf{PK}\{(x_2, k) : k = g^{x_2} \wedge C_2 = \mathsf{Com}_p(x_2) \wedge C_3 = \mathsf{Com}_q(k)\}$ using ddlog, and a Sigma protocol for $\mathsf{PK}\{(x_1, x_2) : x_2 = x_1^2 \wedge C_1 = \mathsf{Com}_p(x_1) \wedge C_2 = \mathsf{Com}_p(x_2)\}$. Given the above building blocks, it is easy to see that we can extend the techniques to devise proofs comSigma for $f(x_1, \ldots x_n) = \Pi g_i^{P_i(x_i)}$.

4 NIZK on Committed IO for Non-Algebraic Statements

In this section we instantiate the following two building blocks which are critical for our NIZKs for composite statements.

- *zk-SNARK on committed input*. Given an algebraic commitment $C = g^x h^r$, and a circuit f, a zk-SNARK proof that $f(x, z) = b$.
- *zk-SNARK on committed input and output*. Given algebraic commitments $C_1 = g^x h^r, C_2 = g^b h^r$, and a circuit f, a zk-SNARK proof that $f(x, z) = b$.

We first give a brief high-level description of our central ideas. Our starting point is a SNARK where the proof consists of multi-exponentiation that resembles a Pedersen commitment. We identify what part of the proof allows commitments to a private input (witness) and private output (for hiding intermediate values of a larger computation) by suitably separating the input/output wires so there are corresponding distinct proof elements in the SNARK. We then commit to the private input and output of the SNARK proof independently using Pedersen commitment, and show equality of the committed values and the values in the multi-exponentiation proof element. While this observation has been used in prior works in verifiable computation [CFH+15, FFG+16], it has been in different contexts and for different purposes. We briefly discuss how our ideas relate to two such ideas.

In [CFH+15], the authors present a verifiable computation scheme called Geppetto where the prover can share state across proofs. They generalize QAPs to create MultiQAPs which allow one to commit to data, and use it in many proofs. But crucially, all the proofs are for statements still represented as circuits while we also utilize the commitment to switch to sigma protocol proofs.

In [FFG+16], certain proof elements of a SNARK act as "accumulated" value of inputs in the context of large data size. The multi-exponentiations computed by the verifier in [FFG+16] act as a hash on data and different computations may be performed (verifiably) on it. The verifier computes the hash, and the proof verification involves checking the proof is consistent with the hash along with checks that the computation was performed correctly on the data using only the hash that was computed. On the other hand, in our setting, the multi-exponentiation is part of the proof, and computed by the prover, whose consistency across proofs must be shown. Additionally, these proofs could be different sigma protocols proving a variety of algebraic relations among some subset of the input used in the SNARK. Though our idea of exploiting a proof element with a certain structure is similar to the above works, we use it towards a different end.

For concreteness, we describe our protocol using the verifiable computation protocol Pinocchio [PHGR13] (see Appendix H) as a starting point. But our techniques carry over to other SNARK constructions as well. The key property we need from a SNARK construction is that the proof contains a multi-exponentiation of the input/output. Given this, we separate the circuit wires and obtain in a non-blackbox way, commitments as part of the SNARK proof.

Before giving the description of the above building blocks, we introduce an important ingredient: a protocol for proving equality of the discrete logarithms $(a_1, \ldots, a_n)$ in $y = \prod_{i=1}^{n} G_i^{a_i}$ and individual algebraic commitments to them.

4.1 Proof of Equality of Aggregated Discrete Logs & Commitments

Let $\mathbb{G}$ be a group of prime order q. Given $y = \prod G_i^{a_i}$ and $C_i = g^{a_i} h^{r_i}$, where g, G_i are generators of the group $\mathbb{G}$, h is a random element of the group, and the prover does not know the discrete logarithm of h with respect to g, and the discrete logarithms of G_is with respect to each other. We want to prove equality of the discrete logarithms in y and the respective values committed to in C_is. Let k be the statistical security parameter. We give a Sigma protocol, and following standard notation, we denote the protocol by $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$.

We show that the protocol in Figure 5 is correct, has a soundness error of $1/2^k$, and is honest verifier zero knowledge in Appendix C.1.

4.2 zk-SNARK on Committed Inputs

Recall that at a high level, each polynomial of the quadratic program (Definition 2.6), say, $v_k(x) \in \mathbb{F}[x]$ is mapped to an element in a bilinear group, $g^{v_k(s)}$, where s is a secret value chosen during CRS generation. Given these group elements and the values a_i on the circuit wires which are the coefficients of the quadratic program, the prover can compute "in the exponent" to obtain $g^{v(s)}$, where $v(s) = \sum a_i v_k(s)$. The verifier uses the bilinear map to verify that the divisibility check of the QAP holds. We assume the computations are over large fields, that is, the QAP is defined over $\mathbb{F}_p$ for a large p. The size of the field is exponential in the security parameter. We omit p in all further descriptions of the field.

Let $f : \mathbb{F}^N \to \mathbb{F}^{n'}$ be a function with input/output values from $\mathbb{F}$, computed by an arithmetic circuit C with input wires labeled $1, \ldots, N$, output wires labeled $m - n' + 1, \ldots, m$. Let $\mathcal{Q}$ be a QAP

Given $y = \prod_{i=1}^{n} G_i^{a_i}$ and $C_i = g^{a_i} h^{r_i}$

1. The prover computes the following values: $u = \prod_{i=1}^{n} G_i^{\alpha_i}$ and $v_i = g^{\alpha_i} h^{R_i}$ for randomly chosen $\alpha_i, R_i \in \mathbb{Z}_q$ and sends u, v_i to the verifier.
2. The verifier chooses a challenge c at random from $\mathbb{Z}_{2^k}$ for a fixed k, such that $2^k < q$, and sends it to the prover.
3. For a challenge string c, prover computes and sends the tuple (s_i, t_i)

$$s_i = \alpha_i - ca_i \pmod{q}, \qquad t_i = R_i - cr_i \pmod{q}$$

4. Verification: Check if $u = y^c \prod G_i^{s_i}$ and $v_i = (C_i)^c g^{s_i} h^{t_i}$. The verifier accepts if checks succeed for all i.

Figure 5: comEq : $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$

of size m and degree d corresponding to C. We separate the circuit wires I into private input, public input, intermediate values, and output wires. Let $I_{com} \subseteq \{1, \ldots, N\}$ be the set of indices corresponding to the private inputs $a_1, \ldots, a_n$, I_{pub} the indices for the public input wires, and I_{out} the indices for the public output. Then let $I_{mid} = \{1, \ldots, m\} \setminus (I_{pub} \cup I_{com} \cup I_{out})$ be the indices of the intermediate wires. This way there are separate CRS elements corresponding to the private input and public input allowing the prover to compute corresponding proof elements. The divisibility check can still proceed, and we include additional span checks for the new proof elements. Now, we bind the multi-exponentiation corresponding to the private input in the proof to the value committed to in a Pedersen commitment using the protocol comEq. Let $C_i = g^{a_i} h^{r_i}$ be a Pedersen commitment to the ith input a_i. The construction $\mathsf{comInSnark}$: $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : f(a_1, \ldots a_n, z_1, \ldots, z_{N-n}) = (b_1, \ldots, b_{n'}) \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$ is given in Figure 6.

Given commitments to private inputs $C_i = g^{a_i} h^{r_i}$ for $i \in [n]$, public inputs $z_1, \ldots, z_{N-n}$, and public outputs $b_1, \ldots, b_{n'}$.

1. CRS generation: Run $\mathsf{GroupGen}(1^\kappa)$ to get $(p, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, g, \tilde{g}, e)$. Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, g_w = g^{r_w}, \tilde{g}_w = \tilde{g}^{r_w}, g_y = g^{r_y}$.
   Set the CRS to be:

$$\begin{aligned}\mathsf{crs} = \Big(&\{g_v^{v_k(s)}\}_{k \in I_{com}}, \{g_v^{v_k(s)}\}_{k \in I_{mid}}, \{\tilde{g}_w^{w_k(s)}\}_{k \in I_{com}},\\ &\{\tilde{g}_w^{w_k(s)}\}_{k \in I_{mid}}, \{g_y^{y_k(s)}\}_{k \in I_{com}}, \{g_y^{y_k(s)}\}_{k \in I_{mid}}, \{g_v^{\alpha_v v_k(s)}\}_{k \in I_{com}},\\ &\{g_v^{\alpha_v v_k(s)}\}_{k \in I_{mid}}, \{\tilde{g}_w^{\alpha_w w_k(s)}\}_{k \in I_{com}}, \{\tilde{g}_w^{\alpha_w w_k(s)}\}_{k \in I_{mid}},\\ &\{g_y^{\alpha_y y_k(s)}\}_{k \in I_{com}}, \{g_y^{\alpha_y y_k(s)}\}_{k \in I_{mid}}, \{g^{s^i}\}_{i \in [d]}, \{\tilde{g}^{s^i}\}_{i \in [d]},\\ &\{g^{\alpha_v s^i}\}_{i \in [d]}, \{\tilde{g}^{\alpha_v s^i}\}_{i \in [d]}, \{g^{\alpha_w s^i}\}_{i \in [d]}, \{\tilde{g}^{\alpha_w s^i}\}_{i \in [d]}, \{g^{\alpha_y s^i}\}_{i \in [d]},\\ &\{\tilde{g}^{\alpha_y s^i}\}_{i \in [d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k \in I_{com}}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k \in I_{mid}}\Big)\end{aligned}$$

   Set the short verification CRS to be:

$$\begin{aligned}\mathsf{shortcrs} = \Big(&g, \tilde{g}, \tilde{g}^{\alpha_v}, g^{\alpha_w}, \tilde{g}^{\alpha_y}, \tilde{g}^{\gamma}, g^{\beta\gamma}, \tilde{g}^{\beta\gamma}, g^{t(s)},\\ &\{g_v^{v_k(s)}\}_{k \in I_{com}}, \{g_v^{v_k(s)}\}_{k \in I_{pub} \cup I_{out}}, \{\tilde{g}_w^{w_k(s)}\}_{k \in I_{pub} \cup I_{out}}, \{g_y^{y_k(s)}\}_{k \in I_{pub} \cup I_{out}}\Big)\end{aligned}$$

2. Prove: On input $z_1, \ldots, z_{N-n}$, witness $a_1, \ldots, a_n$, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i\in[m]}$. (Equivalently, evaluates the circuit to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{com}(x) = \sum_{k\in I_{com}} a_k v_k(x)$, $v_{mid}(x) = \sum_{k\in I_{mid}} a_k v_k(x)$ and similarly define $w_{com}(x)$, $w_{mid}(x)$, $y_{com}(x)$ and $y_{mid}(x)$.

   - The prover computes the proof π:

$$\Big(g_v^{v_{com}(s)}, g_v^{v_{mid}(s)}, \tilde{g}_w^{w_{com}(s)}, \tilde{g}_w^{w_{mid}(s)}, g_y^{y_{com}(s)}, g_y^{y_{mid}(s)}, \tilde{g}^{h(s)},$$
$$\tilde{g}_v^{\alpha_v v_{com}(s)}, \tilde{g}_v^{\alpha_v v_{mid}(s)}, g_w^{\alpha_w w_{com}(s)}, g_w^{\alpha_w w_{mid}(s)}, \tilde{g}_y^{\alpha_y y_{com}(s)}, \tilde{g}_y^{\alpha_y y_{mid}(s)}$$
$$g_v^{\beta v_{com}(s)} g_w^{\beta w_{com}(s)} g_y^{\beta y_{com}(s)}, g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)}\Big)$$

   - Prove input consistency with commitment: The prover uses the Sigma protocol comEq to compute π_{in}: $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1}h^{r_1} \wedge \cdots \wedge C_n = g^{a_n}h^{r_n}\}$, for $G_i = g_v^{v_i(s)}, i \in I_{com}$, and $y = g_v^{v_{com}(s)}$.

