

# Non-Interactive Zero-Knowledge Proofs for Composite Statements

Shashank Agrawal<sup>1</sup>, Chaya Ganesh<sup>2\*</sup>, and Payman Mohassel<sup>1</sup>

<sup>1</sup> Visa Research

<sup>2</sup> Aarhus University

**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.

## 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 [41, 36, 7, 61]. Zero-knowledge proofs (ZKPs) have found numerous applications as a building block in other cryptographic constructions such as identification schemes [32], group signature schemes [19], public-key encryption [55], anonymous credentials [17], voting [23], and secure multi-party computation [42]. 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 [8, 56].

Zero-knowledge proofs exist for all languages in NP [41], 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

---

\* Work done as an intern at Visa Research.

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 [48] has led to (Boolean) circuit-friendly NIZKs [40, 20, 6], 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 [46, 60, 26, 59, 38, 45]. Moreover, they can be made non-interactive, i.e. only a single message from prover to verifier, using the efficient Fiat-Shamir transformation [34].

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) [44, 51, 12, 39, 57, 9, 52, 10]. Starting with the construction of Kilian [50] based on probabilistically checkable proofs (PCPs), made non-interactive by Micali [53], there has been further works [43, 11, 29] 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) [39] 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 [4]. 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 [62, 27]. 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 [2], where the public portion is derived from the private key by doing an exponentiation operation on the secp256k1 curve [1]<sup>3</sup>. 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

---

<sup>3</sup> 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.

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 get around this problem by assuming (incorrectly) that public keys themselves are available on the blockchain so that Sigma protocols alone suffice [27]. 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 [22] 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  $\sigma$ , where  $\sigma$  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. [30] 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. [21] 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 [30].

*zk-SNARKs with composable CRSs.* Anonymous decentralized digital cryptocurrencies 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  $H(x_1 || H(x_2) || \dots || H(x_n)) = y$  for a large value of  $n$ . The CRS generated for proving this statement is extremely large (about a gigabyte for ZCash [3]) and cannot be reused to prove any other statement. A better alternative is to gener-

ate 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  $C_1, \dots, C_n$  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 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.

## 2 Preliminaries

*Notation.* Throughout the paper, we use  $\kappa$  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  $\text{negl}$  to denote a negligible function. We write  $\mathcal{X}_\kappa \equiv \mathcal{Y}_\kappa$  to mean that distributions  $\mathcal{X}_\kappa$  and  $\mathcal{Y}_\kappa$  are identical. We use  $[1, n]$  to represent the set of numbers  $\{1, 2, \dots, n\}$ . If  $\text{Alg}$  is a randomized algorithm, we use  $y \leftarrow \text{Alg}(x)$  to denote that  $y$  is the output of  $\text{Alg}$  on  $x$ . We write  $x \xleftarrow{R} \mathcal{X}$  to mean sampling a value  $x$  uniformly from the set  $\mathcal{X}$ .

We denote an interactive protocol between two parties  $A$  and  $B$  by  $\langle A, B \rangle$ .  $\langle A(x), B(y) \rangle(z)$  denotes a protocol where  $A$  has input  $x$ ,  $B$  has input  $y$  and  $z$  is a common input. Also,  $\text{view}_A$  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  $B$  ( $\text{view}_B$  is defined in a similar manner).

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

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  $\mathbb{G} \neq \tilde{\mathbb{G}}$ , and discrete logarithm is hard in  $\mathbb{G}$ . We also rely on  $q$ -type assumptions similar to Parno et al. [57] (but in asymmetric groups).

*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 [13, 33]).** *A NIZK argument for an NP relation  $R$  consists of a triple of polynomial time algorithms (Setup, Prove, Verify) defined as follows.*

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

*The algorithms above should satisfy the following properties.*

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

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

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

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

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

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

**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  $\text{Ext}$  such that the following probability is negligible in  $\kappa$ :

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

**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  $\pi$  is given by  $|\pi| = \text{poly}(\kappa) \cdot \text{polylog}(|s| + |w|)$ .

## 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  $\kappa$ -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 [28]).** *An interactive protocol between a prover  $P$  and a verifier  $V$  is a  $\Sigma$  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  $\Sigma$  protocol can be efficiently compiled into a non-interactive zero-knowledge proof of knowledge (in the random oracle model) through the Fiat-Shamir transform [34]. 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  $\Sigma$ -protocols.* In Cramer et al. [26], 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 [26]).** *If  $\Pi_0$  is a  $\Sigma$ -protocol for  $R_0$  and  $\Pi_1$  a  $\Sigma$ -protocol for  $R_1$ , then there is a  $\Sigma$ -protocol  $\Pi_{\text{OR}}$  for the relation  $R_{\text{OR}}$  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 [58] 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  $\text{Com}_q(x)$  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 [60, 37, 19, 18]. For example, one could show that  $y = ax + b$  for some public values  $a$  and  $b$ , given  $\text{Com}_q(x)$  and  $\text{Com}_q(y)$ .