3. Verify:

   - On input $\mathsf{shortcrs}$, z, and proofs π, π_{in} parse π as

$$\pi = \big(g^{V_{com}}, g^{V_{mid}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, g^{Y_{com}}, g^{Y_{mid}}, \tilde{g}^{H},$$
$$\tilde{g}^{V'_{com}}, \tilde{g}^{V'_{mid}}, g^{W'_{com}}, g^{W'_{mid}}, \tilde{g}^{Y'_{com}}, \tilde{g}^{Y'_{mid}}, g^{Z_{com}}, g^{Z_{mid}}\Big)$$

   - Divisibility check. Compute $g_v^{v_{io}(s)} = \prod_{k\in I_{pub}\cup I_{out}} (g_v^{v_k(s)})^{a_k}$. Similarly, compute $\tilde{g}_w^{w_{io}(s)}$ and $g_y^{y_{io}(s)}$. Verify that

$$e\left(g_v^{v_0(s)} g_v^{v_{io}(s)} g^{V_{com}} g^{V_{mid}}, \tilde{g}_w^{w_0(s)} \tilde{g}_w^{w_{io}(s)} \tilde{g}^{W_{com}} \tilde{g}^{W_{mid}}\right)$$
$$= e\left(g^{t(s)}, \tilde{g}^{H}\right) \cdot e\left(g_y^{y_0(s)} g_y^{y_{io}(s)} g^{Y_{com}} g^{Y_{mid}}, \tilde{g}\right).$$

   - Verify that the linear combinations are in correct spans.
     (a) $e\left(g^{V_{com}}, \tilde{g}^{\alpha_v}\right) = e\left(g, \tilde{g}^{V'_{com}}\right)$
     (b) $e\left(g^{V_{mid}}, \tilde{g}^{\alpha_v}\right) = e\left(g, \tilde{g}^{V'_{mid}}\right)$
     (c) $e\left(g^{W'_{com}}, \tilde{g}\right) = e\left(g^{\alpha_w}, \tilde{g}^{W_{com}}\right)$
     (d) $e\left(g^{W'_{mid}}, \tilde{g}\right) = e\left(g^{\alpha_w}, \tilde{g}^{W_{mid}}\right)$
     (e) $e\left(g^{Y_{com}}, \tilde{g}^{\alpha_y}\right) = e\left(g, \tilde{g}^{Y'_{com}}\right)$
     (f) $e\left(g^{Y_{mid}}, \tilde{g}^{\alpha_y}\right) = e\left(g, \tilde{g}^{Y'_{mid}}\right)$
   - Verify same coefficients in all linear combinations.
     (a) $e\left(g^{Z_{com}}, \tilde{g}^{\gamma}\right) = e\left(g^{V_{com}} g^{Y_{com}}, \tilde{g}^{\beta\gamma}\right) \cdot e\left(g^{\beta\gamma}, \tilde{g}^{W_{com}}\right)$
     (b) $e\left(g^{Z_{mid}}, \tilde{g}^{\gamma}\right) = e\left(g^{V_{mid}} g^{Y_{mid}}, \tilde{g}^{\beta\gamma}\right) \cdot e\left(g^{\beta\gamma}, \tilde{g}^{W_{mid}}\right)$
   - Verify input consistency with commitment: Set $G_i = g_v^{v_i(s)}$, $i \in I_{com}$, and $y = g^{V_{com}}$. Verify the proof π_{in}.

Figure 6: comInSnark : $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : f(a_1, \ldots a_n, z_1, \ldots, z_{N-n}) = (b_1, \ldots, b_{n'}) \wedge C_1 = g^{a_1} h^{r_1} \wedge \ldots \wedge C_n = g^{a_n} h^{r_n}\}$

Zero-knowledge. We make our construction zero-knowledge, and obtain zkcomInSnark, by randomizing the elements in the proof π such that the checks verify and the proof is statistically indistinguishable from random group elements. Specifically, the prover chooses random $\delta_v, \delta_w, \delta_y \leftarrow \mathbb{F}$, and adds $\delta_v t(s)$ in the exponent to $v_{com}(s)$, $v_{mid}(s)$; $\delta_w t(s)$ to $w_{com}(s)$, $w_{mid}(s)$; and $\delta_y t(s)$ to $y_{com}(s)$, $y_{mid}(s)$. It is easy to see that the modified value of $p(x)$ remains divisible by $t(x)$. The following terms are added to crs: $g_v^{t(s)}$, $\tilde{g}_w^{t(s)}$, $g_y^{t(s)}$, $g_v^{\alpha_v t(s)}$, $g_w^{\alpha_w t(s)}$, $g_y^{\alpha_y t(s)}$, $g_v^{\beta t(s)}$, $g_w^{\beta t(s)}$, $g_y^{\beta t(s)}$ ($g_v^{t(s)}$ is also added to shortcrs). Prover can now compute the new values in π from crs, and they are verified in the same manner as before. The proof π_{in} now proves a slightly different statement: $\mathsf{PK}\{(a_1, \ldots, a_n, \delta, r_1, \ldots, r_n) : y = H^\delta \prod_{i=1}^n G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \ldots \wedge C_n = g^{a_n} h^{r_n}\}$. To verify it, the verifier uses $g_v^{t(s)}$ from shortcrs.

Theorem 4.1. *If q-PDH, $2q$-SDH and d-PKE assumptions hold for* GroupGen *for $q \geq 4d + 4$, then zk-*comInSnark *instantiated with a QAP of degree d is secure under Definition 2.2.*

We prove the above theorem in Appendix B. Similarly, by separating the circuit wires into private input, public input, intermediate values and private output, we obtain zk-SNARK on committed input and output. This construction, comIOSnark, is presented in Appendix A. We state the theorem below.

Theorem 4.2. *If q-PDH, $2q$-SDH and d-PKE assumptions hold for* GroupGen *for $q \geq 4d + 4$, and discrete logarithm assumption holds in $\mathbb{G}$, then zk-*comIOSnark *instantiated with a QAP of degree d is secure under Definition 2.2.*

5 Constructions for Compound Statements

In this section we use the building blocks we constructed in Sections 4 and 3, to devise proofs for compound statements. In the following, we distinguish between functions that have an efficient algebraic representation versus functions that are efficiently represented as an arithmetic circuit over a field. Of course, any algebraic function can be written as a circuit over some field. But certain functions, modular exponentiation for instance, have a large circuit size and hence it is more desirable to not use a circuit in computing them. Therefore, when we say *algebraic* or *arithmetic* for functions below, we really mean the efficient representation of the function for computation. We say a function f is arithmetic if an arithmetic circuit is used to compute f, and say f is algebraic if it is represented algebraically. In this section, we show how to prove compound statements involving function compositions, OR, and AND. In our compositions, the SNARK used for the circuit could use a group whose order does not match with the group of the sigma protocol for the algebraic part. We construct a building block Eq to prove equality of committed values in different groups, given in Appendix D, which we use in our compositions.

5.1 Function Composition

We assume that the commitments we use in the following are in groups of correct order for the computation, so as to focus on the ideas for the composition. Wlog., our compositions hold even when the the scalar field of the elliptic curve group, the field the curve is defined over and the field of the arithmetic circuit are all different, since we can prove equality of committed values in different groups using the protocol Eq given in Figure 12. We present the interactive variant for ease of presentation but note that all our constructions can be made non-interactive by running all the proofs in parallel and invoking the standard Fiat-Shamir transform (see Section 2.1). The constructions below also easily generalize to functions that have more input/output elements than shown, i.e. we can obtain constructions for statements of the form $\mathsf{PK}\{(x_1,\ldots,x_n,y_1,\ldots,y_m) : f_1(x_1,\ldots,x_n,f_2(y_1,\ldots,y_m)) = z\}$ where f_1, f_2 may each be arithmetic or algebraic. We give constructions composition by elaborating on the four possible compositions next:

1. f_1 and f_2 are functions represented as arithmetic circuits. Let $f_1 : \mathbb{F}_p^2 \to \mathbb{F}_p$, and $f_2 : \mathbb{F}_p \to \mathbb{F}_p$, and we want to prove knowledge of secrets x_1, x_2 such that $f_1(x_1, f_2(x_2)) = z$ for a public z. An example is proof of knowledge of x_1 and x_2 such that $H(x_1||H(x_2)) = z$ where H is a collision resistant hash function such as SHA256. Such a composition can help reduce the size of CRS by composing the same or a few SNARK systems multiple times to obtain more complex statements without an increase in CRS size.

- The prover commits to x_1, x_2 and $x_3 = f_2(x_2)$ by computing $c_1 = \mathsf{Com}_p(x_1), c_2 = \mathsf{Com}_p(x_2), c_3 = \mathsf{Com}_p(x_3)$. The prover sends c_1, c_2, c_3 to the verifier.
- The prover uses zk-comIOSnark to give a proof that $f_2(x_2) = x_3$, given c_2 and c_3. $\mathsf{PK}\{(x_2, x_3, r_2, r_3) : f_2(x_2) = x_3 \wedge c_2 = \mathsf{Com}_p(x_2) \wedge c_3 = \mathsf{Com}_p(x_3)\}$.
- The prover uses zk-comInSnark to give a proof that $f_1(x_1, x_3) = z$ given c_1, c_3 and z. $\mathsf{PK}\{(x_1, x_3, r_1, r_3) : f_1(x_1, x_3) = z \wedge c_1 = \mathsf{Com}_p(x_1) \wedge c_3 = \mathsf{Com}_p(x_3)\}$.

2. f_1 is an arithmetic circuit and f_2 is algebraic. Let $f_1 : \mathbb{F}_p^3 \to \mathbb{F}_p$, $f_2 : \mathbb{Z}_q \to \mathcal{G}$ and $T : \mathcal{G} \to \mathbb{F}_p^2$. In this proof, we assume the algebraic function is over an elliptic curve group and assume the natural transformation for mapping an elliptic curve point to a tuple of field elements, i.e. its coordinates. Let $\mathcal{G}$ be an elliptic curve group of prime order q, and let $T(k) = (k_x, k_y)$ for $k \in \mathcal{G}$, where (k_x, k_y) are the coordinates of the elliptic curve point. The following is a protocol for $\mathsf{PK}\{(x_1, x_2) : f_1(x_1, T(f_2(x_2))) = z\}$. An example is proving knowledge of x such that $H(g^x) = z$.

- The prover commits to x_1, x_2 and $k = f_2(x_2)$ by computing $c_1 = \mathsf{Com}_p(x_1), c_2 = \mathsf{Com}_q(x_2), c_3 = \mathsf{Com}_p(k) = (\mathsf{Com}_p(k_x), \mathsf{Com}_p(k_y))$, and sends c_1, c_2, c_3 to the verifier.
- The prover uses the protocols ddlog and the sigma protocol on committed group element comSigma to give the following proof: $\mathsf{PK}\{(x_2, k, r_2, r_3) : f_2(x_2) = k \wedge c_2 = \mathsf{Com}_q(x_2) \wedge c_3 = \mathsf{Com}_p(k)\}$.
- The prover uses zk-comInSnark to prove $f_1(x_1, T(k)) = z$ given c_1, c_3, c_4. $\mathsf{PK}\{(x_1, k, r_1, r_3) : f_1(x_1, T(k)) = z \wedge c_1 = \mathsf{Com}_p(x_1) \wedge c_3 = \mathsf{Com}_p(k)\}$.

3. f_1 is algebraic, and f_2 is an arithmetic circuit. Let $f_1 : \mathbb{Z}_q^2 \to \mathcal{G}, f_2 : \mathbb{F}_p \to \mathbb{F}_p$. Let Π be a Σ-protocol for f_1. The following is a protocol for $\mathsf{PK}\{(x_1, x_2) : f_1(x_1, f_2(x_2)) = z\}$. An example is proving knowledge of x such that $g^{H(x)} = z$ where H is a hash function. This composition commonly appears when proving knowledge of a digitally signed message.

- The prover commits to $x_1, x_2, x_3 = f_2(x_2)$ by computing $c_1 = \mathsf{Com}_q(x_1), c_2 = \mathsf{Com}_p(x_2), c_3 = \mathsf{Com}_q(x_3), c_3' = \mathsf{Com}_p(x_3)$. c_3 is committed to twice, in groups of order p and q. The prover sends c_1, c_2, c_3, c_3' to the verifier.
- The prover uses zk-comIOSnark to give a proof that $f_2(x_2) = x_3$, given c_2 and c_3'. $\mathsf{PK}\{(x_2, x_3', r_2, r_3') : f_2(x_2) = x_3' \wedge c_2 = \mathsf{Com}_p(x_2) \wedge c_3' = \mathsf{Com}_p(x_3')\}$.
- The prover uses the sigma protocol Π to give the following proof. $\mathsf{PK}\{(x_1, x_3, r_1, r_3) : f_1(x_1, x_3) = z \wedge c_1 = \mathsf{Com}_q(x_1) \wedge c_3 = \mathsf{Com}_q(x_3)\}$.
- The prover uses the protocol Eq to prove that c_3' and c_3 are commitments to the same value. $\mathsf{PK}\{(x_3, x_3', r_3, r_3') : x_3 \equiv x_3' \pmod q \wedge c_3 = \mathsf{Com}_q(x_3) \wedge c_3' = \mathsf{Com}_p(x_3')\}$

4. f_1 and f_2 are algebraic. Let $f_1 : \mathbb{Z}_p^3 \to \mathcal{G}_1, f_2 : \mathbb{Z}_q \to \mathcal{G}_2$, where $\mathcal{G}_1$ and $\mathcal{G}_2$ are elliptic curve groups of prime order p and q respectively. Let $T(k) = (k_x, k_y)$ for $k \in \mathcal{G}_2$, where (k_x, k_y) are the coordinates of the elliptic curve point. Let Π_1 be a Σ-protocol for f_1. Let $x_1 \in \mathbb{Z}_p, x_2 \in \mathbb{Z}_q$. An example is proving knowledge of x such that $g_1^{T(g_2^x)}$ for generators g_1 and g_2 for two different groups and a valid transformation T for mapping from one group to another. These statements often occur in anonymous credential constructions or proving statements about accumulators but the only previous constructions are for RSA groups.