We use  $\text{PK}\{(x, y, \dots) : \text{statements about } x, y, \dots\}$  to denote a proof of knowledge of  $x, y, \dots$  that satisfies *statements* [19]. Other values in *statements* are public.

### 2.3 SNARK Construction from QAP

The work of Gennaro et al. [39] 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. [57] 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 [39]).** *A quadratic arithmetic program (QAP)  $Q$  over a field  $\mathbb{F}$  consists of three sets of polynomials  $V = \{v_k(x) : k \in \{0, \dots, m\}\}$ ,  $W = \{w_k(x) : k \in \{0, \dots, m\}\}$ ,  $Y = \{y_k(x) : k \in \{0, \dots, m\}\}$  and a target polynomial  $t(x)$ , all in  $\mathbb{F}[X]$ .*

*Let  $f : \mathbb{F}^n \rightarrow \mathbb{F}^n$  be a function with input variables labeled  $1, \dots, n$  and output variables labeled  $m - n' + 1, \dots, m$ . A QAP  $Q$  is said to compute  $f$  if the following holds:  $a_1, \dots, a_n, a_{m-n'+1}, \dots, a_m \in \mathbb{F}^{n+n'}$  is a valid assignment to the input and output variables of  $f$  (i.e.,  $f(a_1, \dots, a_n) = (a_{m-n'+1}, \dots, a_m)$ ) iff there exist  $(a_{n+1}, \dots, a_{m-n'}) \in \mathbb{F}^{m-n-n'}$  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.

## 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  $\text{Com}(x_i)$  to inputs

and a commitment  $\text{Com}(\prod g_i^{P_i(x_i)})$  to the output of an algebraic function where  $g_i$ s are public generators in an elliptic curve group and  $P_i$ s 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 [19, 54]. 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

$$y^2 = x^3 + ax + b, \quad (1)$$

where  $a, b \in \mathbb{F}_t$ , but the techniques we describe below would extend to other curve families like Edwards [31]. 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

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

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

We use `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 `doubleFormula(P)`. In this case,  $(x_3, y_3)$  is given by

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

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

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

$$x_3x_2^2 + x_3x_1^2 + x_1^3 + x_2^3 + 2y_1y_2 = y_2^2 + y_1^2 + x_1^2x_2 + x_1x_2^2 + 2x_1x_2x_3, \quad (6)$$

$$x_2y_3 + x_3y_2 + x_2y_1 = x_1y_2 + x_3y_1 + x_1y_3. \quad (7)$$

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  $C_1 = \text{Com}(a) = g^ah^{r_1}$ ,  $C_2 = \text{Com}(b) = g^bh^{r_2}$ ,  $C_3 = \text{Com}(c) = g^ch^{r_3}$ , 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:  $\text{PK}\{(a, b, c, b', c', r_1, r_2, r_3, r_4) : C_1 = g^ah^{r_1} \wedge C_2 = g^bh^{r_2} \wedge C_3 = g^ch^{r_3} \wedge C_4 = C_1^{b'} \wedge C_4 = g^{c'}h^{r_4} \wedge b' = b \wedge c' = c\}$ . In general, Sigma protocols for polynomial relationships among committed values were given by Camenisch and Michels [18].

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 [14, 16], that is, to prove that  $x \in [0, S]$  for a public  $S$  and committed  $x$ , including the recent, very efficient technique called Bulletproof [15].)

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 the full version. 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  $C_P = \text{Com}_q(P) = (\text{Com}_q(P_x), \text{Com}_q(P_y))$  denote a commitment to a point  $P = (P_x, P_y)$ .