- The prover commits to x_1, x_2 and $k = f_2(x_2)$ by computing $c_1 = \mathsf{Com}_p(x_1), c_2 = \mathsf{Com}_q(x_2), c_3 = \mathsf{Com}_p(k) = (\mathsf{Com}_p(k_x), \mathsf{Com}_p(k_y))$, and sends c_1, c_2, c_3 to the verifier.
- The prover uses the protocols ddlog and the sigma protocol on committed group element comSigma for f_2 to give the following proof: $\mathsf{PK}\{(x_2, k, r_2, r_3) : f_2(x_2) = k \wedge c_2 = \mathsf{Com}_q(x_2) \wedge c_3 = \mathsf{Com}_p(k)\}$.
- The prover uses the sigma protocol Π_1 to give the following proof. $\mathsf{PK}\{(x_1, k, r_1, r_3) : f_1(x_1, T(k)) = z \wedge c_1 = \mathsf{Com}_p(x_1) \wedge c_3 = \mathsf{Com}_p(k)\}$.

Theorem 5.1 (Function Composition)**.** *The constructions* composition *are non-interactive zero-knowledge arguments* $\mathsf{PK}\{(x_1, \ldots, x_n, y_1, \ldots, y_m) : f_1(x_1, \ldots, x_n, f_2(y_1, \ldots, y_m)) = z\}$*, as per Definition 2.2, for any* $f_1, f_2 \in \{\mathsf{algebraic}, \mathsf{arithmetic}\}$ *assuming the security of zk-*comInSnark*, zk-*comIOSnark*,* ddlog*,* Eq.

5.2 OR Composition

Consider the OR composition where a prover wants to show that $f_1(x_1, x_2) = 1$ or $f_2(x_1, x_3) = 1$ *but* without revealing which one is true. We give constructions compoundOR : $\mathsf{PK}\{(x_1, x_2, x_3) : f_1(x_1, x_2) \vee f_2(x_1, x_3) = 1\}$, where the f_is could have either an arithmetic or algebraic representation, and could have shared secret inputs.

1. f_1 and f_2 are functions represented as arithmetic circuits. Let $f_1 : \mathbb{F}_p^2 \to \{0, 1\}$, and $f_2 : \mathbb{F}_q^2 \to \{0, 1\}$, $q < p$. An example is composing proofs for two SNARK systems that work over different elliptic curve groups.

- The prover commits to the inputs by computing, $c_1 = \mathsf{Com}_p(x_1), c_1' = \mathsf{Com}_q(x_1), c_2 = \mathsf{Com}_p(x_2), c_3 = \mathsf{Com}_q(x_3)$, and to the output bits $b_1 = f_1(x_1, x_2), b_2 = f_1(x_1, x_3)$, $c_4 = \mathsf{Com}_p(b_1), c_5 = \mathsf{Com}_q(b_2), c_5' = \mathsf{Com}_p(b_2)$. x_1 and b_2 are committed to in both groups of order p and q.
- The prover uses zk-comIOSnark to give proofs.
  $\mathsf{PK}\{(x_1, x_2, b_1, r_1, r_2, r_4) : f_1(x_1, x_2) = b_1 \wedge c_1 = \mathsf{Com}_p(x_1) \wedge c_2 = \mathsf{Com}_p(x_2) \wedge c_4 = \mathsf{Com}_p(b_1)\}$.
  $\mathsf{PK}\{(x_1', x_3, b_2, r_1', r_3, r_5) : f_2(x_1', x_3) = b_2 \wedge c_1' = \mathsf{Com}_q(x_1') \wedge c_3 = \mathsf{Com}_q(x_3) \wedge c_5 = \mathsf{Com}_q(b_2)\}$.
- The prover uses the protocol Eq to prove that c_1' and c_1 are commitments to the same value.
  $\mathsf{PK}\{(x_1, x_1' r_1, r_1') : x_1 \equiv x_1' \pmod q \wedge c_1 = \mathsf{Com}_p(x_1) \wedge c_1' = \mathsf{Com}_q(x_1)\}$
- The prover uses the protocol Eq to prove that c_5' and c_5 are commitments to the same value.
  $\mathsf{PK}\{(b_2, b_2', r_5, r_5') : b_2 \equiv b_2' \pmod q \wedge c_5 = \mathsf{Com}_q(b_2) \wedge c_5' = \mathsf{Com}_p(b_2')\}$
- The prover uses the Sigma protocol OR-transform to give the following proof.
  $\mathsf{PK}\{(b_1, b_2, r_4, r_5) : (b_1 = 1 \wedge c_4 = \mathsf{Com}_p(b_1)) \vee (b_2 = 1 \wedge c_5' = \mathsf{Com}_p(b_2))\}$

2. One of them is an arithmetic circuit and the other is an algebraic relation. Wlog., f_1 is represented as an arithmetic circuit and f_2 is an algebraic statement. Let $f_1 : \mathbb{F}_p^2 \to \{0,1\}, f_2 : \mathbb{Z}_q^2 \to \{0,1\}, q < p$. Let Π be a Σ-protocol for f_2. An example is proving knowledge of x such that $H(x) = y$ OR $g^x = z$.

- The prover commits to the inputs, $c_1 = \mathsf{Com}_q(x_1), c_1' = \mathsf{Com}_p(x_1), c_2 = \mathsf{Com}_p(x_2), c_3 = \mathsf{Com}_q(x_3)$. The prover computes the outputs $b_1 = f_1(x_1, x_2), b_2 = f_1(x_1, x_3)$ and commits to them by computing $c_4 = \mathsf{Com}_p(b_1), c_5 = \mathsf{Com}_q(b_2), c_5' = \mathsf{Com}_p(b_2)$.
- The prover uses comIOSnark to give the following proof.
  $\mathsf{PK}\{(x_1', x_2, b_1, r_1', r_2, r_4) : f_1(x_1', x_2) = b \wedge c_1' = \mathsf{Com}_p(x_1) \wedge c_2 = \mathsf{Com}_p(x_2) \wedge c_4 = \mathsf{Com}_p(b_1)\}$.
- The prover uses the protocol Π and protocol comBitSigma (Figure 3) to prove the following.
  $\mathsf{PK}\{(x_1, x_3, b_2, r_1, r_3, r_5) : f_2(x_1, x_3) = b_2 \wedge c_1 = \mathsf{Com}_q(x_1) \wedge c_3 = \mathsf{Com}_q(x_3) \wedge c_5 = \mathsf{Com}_q(b_2)\}$
- The prover uses the protocol Eq to prove that c_1' and c_1 are commitments to the same value.
  $\mathsf{PK}\{(x_1, x_1' r_1, r_1') : x_1 \equiv x_1' \pmod q \wedge c_1 = \mathsf{Com}_q(x_1) \wedge c_1' = \mathsf{Com}_p(x_1)\}$
- The prover uses the protocol Eq to prove that c_5' and c_5 are commitments to the same value.
  $\mathsf{PK}\{(b_2, b_2', r_5, r_5') : b_2 \equiv b_2' \pmod q \wedge c_5 = \mathsf{Com}_q(b_2) \wedge c_5' = \mathsf{Com}_p(b_2')\}$
- The prover uses the Sigma protocol OR-transform to prove the following.
  $\mathsf{PK}\{(b_1, b_2, r_4, r_5) : (b_1 = 1 \wedge c_4 = \mathsf{Com}_q(b_1)) \vee (b_2 = 1 \wedge c_5 = \mathsf{Com}_q(b_2))\}$.

Let f_{OR} be the relation given by $f_{\mathsf{OR}} = \{((f_1, f_2), (x_1, x_2, x_3)) : ((x_1, x_2) \in R_{f_1}) \vee ((x_1, x_3) \in R_{f_2})\}$.

Theorem 5.2 (OR Composition)**.** *The constructions* compoundOR *are non-interactive zero-knowledge arguments* $\mathsf{PK}\{(x_1, x_2, x_3) : f_1(x_1, x_2) \vee f_2(x_1, x_3) = 1\}$*, as per Definition 2.2, for the relation* f_{OR}*, for any* $f_1, f_2 \in \{\mathsf{algebraic}, \mathsf{arithmetic}\}$*, assuming the security of zk-*comInSnark*, zk-*comIOSnark*,* comBitSigma*,* Eq*.*

5.3 AND Composition

Techniques shown in Section 5.2 extend for proofs of the form, $\mathsf{PK}\{(x_1, x_2, x_3) : f_1(x_1, x_2) \wedge f_2(x_1, x_3) = 1\}$ for all combinations of f_1 and f_2 being arithmetic and algebraic. In particular, to prove the AND of multiple statements, we use our building blocks $\mathsf{comInSnark}$ for the arithmetic part, Σ-protocol for the algebraic part, and Eq to switch between groups.

6 Applications

6.1 Privacy-preserving Audits of Bitcoin Exchanges

In this section, we show how to use our constructions for proving composite statements in zero-knowledge to build a privacy-preserving proof of solvency for Bitcoin exchanges. A proof of solvency demonstrates that an exchange controls sufficient reserves to settle each customer's account. If the exchange loses a large amount of money in an attack, it would not be able to provide such a proof. Thus customers will find out about the attack very soon and take necessary actions.

A proof of solvency consists of three components:

- A *proof of liabilities* that allows customers to verify that their accounts are included in the total.
- A *proof of assets* which shows that the exchange has a certain amount of reserves.
- A proof that the reserves cover the liabilities to an acceptable degree.

Let g, h be fixed public generators of a group G of order q. For a Bitcoin public key y, $x \in \mathbb{Z}_q$ is the corresponding secret key such that $y = g^x$. In the proof of assets below, for a group element $k = (k_x, k_y)$, we write $\mathsf{Com}(k)$ to mean a commitment to the coordinates of k, i.e. $\mathsf{Com}(k) = (\mathsf{Com}(k_x), \mathsf{Com}(k_y))$. The Bitcoin address corresponding to a key y is given by $\mathsf{h} = H(y)$, where H hashes y to a more compact representation. We denote the balance associated with an address h by $\mathsf{bal}(\mathsf{h})$.

6.1.1 Proof of assets

We give the proof of assets in Figure 7, which allows an exchange to generate a commitment to its total assets along with a zero-knowledge proof that the exchange knows the private keys for a set of Bitcoin addresses whose total value is equal to the committed value. The exchange creates a set of hashes $\mathcal{PK}$ to serve as an anonymity set: $\mathcal{PK} = \{\mathsf{h}_1, \cdots, \mathsf{h}_n\}$ from the public data available on the blockchain. Let $x_1, \cdots, x_n$ be the corresponding secret keys, so that $\mathsf{h}_i = H(g^{x_i})$, s_i indicates whether the exchange knows the ith secret key. The total assets can now be expressed as $\mathsf{Assets} = \sum_{i=1}^{n} s_i \cdot \mathsf{bal}(\mathsf{h}_i)$. The public data available on the blockchain is $\mathsf{h}_i = H(y_i), p_i = g^{\mathsf{bal}(\mathsf{h}_i)}$ for all $i \in [1, n]$.

6.1.2 Proof of liabilities

The proof of liabilities has the exchange commit to its total liability, and in addition, convince all its customers of the inclusion of their balances in the commitment. Like in Provisions, each customer is mapped to an entry on a liability list. Each customer is provided with an identifier $user_i$ (which could potentially include username, email address, or account number), and the exchange uses a hash-based commitment scheme to commit to the customer identifiers. To ensure that any included users can only add to the exchange's total liabilities, the protocol has the exchange give a proof that each

- The exchange computes the commitments. For $i \in [1, n]$, commit to x_i by publishing $\alpha_i = \mathsf{Com}_q(x_i) = g^{x_i} h^{r_i}$, and commit to y_i by publishing $\beta_i = \mathsf{Com}_q(y_i)$.
- The exchange commits to the balance in each address for the public keys he controls and to 0 otherwise, by publishing $u_i = \mathsf{Com}_q(s_i \cdot \mathsf{bal}(\mathsf{h}_i)) = g^{s_i \cdot \mathsf{bal}(\mathsf{h}_i)} h^{t_i}$, $s_i \in \{0, 1\}$, where $s_i = 1$ if the exchange knows x_i such that $y_i = g^{x_i}$.
- The exchange uses protocols ddlog, comIOSnark and the constructions for function composition and OR composition, composition and compoundOR respectively, to prove the following for each i,

$$\pi_i : \mathsf{PK}\{(x_i, y_i, s_i, r_i, a_i, b_i, t_i) : \big(\alpha_i = \mathsf{Com}_q(x_i) \wedge \beta_i = \mathsf{Com}_q(y_i) \wedge \\ u_i = \mathsf{Com}_q(s_i \cdot \mathsf{bal}(\mathsf{h}_i)) \wedge f_1(f_2(x_i), \mathsf{h}_i) = s_i \wedge s_i = 1\big) \vee \big(s_i = 0\big)\}$$

  where $f_2(x) = g^x$ and

$$f_1(y, \mathsf{h}) = \begin{cases} 1 & \text{if } H(y) = \mathsf{h} \\ 0 & \text{otherwise.} \end{cases}$$

- Compute and publish $Z_{Assets} = \prod_{i=1}^{n} u_i$.

Figure 7: Proof of assets

committed balance is in an interval between 0 and $\mathsf{Max} = 2^{51}$. While Provisions achieves this range proof by using bitwise commitments (which contributes to the bulk of the proof size), our comInSnark protocol for zk-SNARK on committed input allows us to use a circuit to check the range instead. The rest of the proof remains similar to Provisions, allowing the exchange to verifiably commit to its total liabilities Z_{Liab}, and convince clients of inclusion of their balances in Z_{Liab}. We give the proof of liabilities in Figure 8.

Given the proofs in Figures 7 and 8, the proof of solvency involves the exchange proving that Z_{Assets}/Z_{Liab} is a commitment to 0, and is similar to the protocol for proof of solvency in Provisions. For completeness, we include the proof in Figure 9. Zero-knowledge and soundness of the proofs of assets and liabilities follow from properties of our constructions for compound statements (Theorems 5.1, 5.2) and properties of the Sigma protocols used. We compare the trade-off between proof size and prover's work in our approach versus Provisions and a full SNARK solution in Table 1 in Appendix F.

6.2 Privacy-Preserving Credentials

Another application of our compositions for compound statements is in privacy-preserving verification of credentials. A credential system allows a user to obtain credentials from an organization or a Certificate Authority, and later prove to a verifier that she has been given appropriate credentials. Typically, the user's credentials will contain a set of attributes, and the verifier will require that the user prove that the attributes in his credential satisfy certain policy. Many different constructions have been proposed for anonymous credential systems built around sigma protocols. The signatures used, therefore, are specially designed so that a sigma protocol can be used to prove knowledge of the signature on a committed message. If we want to base anonymous credentials on standard signatures, like RSA signatures, we will need to prove a compound statement involving an algebraic relation (for the exponentiation), and a circuit-based statement (for the hash function). The recent work of [DLFKP16]

- Let C be a circuit that takes as input m bit integers $x_1, \cdots, x_n$ and outputs 1 if $x_i < \mathsf{Max}$ for all i and 0 otherwise.
- The exchange commits to each customer $\mathcal{C}_i$'s balance x_i by publishing $c_i = \mathsf{Com}_q(x_i) = g^{x_i}h^{r_i}$
- The exchange uses the protocol comInSnark to prove that $x_i < \mathsf{Max}$ for all customers.
  $\pi : \mathsf{PK}\{(x_i, r_i) : C(x_1, \cdots, x_n) = 1 \wedge c_i = \mathsf{Com}_q(x_i)\}$.
- The exchange computes a customer identifier for each customer by choosing a random nonce and computing
  $$\mathsf{CID}_i = H(user_i||n_i)$$
  where $n_i \in \{0,1\}^{512}$, $user_i$ is the ith customer's username, and H is a collision resistant hash function.
- The exchange publishes the liabilities list of all customers' tuples.
  $$\mathsf{ListLiab} = (\mathsf{CID}_1, \cdots, \mathsf{CID}_n, c_1, \cdots, c_n, \pi)$$
- Each client is privately given (r_i, n_i)
  - The client computes CID and verifies inclusion in the liabilities list.
  - The client checks its own balance is included by computing $c_i = g^{\mathsf{bal}_i}h^{r_i}$.
  - Verifies the proof π.
  - Each client computes $Z_{Liab} = \prod_{i=1}^{n} c_i$.

Figure 8: Proof of liabilities

1. The exchange uses the proof of assets in Figure 7 and generates a commitment to its total assets Z_{Assets}.
2. The exchange uses the proof of liabilities in Figure 8 to generate a commitment to its total liabilities Z_{Liab} and a list of its liabilities, ListLiab.
3. The exchange gives a proof $\pi : \mathsf{PK}\{(R) : Z = h^R\}$, where $Z = Z_{Assets} \cdot Z_{Liab}^{-1}$.

Figure 9: Proof of solvency

achieves privacy-preserving verification of X.509 certificates by using zk-SNARKs, and this involves representing the exponentiation in an RSA group as a circuit. Here, we use our composition constructions to build an efficient proof avoiding expensive circuit representation of algebraic statements.

Given a SHA hash digest of a message m, a candidate RSA signature σ, and an RSA modulus N, verification involves checking whether $\sigma^e \mod n = h$, where $h = \mathsf{padding}(\mathsf{SHA}(m))$. The construction given in Figure 10 achieves privacy-preserving verification for credentials based on RSA signatures. We compare the trade off between the proof size and prover's work in our approach versus other methods in Table 2 in Appendix F. Our compositions and similar techniques extend to yield efficient privacy-preserving verification for credentials based on existing infrastructure like standard RSA-PSS, RSA-PKCS etc.

- The prover commits to the message m, the digest h, and the signature σ by computing $c_1 = \mathsf{Com}_p(m)$, $c_2 = \mathsf{Com}_p(h)$, $c_3 = \mathsf{Com}_n(\sigma)$, $c_4 = \mathsf{Com}_n(h)$ for $p < n$.
- The prover uses zk-comIOSnark to give a proof that the hash digest is correct, given c_1 and c_2.
  $\mathsf{PK}\{(m, h, r_1, r_2) : \mathsf{padding}(\mathsf{SHA}(m)) = h \wedge c_1 = \mathsf{Com}_p(m) \wedge c_2 = \mathsf{Com}_p(h)\}$.
- The prover uses a sigma protocol to prove knowledge of e-th root of a committed value [CS97].
  $\mathsf{PK}\{(h, \sigma, r_2, r_3) : \sigma^e \mod n = h \wedge c_2 = \mathsf{Com}_n(h) \wedge c_3 = \mathsf{Com}_n(\sigma)\}$.
- The prover uses the protocol Eq to prove that the commitments c_2 and c_4 are to the same value: $\mathsf{PK}\{(h, h', r_2, r_4) : c_2 = \mathsf{Com}_p(h) \wedge c_4 = \mathsf{Com}_n(h') \wedge h \equiv h' \mod p\}$.

Figure 10: RSA Signature Verification

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 17*, pages 2087–2104. ACM Press, October / November 2017.

[Aho87] Alfred Aho, editor. *19th ACM STOC*. ACM Press, May 1987.

[BBB+17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Efficient range proofs for confidential transactions. Cryptology ePrint Archive, Report 2017/1066, 2017. https://eprint.iacr.org/2017/1066.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013.

[BCG+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. IEEE Computer Society Press, May 2014.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 781–796, San Diego, CA, 2014. USENIX Association.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[bit] Technical background of version 1 bitcoin addresses. https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses.

[BOGG+90] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, August 1990.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444. Springer, Heidelberg, May 2000.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1825–1842. ACM, 2017.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th FOCS*, pages 372–382. IEEE Computer Society Press, October 1985.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society Press, May 2015.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 499–530. Springer, Heidelberg, August 2016.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 107–122. Springer, Heidelberg, May 1999.

[Cra12] Ronald Cramer, editor. *TCC 2012*, volume 7194 of *LNCS*. Springer, Heidelberg, March 2012.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Kaliski Jr. [Kal97], pages 410–424.

[Dåm] Ivan Dåmgard. On Sigma Protocols. `http://www.cs.au.dk/~ivan/Sigma.pdf`.

[DBB$^+$15] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 720–731. ACM Press, October 2015.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2002.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Cramer [Cra12], pages 54–74.

[DLFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254. IEEE Computer Society Press, May 2016.

[Edw07] Harold Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

[FFG$^+$16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1304–1316. ACM Press, October 2016.

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In Aho [Aho87], pages 210–217.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Kaliski Jr. [Kal97], pages 16–30.

[For87] Lance Fortnow. The complexity of perfect zero-knowledge (extended abstract). In Aho [Aho87], pages 204–209.

[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.

[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.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280. Springer, Heidelberg, April 2015.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-Proofs. Cryptology ePrint Archive, Report 2011/456, 2011. `http://eprint.iacr.org/2011/456`.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Aho [Aho87], pages 218–229.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[IEE13] *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[Kal97] Burton S. Kaliski Jr., editor. *CRYPTO'97*, volume 1294 of *LNCS*. Springer, Heidelberg, August 1997.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Cramer [Cra12], pages 169–189.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Heidelberg, December 2013.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In IEEE S&P 2013 [IEE13], pages 397–411.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[NMT] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring confidential transactions. `https://lab.getmonero.org/pubs/MRL-0005.pdf`.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[para] Zcash 1.0 “Sprout” Guide. `https://github.com/zcash/zcash/wiki/1.0-User-Guide`.

[parb] Zcash Parameter Generation. `https://z.cash/technology/paramgen.html`.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO’91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In IEEE S&P 2013 [IEE13], pages 238–252.

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

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[sec] Secp256k1. `https://en.bitcoin.it/wiki/Secp256k1`.

[Vad99] Salil Pravin Vadhan. *A study of statistical zero-knowledge proofs*. PhD thesis, Massachusetts Institute of Technology, 1999.

[Wil] Zooko Wilcox. Proving bitcoin reserves. `https://iwilcox.me.uk/2014/proving-bitcoin-reserves`.

[Woo] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. `http://gavwood.com/paper.pdf`.

A zk-SNARK on Committed Input/Output

We separate the circuit wires into private input, public input, intermediate values and private output. Let $I_{com} \subseteq \{1, \ldots, m\}$ be the set of indices corresponding to the private inputs $a_1, \ldots, a_n$, and I_{pub} the indices for the public input wires. Let I_{out} be the set of indices corresponding to the outputs b_i, and $I_{mid} = \{1, \ldots, m\} \setminus I_{pub} \cup I_{com} \cup I_{out}$. Let $C_i = g^{a_i} h^{r_i}$ be a Pedersen commitment, to the ith input a_i and $D_i = g^{b_i} h^{R_i}$, commitment to the outputs. The construction $\mathsf{comIOSnark}$ is given in Fig. 11.

Given $C_i = g^{a_i} h^{r_i}$, for all $i \in [n]$, commitments to private inputs, $D_i = g^{b_i} h^{R_i}$, for all $i \in [n']$, commitments to private outputs, and public input z.

1. CRS generation: Run $\mathsf{GroupGen}(1^\kappa)$ to get $(p, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, g, \tilde{g}, e)$. Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, \tilde{g}_v = \tilde{g}^{r_v}, g_w = g^{r_w}, \tilde{g}_w = \tilde{g}^{r_w}, g_y = g^{r_y}, \tilde{g}_y = \tilde{g}^{r_y}$.
   Set the CRS to be:
$$\begin{aligned}\mathsf{crs} = \Big(&\{g_v^{v_k(s)}\}_{k\in I_{com}}, \{g_v^{v_k(s)}\}_{k\in I_{out}}, \{g_v^{v_k(s)}\}_{k\in I_{mid}},\\ &\{\tilde{g}_w^{w_k(s)}\}_{k\in I_{com}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{out}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{mid}}, \{g_y^{y_k(s)}\}_{k\in I_{com}},\\ &\{g_y^{y_k(s)}\}_{k\in I_{out}}, \{g_y^{y_k(s)}\}_{k\in I_{mid}}, \{\tilde{g}_v^{\alpha_v v_k(s)}\}_{k\in I_{com}}, \{\tilde{g}_v^{\alpha_v v_k(s)}\}_{k\in I_{out}},\\ &\{\tilde{g}_v^{\alpha_v v_k(s)}\}_{k\in I_{mid}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{com}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{out}}, \{g_w^{\alpha_w w_k(s)}\}_{k\in I_{mid}},\\ &\{\tilde{g}_y^{\alpha_y y_k(s)}\}_{k\in I_{com}}, \{\tilde{g}_y^{\alpha_y y_k(s)}\}_{k\in I_{out}}, \{\tilde{g}_y^{\alpha_y y_k(s)}\}_{k\in I_{mid}}, \{g^{s^i}\}_{i\in[d]},\\ &\{\tilde{g}^{s^i}\}_{i\in[d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{com}},\\ &\{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{out}}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k\in I_{mid}}\Big)\end{aligned}$$
   Set the short verification CRS to be:
$$\begin{aligned}\mathsf{shortcrs} = \Big(&g, \tilde{g}, \tilde{g}^{\alpha_v}, g^{\alpha_w}, \tilde{g}^{\alpha_y}, \tilde{g}^{\gamma}, g^{\beta\gamma}, \tilde{g}^{\beta\gamma}, g_y^{t(s)},\\ &\{g_v^{v_k(s)}\}_{k\in I_{com}\cup I_{out}}, \{g_v^{v_k(s)}\}_{k\in I_{pub}}, \{\tilde{g}_w^{w_k(s)}\}_{k\in I_{pub}}, \{g_y^{y_k(s)}\}_{k\in I_{pub}}\Big)\end{aligned}$$
2. Prove. On input z, witness $a_1, \ldots, a_n, b_1, \ldots, b_{n'}$, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i\in[m]}$. (Equivalently, evaluates the circuit to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{com}(x) = \sum_{k\in I_{com}} a_k v_k(x)$, $v_{mid}(x) = \sum_{k\in I_{mid}} a_k v_k(x)$, $v_{out}(x) = \sum_{k\in I_{out}} a_k v_k(x)$ and similarly define $w_{com}(x)$, $w_{mid}(x)$, $w_{out}(x)$, $y_{com}(x)$, $y_{mid}(x)$ and $y_{out}(x)$.
   - The prover computes the proof π:
$$\begin{aligned}\Big(&g_v^{v_{com}(s)}, g_v^{v_{mid}(s)}, g_v^{v_{out}(s)}, \tilde{g}_w^{w_{com}(s)}, \tilde{g}_w^{w_{mid}(s)},\\ &\tilde{g}_w^{w_{out}(s)}, g_y^{y_{com}(s)}, g_y^{y_{mid}(s)}, g_y^{y_{out}(s)}, \tilde{g}^{h(s)}, \tilde{g}_v^{\alpha_v v_{com}(s)}, \tilde{g}_v^{\alpha_v v_{mid}(s)},\\ &\tilde{g}_v^{\alpha_v v_{out}(s)}, g_w^{\alpha_w w_{com}(s)}, g_w^{\alpha_w w_{mid}(s)}, g_w^{\alpha_w w_{out}(s)}, \tilde{g}_y^{\alpha_y y_{com}(s)}, \tilde{g}_y^{\alpha_y y_{mid}(s)},\\ &\tilde{g}_y^{\alpha_y y_{out}(s)}, g_v^{\beta v_{com}(s)} g_w^{\beta w_{com}(s)} g_y^{\beta y_{com}(s)}, g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)},\\ &g_v^{\beta v_{out}(s)} g_w^{\beta w_{out}(s)} g_y^{\beta y_{out}(s)}\Big)\end{aligned}$$
   - Prove input consistency with commitment. The prover uses sigma protocol comEq to compute proof π_{in}: $\mathsf{PK}\{(a_1, \ldots, a_n, r_1, \ldots, r_n) : y = \prod_{i=1}^n G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \cdots \wedge C_n = g^{a_n} h^{r_n}\}$, for $G_i = g_v^{v_i(s)}, i \in I_{com}$, and $y = g_v^{v_{com}(s)}$.

- Prove output consistency with commitment. The prover uses sigma protocol comEq to compute proof π_{out}: $\mathsf{PK}\{(b_1,\ldots,b_{n'},R_1,\ldots,R_{n'}) : y = \prod_{i=1}^{n'} G_{m-n'+i}^{b_i} \wedge D_1 = g^{b_1}h^{R_1} \wedge \cdots \wedge D_{n'} = g^{b_{n'}}h^{R_{n'}}\}$, for $G_j = g_v^{v_j(s)}, j \in I_{out}$, and $y = g_v^{v_{out}(s)}$

3. Verify.

- On input $\mathsf{shortcrs}$, y, and a proof π, parse it as

$$\pi = \Big(g^{V_{com}}, g^{V_{mid}}, g^{V_{out}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, \tilde{g}^{W_{out}}, g^{Y_{com}}, g^{Y_{mid}}, g^{Y_{out}}, \tilde{g}^{H}, \tilde{g}^{V'_{com}}, \tilde{g}^{V'_{mid}}, \tilde{g}^{V'_{out}}, g^{W'_{com}}, g^{W'_{mid}}, g^{W'_{out}}, \tilde{g}^{Y'_{com}}, \tilde{g}^{Y'_{mid}}, \tilde{g}^{Y'_{out}}, g^{Z_{com}}, g^{Z_{mid}}, g^{Z_{out}}\Big)$$

- Divisibility check. Compute $g_v^{v_{pub}(s)} = \prod_{k \in I_{pub}} (g_v^{v_k(s)})^{a_k}$. Similarly, compute $\tilde{g}_w^{w_{pub}(s)}$ and $g_y^{y_{pub}(s)}$. Check that,

$$e\left(g_v^{v_0(s)} g_v^{v_{pub}(s)} g_v^{V_{com}} g_v^{V_{mid}} g_v^{V_{out}}, \tilde{g}_w^{w_0(s)} \tilde{g}_w^{w_{pub}(s)} \tilde{g}_w^{W_{com}} \tilde{g}_w^{W_{mid}} \tilde{g}_w^{W_{out}}\right) = e\left(g_y^{t(s)}, \tilde{g}^{H}\right) e\left(g_y^{y_0(s)} g_y^{y_{pub}(s)} g_y^{Y_{com}} g_y^{Y_{mid}} g_y^{Y_{out}}, \tilde{g}\right)$$

- Verify that the linear combinations are in correct spans.
  (a) $e(g^{V_{com}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{com}})$
  (b) $e(g^{V_{mid}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{mid}})$
  (c) $e(g^{V_{out}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{out}})$
  (d) $e(g^{W'_{com}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{com}})$
  (e) $e(g^{W'_{mid}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{mid}})$
  (f) $e(g^{W'_{out}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{out}})$
  (g) $e(g^{Y_{com}}, \tilde{g}^{\alpha_y}) = e(g, \tilde{g}_y^{Y'_{com}})$
  (h) $e(g^{Y_{mid}}, \tilde{g}^{\alpha_y}) = e(g, \tilde{g}_y^{Y'_{mid}})$
  (i) $e(g^{Y_{out}}, \tilde{g}^{\alpha_y}) = e(g, \tilde{g}_y^{Y'_{out}})$
- Verify same coefficients in all linear combinations.
  (a) $e(g^{Z_{com}}, \tilde{g}^{\gamma}) = e(g^{V_{com}} g^{Y_{com}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{com}})$
  (b) $e(g^{Z_{mid}}, \tilde{g}^{\gamma}) = e(g^{V_{mid}} g^{Y_{mid}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{mid}})$
  (c) $e(g^{Z_{out}}, \tilde{g}^{\gamma}) = e(g^{V_{out}} g^{Y_{out}}, \tilde{g}^{\beta\gamma}) e(g^{\beta\gamma}, \tilde{g}^{W_{out}})$
- Verify input consistency with commitment. Verify comEq proof π_{in}. The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{com}$, and sets $y = g^{V_{com}}$ from the proof π. The verifier checks that the proof π_{in} is a proof of knowledge of: $\mathsf{PK}\{(a_1,\ldots,a_n,r_1,\ldots,r_n) : y = \prod_{i=1}^{n} G_i^{a_i} \wedge C_1 = g^{a_1}h^{r_1} \wedge \cdots \wedge C_n = g^{a_n}h^{r_n}\}$.
- Verify output consistency with commitment. Verify comEq proof π_{out}. The verifier computes $G_i = g_v^{v_i(s)}, i \in I_{out}$, and sets $y = g^{V_{out}}$ from the proof π. The verifier checks that the proof π_{out} verifies. $\mathsf{PK}\{(b_1,\ldots,b_{n'},R_1,\ldots,R_{n'}) : y = \prod_{i=1}^{n'} G_{m-n'+i}^{b_i} \wedge D_1 = g^{b_1}h^{R_1} \wedge \cdots \wedge D_{n'} = g^{b_{n'}}h^{R_{n'}}\}$.

Figure 11: $\mathsf{comIOSnark} : \mathsf{PK}\{(a_1,\ldots,a_n,b_1,\ldots,b_{n'},r_1,\ldots,r_n,R_1,\ldots,R_{n'}) : f(a_1,\ldots a_n,z) = (b_1,\ldots,b_{n'}) \wedge C_i = g^{a_i}h^{r_i} \wedge D_i = g^{b_i}h^{R_i}\}$

The construction $\mathsf{comIOSnark}$ is made zero-knowledge by randomizing the elements in the proof π in a way similar to $\mathsf{comInSnark}$ and we obtain zk-$\mathsf{comIOSnark}$. The proof of the above is omitted, and follows from ideas similar to the proof of Theorem 4.1.

B Proof of Theorem 4.1

We recall a technical lemma from Gennaro et al. [GGPR13] below, on which we rely for soundness.

Lemma B.1 (Lemma 10, [GGPR13]). *Let $\mathbb{F}[x]^{(k)}$ denote polynomials over $\mathbb{F}[x]$ of degree at most k. Let $\mathbb{F}[x]^{(\neg k)}$ denote polynomials over $\mathbb{F}[x]$ that have a zero coefficient for x^k. For some d, let $\mathcal{U} = \{u_k(x)\} \subset \mathbb{F}[x]^d$, and let $\mathsf{span}(\mathcal{U})$ denote the set of polynomials that can be generated as $\mathbb{F}$-linear combinations of the polynomials in $\mathcal{U}$. Let $a(x) \in \mathbb{F}[x]^{(d+1)}$ be generated uniformly at random subject to the constraint that $\{a(x) \cdot u_k(x) : u_k(x) \in \mathcal{U}\} \subset \mathbb{F}[x]^{(\neg(d+1))}$. Let $s \in \mathbb{F}^*$. Then, for all algorithms $\mathcal{A}$*

$$\Pr[u(x) \leftarrow \mathcal{A}(\mathcal{U}, s, a(s)) : u(x) \in \mathbb{F}[x]^d \wedge u(x) \notin \mathsf{span}(\mathcal{U}) \wedge a(x) \cdot u(x) \in \mathbb{F}[x]^{(\neg(d+1))}] \leq \frac{1}{|\mathbb{F}|}$$

Proof of Theorem 4.1.

Proof. **Soundness.** Assume there exists an adversary $\mathcal{A}$ who returns the proof of a false statement. We use this adversary $\mathcal{A}$ along with the knowledge extractor that exists by the d-PKE assumption to construct an adversary $\mathcal{B}$ to break either the q-PDH assumption or the $2q$-SDH assumption. $\mathcal{B}$ is given the description of a bilinear map, $(p, \mathbb{G}, \widetilde{\mathbb{G}}, \mathbb{G}_T, g, \tilde{g}, e)$, and the challenge $g^s, \tilde{g}^s, \cdots, g^{s^q}, \tilde{g}^{s^q}, g^{s^{q+2}}, \tilde{g}^{s^{q+2}}, \cdots, g^{s^{2q}}, \tilde{g}^{s^{2q}}$. $\mathcal{A}$ generates a function f that has a QAP $\mathcal{Q} = (t(x), \mathcal{V}, \mathcal{W}, \mathcal{Y})$ of size m and degree d.

$\mathcal{B}$ first picks $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s$ at random and sets $r_y = r_v r_w, g_v = g^{r_v s^{d+1}}$, $g_w = g^{r_w s^{2(d+1)}}$, and $g_y = g^{r_y s^{3(d+1)}}$. Using these values, the final term in the proof π can be rewritten as

$$g_v^{\beta v(s)} g_w^{\beta w(s)} g_y^{\beta y(s)} = g^{\beta(r_v s^{d+1} v(s) + r_w s^{2(d+1)} w(s) + r_y s^{3(d+1)} y(s))}. \tag{8}$$

$\mathcal{B}$ sets $\beta = s^{q-(4d+3)} \beta_{poly}(s)$ where $\beta_{poly}(x)$ is a polynomial of degree at most $3d+3$ sampled uniformly at random such that $\beta_{poly}(x) \cdot (r_v v_k(x) + r_w x^{(d+1)} w_k(x) + r_y x^{2(d+1)} y_k(x))$ has a zero coefficient in front of x^{3d+3} for all k. Such a polynomial is guaranteed to exist by Lemma B.1. Rewriting (8) by writing β in terms of s, we have

$$\begin{aligned} & g^{s^{q-3d-2} r_v \beta_{poly}(s) v(s) + s^{q-2d-1} r_w \beta_{poly}(s) w(s) + s^{q-d} r_y \beta_{poly}(s) y(s)} \\ &= g^{s^{q-3d-2} \beta_{poly}(s)(r_v v(s) + s^{d+1} r_w w(s) + s^{2d+2} r_y y(s))}. \end{aligned} \tag{9}$$

Since $\beta_{poly}(x) \cdot (r_v v_k(x) + r_w x^{(d+1)} w_k(x) + r_y x^{2(d+1)} y_k(x))$ has a zero coefficient in front of x^{3d+3}, the exponent in (9) has a zero in front of s^{q+1}. The powers of q in the exponent go up to $(q - 3d - 2) + (3d + 3) + (2d + 2) + d = q + 3d + 3 \leq 2q$. Thus, $\mathcal{B}$ can efficiently generate the terms in the CRS that contain β by using the elements in the challenge.

$\mathcal{B}$ picks γ' uniformly at random from $\mathbb{F}$ and sets $\gamma = \gamma' s^{q+2}$. $\mathcal{B}$ can generate g^γ from the challenge, since $g^{s^{q+2}}$ is given. Note that $\beta\gamma = s^{q-(4d+3)} \beta_{poly}(s) \gamma' s^{q+2}$ does not have the s^{q+1} term, and it has degree at most $q - (4d+3) + (3d+3) + (q+2) \leq 2q$ (assuming $d \geq 2$). Hence, $\mathcal{B}$ can generate $g^{\beta\gamma}$ using the elements in its challenge.

Let $(\widehat{\pi}, \widehat{\pi_{in}})$ be a cheating proof returned by $\mathcal{A}$ for the computation of f with public input and public output $\{c_k\}_{k \in I_{pub} \cup I_{out}}$. Let $\widehat{\pi} = (g^{V_{com}}, g^{V_{mid}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, g^{Y_{com}}, g^{Y_{mid}}, \tilde{g}^{H}, g^{V'_{com}}, g^{V'_{mid}}, g^{W'_{com}}, g^{W'_{mid}}, g^{Y'_{com}}, g^{Y'_{mid}}, g^{Z_{com}}, g^{Z_{mid}})$. Since the verification holds, we have that $e(g^{V_{com}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{com}})$ and $e(g^{V_{mid}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{mid}})$. $\mathcal{B}$ can run the d-PKE extractor to recover polynomials $V_{mid}(x)$ and $V_{com}(x)$ of degree at most d such that $V_{mid} = V_{mid}(s), V_{com} = V_{com}(s)$. Note that the parameters received by $\mathcal{A}$ is a valid input for the d-PKE assumption from which all the other terms in the CRS can be efficiently generated. That is, the d-PKE adversary receives input (σ, z), where $\sigma = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, \{g^{s^i}\}_{i \in [0,d]}, \{\tilde{g}^{s^i}\}_{i \in [0,d]})$, and the auxiliary input z consists of all the other terms in the CRS. Note that the terms $\{g_v^{v_k(s)}\}$ can be efficiently generated from σ. $\mathcal{A}$ returns (V, V') such that $e(V, \tilde{g}^{\alpha_v}) = e(g, V')$. Thus, $\mathcal{B}$ can invoke the d-PKE extractor χ_A to recover a polynomial $V_{com}(x) = \sum_{i=0}^{d} c_i x^i$ of degree at most d such that $V = g^{V_{com}(s)}$. Similarly, $\mathcal{B}$ recovers polynomials $W_{mid}(x), W_{com}(x), Y_{mid}(x), Y_{com}(x)$ such that $W_{mid} = W_{mid}(s), W_{com} = W_{com}(s), Y_{mid} = Y_{mid}(s), Y_{com} = Y_{com}(s)$. Now, $\mathcal{B}$ computes,

$$V(x) = v_0(x) + \sum_{k \in I_{pub}} c_k v_k(x) + \sum_{k \in I_{out}} c_k v_k(x) + V_{com}(x) + V_{mid}(x)$$

and similarly $W(x)$ and $Y(x)$, and sets $H(x) = (V(x)W(x) - Y(x))/t(x)$

Since the proof is of a false statement, either the extracted polynomials do not form a QAP solution, or the co-efficients of the extracted com polynomials are not equal to the values committed to in $C_1, \ldots, C_n$. There are the following cases:

- $H(x)$ has a non-trivial denominator.
- The polynomial $R(x) = r_v x^{d+1} V_{mid}(x) + r_w x^{2(d+1)} W_{mid}(x) + r_y x^{3(d+1)} Y_{mid}(x)$ is not in the linear subspace generated by the polynomials $\{r_k(x) = r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x)\}_{k \in I_{mid}}$
- The polynomial $S(x) = r_v x^{d+1} V_{com}(x) + r_w x^{2(d+1)} W_{com}(x) + r_y x^{3(d+1)} Y_{com}(x)$ is not in the linear subspace generated by the polynomials $\{r_k(x) = r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x)\}_{k \in I_{com}}$
- By the soundness of the protocol comEq, there exists an extractor that extracts $a_1, \ldots, a_n$ such that $V_{com}(s) = \sum_{k \in I_{com}} a_k v_k(s), C_i = g^{a_i} h^{r_i}$. $a_1, \cdots, a_n$ are different from the coefficients c_i of the polynomial V_{com} extracted by the d-PKE extractor.

If none of the above cases hold, then $V(x), W(x), Y(x)$ are a QAP solution, with input consistent with commitments C_i.

Case 1 $t(x)$ does not divide $p(x) = V(x)W(x) - Y(x)$. Let $(x - r)$ be a polynomial that divides $t(x)$ but not $p(x)$, and let $T(x) = t(x)/(x - r)$. Let $d(x) = \gcd(t(x), p(x))$. $t(x)$ has degree at most d and $p(x)$ has degree at most $2d$. $\mathcal{B}$ can use the extended Euclidean algorithm to find polynomials $a(x), b(x)$ with degrees $2d - 1$ and $d - 1$ respectively, such that $a(x)t(x) + b(x)p(x) = d(x)$. Now set $A(x) = a(x) \cdot (T(x)/d(x))$ and $B(x) = b(x) \cdot (T(x)/d(x))$. $A(x)$ and $B(x)$ do not have any denominator since $d(x)$ divides $T(x)$. We have, $A(x)t(x) + B(x)p(x) = T(x)$. Dividing by $t(x)$ we have, $A(x) + B(x)H(x) = \dfrac{1}{(x - r)}$. $A(x)$ and $B(x)$ have degree at most $2d - 1 \leq q$; hence, $\mathcal{B}$ can use the terms in its challenge to compute $e(g^{A(s)}, \tilde{g}) e(g^{B(s)}, \tilde{g}^{H}) = e(g, \tilde{g})^{1/(s-r)}$ which solves the $2q$-SDH.

Case 2 There does not exist $\{c_k\}_{k\in I_{mid}}$ such that $V_{mid}(x) = \sum_{k\in I_{mid}} c_k v_k(x), W_{mid}(x) = \sum_{k\in I_{mid}} c_k w_k(x)$ and $Y_{mid}(x) = \sum_{k\in I_{mid}} c_k y_k(x)$. By Lemma B.1, we have that $x^{q-(4d+3)}\beta_{poly}(x)(r_v x^{d+1} v_k(x) + r_w x^{2(d+1)} w_k(x) + r_y x^{3(d+1)} y_k(x))$ has a non-zero coefficient for the x^{q+1} term with high probability. $\mathcal{B}$ can use $g^{Z_{mid}} = g^{s^{q-(4d+3)}\beta_{poly}(x)(s^{d+1}V_{mid}(s)+s^{2(d+1)}W_{mid}(s)+s^{3(d+1)}Y_{mid}(s))}$ to subtract off all elements of the form g^{s^j} for $j \neq q+1$, and obtain $g^{s^{q+1}}$. This breaks the q-PDH assumption.

Case 3 Similar to Case 2 with V_{com} polynomial, and using $g^{Z_{com}}$.

Case 4 This breaks the binding property of the multi-commitment y, since we have $y = \prod_{i\in I_{com}} G_i^{a_i} = \prod_{i\in I_{com}} G_i^{c_i}$, $a_i \neq c_i$ for some $i \in I_{com}$.

Zero-knowledge. We now show a simulator (S, Sim) such that S outputs a simulated crs and trapdoor, and Sim outputs a simulated proof. S generates crs in the same way and sets the trapdoor τ to be $\tau = (s, \alpha_v, \alpha_w, \alpha_y, \beta, \gamma)$. Sim, given the trapdoor τ picks polynomials $v(x), w(x)$ at random such that $t(x)$ divides $v(x)w(x)$. It sets $h(x)$ to be the quotient polynomial. Now, it chooses polynomials $v_{com}(x), w_{com}(x)$ at random, and sets $v_{mid}(x) = v(x) - v_0(x) - v_{io}(x) - v_{com}(x)$, and $w_{mid}(x) = w(x) - w_0(x) - w_{io}(x) - w_{com}(x)$. Given these polynomials, and s, α, β, γ from the trapdoor, Sim can compute the encodings of $V_{mid} = v_{mid}(s), V_{com} = v_{com}(s)$, and other elements of the proof. Moreover, the simulated proof elements are statistically uniform, subject to the verification constraints. By the zero-knowledge property of the protocol comEq, there exists a simulator that is invoked by Sim to generate a simulated proof that is statistically indistinguishable from π_{in}.

□

C Other Proofs

C.1 Protocol comEq

We show that the protocol comEq in Figure 5 is correct, has a soundness error of $1/2^k$ for a challenge length k and is honest verifier zero knowledge.

Proof.

- **Completeness:** If the prover and the verifier behave honestly, it is easy to see that verification conditions hold.

$$y^c \prod G_i^{s_i} = (\prod G_i^{a_i})^c \prod G_i^{s_i} = \prod G_i^{a_i c + s_i} = u$$

$$(C_i)^c g^{s_i} h^{t_i} = (g^{a_i} h^{r_i})^c g^{s_i} h^{t_i} = g^{ca_i + s_i} h^{cr_i + t_i} = v_i$$

- **Soundness:** We show an extractor that computes $a_1, \ldots a_n, r_1, \ldots, r_n$ given two accepting views with same commitments but different challenge strings. Say, we have two accepting views: $\{(u, v_i), c, (s_i, t_i)\}$ and $\{(u, v_i), \hat{c}, (\hat{s}_i, \hat{t}_i)\}$ for challenges c and $\hat{c} \neq c$. Since the views are accepting, we have,

$$y^c \prod G_i^{s_i} = y^{\hat{c}} \prod G_i^{\hat{s}_i} = u$$
$$y^{c-\hat{c}} = \prod G_i^{\hat{s}_i - s_i}$$

We can now compute (in $\mathbb{Z}_q$), $a_i = (\hat{s}_i - s_i)(c - \hat{c})^{-1}$. The inverse of $(c - \hat{c})$ exists in $\mathbb{Z}_q$, since $c \neq \hat{c}$ by assumption.

Similarly,

$$(C_i)^c g^{s_i} h^{t_i} = (C_i)^{\hat{c}} g^{\hat{s}_i} h^{\hat{t}_i} = v_i$$

and we can compute

$$r_i = (\hat{t}_i - t_i)(c - \hat{c})^{-1}$$

The extractor succeeds in extracting a witness given two accepting transcripts. The prover can, therefore, cheat only when he can answer exactly one challenge correctly, and the probability of that challenge being chosen by the verifier is bounded by $1/2^k$ where k is the length of the challenge.

- **Honest Verifier Zero-Knowledge:** We show a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. The simulator on input c, randomly chooses $s_i, t_i \in \mathbb{Z}_q$ and computes $u = y^c \prod G_i^{s_i}$, and $v_i = (C_i)^c g^{s_i} h^{t_i}$. It is easy to verify that the transcript output by the simulator will pass the verification equations. Moreover, the distribution of the output of the simulator is identical to the distribution of a transcript produced by the protocol between the prover and the verifier.

□

C.2 Protocol pointAddition

We show that the protocol addition is honest verifier zero-knowledge, and sound with a soundness error of $\frac{1}{2^k}$, where k is the length of the challenge.

- **Honest verifier zero-knowledge.** The simulator invokes the simulator for the proofs $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$. Zero-knowledge follows from the zero-knowledge of these proofs.

- **Soundness.** We show an extractor that computes $P = (P_x, P_y), Q = (Q_x, Q_y)$ such that T = P+Q, given two accepting transcripts for two different challenge bits. Say, we have two accepting views for challenge bits c and $\hat{c} \neq c$.

  From the soundness of proofs π_1, π_3, π_4, we can extract P_x, P_y, Q_x, Q_y such that $L_x(P_x, P_y, Q_x, Q_y)$ and $R_x(P_x, P_y, Q_x, Q_y)$ satisfy the following.

  $L_x = k_1 t + r_1 \mod q, R_x = k'_1 t + r'_1 \mod q$ Now,

$$\begin{aligned} L_x \mod t &= ((k_1 t + r_1) \mod q) \mod t \\ &= (k_1 t + r_1) \quad (\text{ Since } q > t^3, k_1 < q/t, r_1 < t) \\ &= r_1 \mod t \quad (\text{ Since } r_1 < t) \\ &= r'_1 \mod t \quad (\text{ Since } r_1 = r'_1 \mod q, r'_1 < t) \\ &= R_x \mod t \quad (\text{ Since } k'_1 < q/t, r'_1 < t) \end{aligned}$$

  Similarly, from soundness of π_2, π_3, π_5 we get, $L_y = R_y \mod t$

C.3 Protocol ddlog

Proof. We will show that the protocol ddlog is honest verifier zero-knowledge, and sound with a soundness error of $\frac{1}{2^k}$, where k is the length of the challenge.

- **Honest verifier zero-knowledge.** We can construct a simulator such that the output of the simulator is statistically indistinguishable from the transcript of the protocol with a prover. On input a bit c, the simulator does the following: if $c = 0$, the simulator randomly chooses $z_1, z_2 \in \mathbb{Z}_p, z_3, z_4 \in \mathbb{Z}_q$, and computes $a_1 = z_1 P + z_2 Q, a_2 = \gamma_1 P' + z_3 Q', a_3 = \gamma_2 P' + z_4 Q'$ for $(\gamma_1, \gamma_2) = z_1 P$. It is easy to see that the output of the simulator is distributed identically with the distribution of the protocol transcript. If $c = 1$, the simulator randomly chooses $z_1 \in \mathbb{Z}_t$ and invokes the simulator for the proof $\mathsf{pointAddition}$. Zero-knowledge follows from the zero-knowledge of the proof π.

- **Soundness.** We show an extractor that computes λ, x, y given two accepting transcripts for two different challenge bits. Say, we have two accepting views for challenge bits c and $\hat{c} \neq c$. We have,

$$\lambda = z_1 - \hat{z}_1 \mod t$$

  From the soundness of proofs π, we can extract x, y, γ_1, γ_2 such that $L_x(x, y, \gamma_1, \gamma_2) = R_x(x, y, \gamma_1, \gamma_2)$, and $L_y(x, y, \gamma_1, \gamma_2) = R_y(x, y, \gamma_1, \gamma_2)$. Thus, $T = (\gamma_1, \gamma_2) - (x, y) = \hat{z}_1 P - \lambda P$. Thus $\lambda P = (x, y)$.

□

D Proof of Equality of Committed Values across Groups

There are techniques known to prove that two committed values in different groups are equal. The integer commitment technique of Damgård and Fujisaki [DF02] allows one to prove properties about discrete logarithms in $\mathbb{Z}$ instead of modulo the order of the group. This technique could be fairly expensive as the group order is hidden and exponentiations need to be computed in an RSA group with large exponents. We give a protocol below for proving equality of committed values in different groups without using integer commitments. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two groups of order p and q respectively, let g be a generator of $\mathcal{G}_1$, and G a generator of $\mathcal{G}_2$.

The protocol Eq in Figure 12 is honest verifier zero-knowledge, and sound with a soundness error of $\frac{1}{2^k}$, where k is the length of the challenge.

- Honest Verifier Zero-Knowledge. Zero-knowledge follows from the zero-knowledge of the protocols for proofs $\pi_1, \pi_2, \pi_3, \pi_4, \pi_{i1}$ and π_{i2}.

- Soundness. We have two accepting transcripts for different challenges $c, \hat{c}$, $c \neq \hat{c}$. From the response to c, we have s_{i1}, s_{i2} for all i such that $s_{i1}, s_{i2} < p$. From the response to $\hat{c}$, and the soundness of the proofs $\pi_{i,1}$, we extract b'_i, s'_{i1}, s'_{i2} such that $z_{i1} = b'_i s'_{i1} + (1 - b'_i) s'_{i2} \mod p$, and $u_i = \mathsf{Com}_p(b'_i)$. Similarly from π_{i2}, we extract $\hat{b}_i, \hat{s}_{i1}, \hat{s}_{i2}$ such that $z_{i1} = \hat{b}_i \hat{s}_{i1} + (1 - \hat{b}_i) \hat{s}_{i2} \mod q$, and $v_i = \mathsf{Com}_q(\hat{b}_i)$. By the soundness of the proof π_3, π_4, we extract $b_i \in \{0, 1\}$. Since we have $s_{i1}, s_{i2} < p < q$, we have $s_{i1} = s'_{i1} = \hat{s}_{i1}, s_{i2} = s'_{i2} = \hat{s}_{i2}$. Therefore, $b_i = b'_i = \hat{b}_i$, and by soundness of π_1 and π_2, $x \equiv y \mod p$.

E Proof that a committed value is a polynomial of committed values

Linear relationship among committed values. Given $C_1 = \mathsf{Com}(x) = g^x h^{r_1}, C_2 = \mathsf{Com}(y) = g^y h^{r_2}, C_3 = \mathsf{Com}(z) = g^z h^{r_3}$, we want to prove that $z = ax + by$ for public a, b. The following protocol, lin proves $\mathsf{PK}\{(x, y, z) : C_1 = \mathsf{Com}(x) \wedge C_2 = \mathsf{Com}(y) \wedge C_3 = \mathsf{Com}(z) \wedge z = ax + by\}$

Given $\mathsf{C}_1 = \mathsf{Com}_p(x) = g^x h^r, \mathsf{C}_2 = \mathsf{Com}_q(y) = G^x H^R, p < q$, prove that $x \equiv y \mod p$

1. The prover commits to bits of x in both groups by computing $u_i = g^{b_i} h^{r_i}, v_i = G^{b_i} H^{R_i}$, for all $i \in [0, n]$ where $x = \sum_{i=0}^{n} 2^i b_i$, $n = \lceil \log p \rceil$. The prover sends $\{u_i, v_i\}$ to the verifier.

2. The prover proves that the bits combine to yield x and y by giving the following proofs π_1 and π_2. $\pi_1 : \mathsf{PK}\{(x, r, b_1, \ldots, b_n, r_1, \ldots, r_n) : \mathsf{C}_1 = \mathsf{Com}_p(x) \wedge u_1 = \mathsf{Com}_p(b_1) \wedge \cdots u_n = \mathsf{Com}_p(b_n) \wedge x = \sum 2^i b_i\}$, and $\pi_2 : \mathsf{PK}\{(y, R, b_1, \ldots, b_n, R_1, \ldots, R_n) : \mathsf{C}_2 = \mathsf{Com}_q(y) \wedge v_1 = \mathsf{Com}_q(b_1) \wedge \cdots v_n = \mathsf{Com}_q(b_n) \wedge y = \sum 2^i b_i\}$

3. For each bit b_i, the prover chooses random $s_{i1}, s_{i2} \in \mathbb{F}_p, s_{i1} \neq s_{i2}$, and computes commitments to them in both groups. The prover computes $a_{i1} = g^{s_{i1}} h^{\alpha_{i1}}, a_{i2} = g^{s_{i2}} g^{\alpha_{i2}}, a_{i3} = G^{s_{i1}} h^{\beta_{i1}}, a_{i4} = G^{s_{i2}} h^{\beta_{i2}}$, and sends $\{a_{i1}, a_{i2}, a_{i3}, a_{i4}\}$ to the verifier.

4. The prover proves that u_i and v_i are commitments to bits in both groups by proving $b_i(1 - b_i) = 0, \forall i$. The prover gives the following proofs. $\pi_3 : \mathsf{PK}\{(b_i, r_i) : u_i = \mathsf{Com}_p(b_i) \wedge b_i(1 - b_i) = 0\}$, and $\pi_4 : \mathsf{PK}\{(b_i, R_i) : v_i = \mathsf{Com}_q(b_i) \wedge b_i(1 - b_i) = 0\}$

5. The verifier chooses a random challenge bit c and sends it to the prover.

6. For challenge c,

   - If $c = 0$, set $z_{i1} = s_{i1}, z_{i2} = s_{i2}, z_{i3} = \alpha_{i1}, z_{i4} = \alpha_{i2}, z_{i5} = \beta_{i1}, z_{i6} = \beta_{i2}$, and send $(z_{i1}, z_{i2}, z_{i3}, z_{i4}, z_{i5}, z_{i6})$ for all i.
   - If $c = 1$, set $z_{i1} = b_i s_{i1} + (1 - b_i) s_{i2}$, and send $(z_{i1}, \pi_{i1}, \pi_{i2})$ for all i, where $\pi_{i1} = \mathsf{PK}\{(b_i, s_{i1}, s_{i2}, r_i, \alpha_{i1}, \alpha_{i2}) : z_{i1} = b_i s_{i1} + (1 - b_i) s_{i2} \wedge u_i = \mathsf{Com}_p(b_i) \wedge a_{i1} = \mathsf{Com}_p(s_{i1}) \wedge a_{i2} = \mathsf{Com}_p(s_{i2})\}$, and $\pi_{i2} = \mathsf{PK}\{(b_i, s_{i1}, s_{i2}, R_i, \beta_{i1}, \beta_{i2}) : z_{i1} = b_i s_{i1} + (1 - b_i) s_{i2} \wedge v_i = \mathsf{Com}_q(b_i) \wedge a_{i3} = \mathsf{Com}_q(s_{i1}) \wedge a_{i4} = \mathsf{Com}_q(s_{i2})\}$

7. Verification:

   - If $c = 0$, verify $z_{i1}, z_{i2} < p, z_{i1} \neq z_{i2}$, check that $a_{i1} = g^{z_{i1}} h^{z_{i3}}, a_{i2} = g^{z_{i2}} h^{z_{i4}}, a_{i3} = G^{z_{i1}} H^{z_{i5}}, a_{i4} = G^{z_{i2}} H^{z_{i6}}$, for all i, and verify proofs $\pi_1, \pi_2, \pi_3, \pi_4$.
   - If $c = 1$, verify proofs $\pi_1, \pi_2, \pi_3, \pi_4$, and proofs π_{i1}, π_{i2} for all i.

Figure 12: $\mathsf{Eq} : \mathsf{PK}\{(x, r, R) : \mathsf{Com}_p(x) = g^x h^r \wedge \mathsf{Com}_q(x) = G^x H^R\}$

- The verifier computes $C_4 = C_1^a C_2^b$.
- The prover gives the following proof $\mathsf{PK}\{(z, r_3, r_4) : C_3 = g^z h^{r_3} \wedge C_4 = g^z h^{r_4}\}$

Multiplicative relationship of committed values. We would like to prove that committed values satisfy a multiplicative relationship. Given $C_1 = \mathsf{Com}(x) = g^x h^{r_1}, C_2 = \mathsf{Com}(y) = g^y h^{r_2}, C_3 = \mathsf{Com}(z) = g^z h^{r_3}$, prove that $z = xy$. The following protocol, mul proves $\mathsf{PK}\{(x, y, z) : C_1 = \mathsf{Com}(x) \wedge C_2 = \mathsf{Com}(y) \wedge C_3 = \mathsf{Com}(z) \wedge z = xy\}$

- The prover computes and sends $C_4 = C_1^y = g^{z'} h^{r_4}$

- The prover gives the following proof: $\mathsf{PK}\{(y, r_2) : C_2 = g^y h^{r_2} \wedge C_4 = C_1^y\}$
- The prover gives the following proof: $\mathsf{PK}\{(z, r_3, r_4) : C_3 = g^z h^{r_3} \wedge C_4 = g^z h^{r_4}\}$

Polynomial relationship of committed values. Using standard techniques outlined above for proving linear relationships and product of committed values, we now sketch how to prove that a committed value is a polynomial P in variables that are committed to as well. Let $P(x_1, x_2, \ldots, x_n) = c_1 M_1 + \cdots c_t M_t$ be a degree d polynomial in n variables where each monomial can be written as $M_i = x_1^{d_1} x_2^{d_2} \cdots x_n^{d_n}, \sum_{i=1}^{n} d_i \leq d$.

- The prover commits to each monomial $C_{M_i} = \mathsf{Com}(M_i)$
- The prover commits to intermediate values, and proves that the monomial committed to is computed correctly.
  - For a degree d monomial M_i, it is written as product of two monomials M_{i1} and M_{i2} of degree $\lfloor d/2 \rfloor$ and $\lceil d/2 \rceil$ respectively. The prover commits to M_{i1} and M_{i2}. Given these commitments, the prover invokes the protocol mul to prove $M_i = M_{i1} M_{i2}$:
    $\mathsf{PK}\{(M_{i1}, M_{i2}) : C_{M_i1} = \mathsf{Com}(M_{i1}) \wedge C_{M_i2} = \mathsf{Com}(M_{i2}) \wedge M_i = M_{i1} M_{i2}\}$
  - Now each of these monomials M_{i1} and M_{i2} are proven to be correct using mul recursively until the proof is for a degree two term involving commitment to the input variables x_i.
- The above step is performed on every monomial M_i in P.
- The prover now proves linear relationship among committed monomials by invoking lin:
  $\mathsf{PK}\{(M_1, M_2, \ldots, M_t) : C_{M_1} = \mathsf{Com}(M_1) \wedge \cdots \wedge C_{M_t} = \mathsf{Com}(M_t) \wedge P = \sum c_i M_i\}$

F Efficiency

We briefly discuss the estimated cost of some of the building blocks. The ddlog proof is dominated by the cost of the range proofs in steps 4, 5, 6 of pointAddition protocol in Figure 1. In a recent work [BBB+17], it was shown how to prove that a committed value is in a range using only a number of field elements that is logarithmic in the bit length of the range. Using these proofs to instantiate all the necessary range proofs in protocol pointAddition, the prover's work is $30 \log t + 1800$ group exponentiations, the verifier's work is $10 \log t$ exponentiations, and the proof size is $2370 + \log \log t$ elements where the proof is for a curve defined over $\mathbb{F}_t$. The cost of comInSnark is the cost of the comEq in addition to the cost incurred by separating the wires in the underlying SNARK construction. The proof size of comInSnark is 15 group elements, and 2 field elements for every committed value (input/output). In the case of our following applications, the proof size is 17 elements. The prover's work is the number of exponentiations for computing the SNARK proof and an additional 2 exponentiations for the comEq proof. The verifier's work is 2 exponentiations and 21 pairings. Similarly, comIOSnark has proof size 26 elements, the prover's work, in addition to the exponentiations for the SNARK proof is 4 exponentiations and the verifier's work is 4 exponentiations and 30 pairings.

Proof of solvency. In Table 1, we compare the proof size and prover's work of Provisions with our protocol and a solution that uses zk-SNARK for the entire statement. The proof size and prover's work are dominated by the range proofs; the numbers below give only the dominating terms ignoring small constants and are assuming that the range proofs are realized using Bulletproofs.

zk technique	Functionality	Proof size (in elements)	Prover
Provisions	pay-to-pub	$10n + \log m + \log c$	$5n + 4mc$ exp.
SNARK	pay-to-pub, pay-to-hash	7	$(\lvert H\rvert + p^3)n + c$ exp.
Our composition techniques	pay-to-pub, pay-to-hash	$2396n + \log p + \log n$	$(\lvert H\rvert + 30p + 1800)n + c$ exp.

Table 1: Comparison of prover work and proof sizes for proof of solvency using different methods. n is the size of the anonymity set, c is the number of customer accounts, m is $\lceil \log \mathsf{Max} \rceil = 51$, p is the bit length of the modulus for exponentiation (size of the field over which the the curve is defined). For $n = 500,000$ and $c = 2$ million, the proof size and prover's work in Provisions is $5 * 10^6$ and $4 * 10^7$ respectively. For the same parameters, our approach gives proof size of 10^9 and prover's work 10^{10}, while also achieving the additional pay-to-hash functionality. A fully zk-SNARK solution requires prover's work roughly 10^{13}. (Exp. stands for exponentiations.)

zk technique	Feature	Proof size	Prover
Cinderella	non-interactive	7	$\lvert H\rvert +$ additional 164,826 equations for RSA (as optimized in Cinderella)
GC + Sigma [CGM16]	interactive	$\lvert H\rvert$	$\lvert m\rvert + \lvert h\rvert$ exp. + $\lvert H\rvert$ symmetric-key operations
Our composition techniques	non-interactive	$42 + \log p$	$\lvert H\rvert + \log p + 16$ exp.

Table 2: Comparison of prover work and proof sizes for credential verification using different methods. p is the order of the group in which commitments are computed, $|m|$ is the bit length of the message. For $e = 65537, \log p = 256, |H| = 23785$, we note an 87% decrease in prover's work compared to Cinderella at the cost of increasing the proof size to 298 from 7 group elements. (Exp. stands for exponentiations.)

Privacy preserving credentials. In Table 2, we compare the proof size and prover's work in privacy-preserving credentials for Cinderella, the interactive protocol of [CGM16], and our composition.

G Assumptions on Bilinear maps

Assumption G.1 (q-PDH)**.** *The q-power Diffie-Hellman (q-PDH) assumption holds for* $\mathsf{GroupGen}$ *if for all non-uniform probabilistic polynomial time algorithm $\mathcal{A}$, the following probability is negligible in the security parameter.*

$$\Pr\left(g_1^{s^{q+1}} \leftarrow \mathcal{A}(\sigma)\ :\ \begin{array}{c} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\kappa), \\ \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \\ s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, g_1^{s^2}, g_2^{s^2}, \ldots, g_1^{s^q}, g_2^{s^q}, g_1^{s^{q+2}}, g_2^{s^{q+2}} \ldots, g_1^{s^{2q}}, g_2^{s^{2q}}) \end{array} \right).$$

Assumption G.2 (q-PKE)**.** *The q power-knowledge of exponent (q-PKE) assumption holds for* $\mathsf{GroupGen}$ *if for all non-uniform probabilistic polynomial time algorithm $\mathcal{A}$, there exists a non-uniform probabilistic polynomial time extractor $\chi_\mathcal{A}$ such that the following probability is negligible in the security parameter.*

$$\Pr\left(e(c, g_2^{\alpha}) = e(g_1, \hat{c}) \wedge c \neq \prod_{i=0}^{q} g_1^{a_i s^i} \;:\; \begin{array}{c} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\kappa), \\ \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e), \\ \alpha, s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, \ldots, g_1^{s^q}, g_2^{s^q}, g_1^{\alpha s}, g_2^{\alpha s}, \ldots, g_1^{\alpha s^q}, g_2^{\alpha s^q}), \\ (c, \hat{c}; a_0, \ldots, a_q) \leftarrow (\mathcal{A}||\chi_{\mathcal{A}})(\sigma, z) \end{array} \right).$$

In the above, z is auxiliary information generated independently of α, and $(x; y) \leftarrow (\mathcal{A}||\chi_{\mathcal{A}})(\sigma, z)$ denotes that on input σ, $\mathcal{A}$ outputs x, and $\chi_{\mathcal{A}}$ given the same input σ, and $\mathcal{A}$'s random tape, outputs y.

Assumption G.3 (q-SDH)**.** *The q-strong Diffie-Hellman (q-SDH) assumption holds for* GroupGen *if for all non-uniform probabilistic polynomial time algorithm $\mathcal{A}$, the following probability is negligible in the security parameter.*

$$\Pr\left(y = e(g_1, g_2)^{\frac{1}{s+c}}, c \in \mathbb{Z}_p^* \;:\; \begin{array}{c} (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\kappa), \\ \sigma_1 = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e), \\ s \xleftarrow{R} \mathbb{Z}_p^*, \\ \sigma = (\sigma_1, g_1, g_2, g_1^s, g_2^s, g_1^{s^2}, g_2^{s^2}, \ldots, g_1^{s^q}, g_2^{s^q}), \\ y \leftarrow \mathcal{A}(\sigma) \end{array} \right).$$

H SNARK Construction of Parno et al.

We review the construction of SNARK from QAP of Parno et al. [PHGR13] below. Let f be a function that maps N elements from $\mathbb{F}$ to 0 or 1. Convert f into an arithmetic circuit C and build a QAP $Q = (V, W, Y, t(x))$ for C of size m and degree d. We let the indices $i \in [1, n]$ denote the public input (the statement y) and $i \in [n+1, N]$ denote the private input (the witness x).

1. **CRS generation**. Choose $r_v, r_w, \alpha_v, \alpha_w, \alpha_y, s, \beta, \gamma \xleftarrow{R} \mathbb{F}$. Set $r_y = r_v r_w, g_v = g^{r_v}, g_w = g^{r_w}$, and $g_y = g^{r_y}$. Set the CRS to be:

$$\mathsf{crs} = \Big(\{g_v^{v_k(s)}\}_{k \in [n+1,m]}, \{g_w^{w_k(s)}\}_{k \in [n+1,m]}, \{g_y^{y_k(s)}\}_{k \in [n+1,m]}, \\ \{g_v^{\alpha_v v_k(s)}\}_{k \in [n+1,m]}, \{g_w^{\alpha_w w_k(s)}\}_{k \in [n+1,m]}, \{g_y^{\alpha_y y_k(s)}\}_{k \in [n+1,m]}, \\ \{g^{s^i}\}_{i \in [d]}, \{g_v^{\beta v_k(s)} g_w^{\beta w_k(s)} g_y^{\beta y_k(s)}\}_{k \in [n+1,m]} \Big).$$

   Set the short verification CRS to be:

$$\mathsf{shortcrs} = \Big(g, g^{\alpha_v}, g^{\alpha_w}, g^{\alpha_y}, g^{\gamma}, g^{\beta\gamma}, g_y^{t(s)}, \\ \{g_v^{v_k(s)}\}_{k \in \{0\} \cup [n]}, \{g_w^{w_k(s)}\}_{k \in \{0\} \cup [n]}, \{g_y^{y_k(s)}\}_{k \in \{0\} \cup [n]} \Big).$$

2. **Prove**. On input statement y, witness x, and crs, the prover evaluates the QAP to obtain $\{a_i\}_{i \in [m]}$. (Equivalently, evaluates C to obtain the values on the circuit wires). The prover solves for the quotient polynomial h such that $p(x) = h(x)t(x)$. Let $v_{mid}(x) = \sum_{k \in [n+1,m]} a_k v_k(x)$,

and similarly define $w_{mid}(x)$ and $y_{mid}(x)$. The prover computes the proof π:

$$\Big(g_v^{v_{mid}(s)}, g_w^{w_{mid}(s)}, g_y^{y_{mid}(s)}, g^{h(s)},$$
$$g_v^{\alpha_v v_{mid}(s)}, g_w^{\alpha_w w_{mid}(s)}, g_y^{\alpha_y y_{mid}(s)},$$
$$g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)}\Big)$$

3. **Verify**. On input $\mathsf{shortcrs}$, y, and a proof $\pi = (g_v^{V_{mid}}, g_w^{W_{mid}}, g_y^{Y_{mid}}, g^H, g_v^{V'_{mid}}, g_w^{W'_{mid}}, g_y^{Y'_{mid}}, g^Z)$:
   - The verifier can compute a term representing the public input y, by representing them as coefficients $a_1, \ldots, a_n \in \mathbb{F}$, and computing

     $$g_v^{v_{in}(s)} = \prod_{k \in [n]} \left(g_v^{v_k(s)}\right)^{a_k}$$

     Similarly, compute $g_w^{w_{in}(s)}$ and $g_y^{y_{in}(s)}$. Check whether,

     $$e(g_v^{v_0(s)} g_v^{v_{in}(s)} g_v^{V_{mid}}, g_w^{w_0(s)} g_w^{w_{in}(s)} g_w^{W_{mid}})$$
     $$= e(g_y^{t(s)}, g^H) \cdot e(g_y^{y_0(s)} g_y^{y_{in}(s)} g_y^{Y_{mid}}, g).$$

   - Verify that $e(g_v^{V'_{mid}}, g) = e(g_v^{V_{mid}}, g^{\alpha_v})$, $e(g_w^{W'_{mid}}, g) = e(g_w^{W_{mid}}, g^{\alpha_w})$, and $e(g_y^{Y'_{mid}}, g) = e(g_y^{Y_{mid}}, g^{\alpha_y})$.
   - Verify $e(g^Z, g^\gamma) = e(g_v^{V_{mid}} g_w^{W_{mid}} g_y^{Y_{mid}}, g^{\beta\gamma})$.

   Output 1 if all the verifications succeed, else output 0.