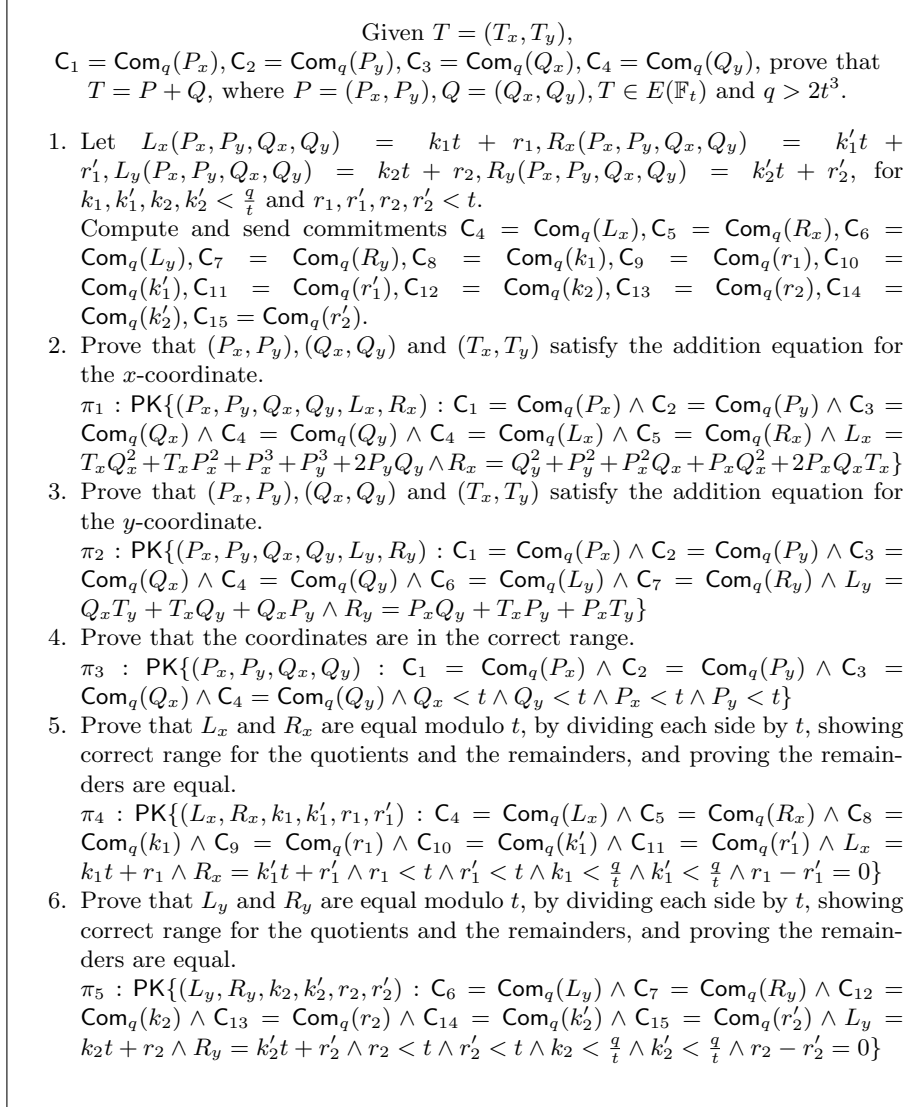


Fig. 1: addition :  $\text{PK}\{(P = (P_x, P_y), Q = (Q_x, Q_y)) : T = (T_x, T_y) = \text{addFormula}(P, Q) \wedge C_1 = \text{Com}_q(P_x) \wedge C_2 = \text{Com}_q(P_y) \wedge C_3 = \text{Com}_q(Q_x) \wedge C_4 = \text{Com}_q(Q_y)\}$

**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  $\Sigma$ -protocol for the relation  $R = \{((T, C_P, C_Q), (P, Q)) : C_P = \text{Com}_q(P) \wedge C_Q = \text{Com}_q(Q) \wedge T = \text{addFormula}(P, Q) \wedge P, Q \in E\}$ .*

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

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

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`.

$$\text{pointAddition} : \text{PK}\{(P, Q) : C_P = \text{Com}_q(P) \wedge C_Q = \text{Com}_q(Q) \wedge P, Q \in E \wedge \\ ((P \neq Q \wedge P \neq -Q \wedge T = \text{addFormula}(P, Q)) \vee \\ (P = Q \wedge T = \text{doubleFormula}(P)) \vee (P = -Q \wedge T = 0))\}$$

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  $\Sigma$ -protocol for the relation  $R = \{((T, C_P, C_Q), (P, Q)) : C_P = \text{Com}_q(P) \wedge C_Q = \text{Com}_q(Q) \wedge T = P + Q \wedge P, Q \in E\}$ .*

We note that the protocol `addition` may be modified to prove point addition for a committed point  $T$  in the following way. The proofs  $\pi_1$  and  $\pi_2$  are on committed coordinates  $(T_x, T_y)$ , and the range proof  $\pi_3$  also includes proving the range of coordinates of  $T$ . We denote the point addition proof  $\text{PK}\{(P, Q, T) : C_P = \text{Com}_q(P) \wedge C_Q = \text{Com}_q(Q) \wedge C_T = \text{Com}_q(T) \wedge T = P + Q \wedge P, Q, T \in E\}$  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 the full version 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 [15]. 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  $\Sigma$ -protocol for the relation  $R = \{(P, C, C_h, (\lambda, h)) : C = \text{Com}(\lambda) \wedge C_h = \text{Com}(h) \wedge h = \lambda P, 0 < \lambda < p\}$  with soundness  $1/2$ .*

Given  $C_1 = \text{Com}_p(\lambda), C_2 = \text{Com}_q(x), C_3 = \text{Com}_q(y)$ , 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:  $a_1 = \text{Com}_p(\alpha) = \alpha P + \beta_1 Q, a_2 = \text{Com}_q(\gamma_1) = \gamma_1 P' + \beta_2 Q', a_3 = \text{Com}_q(\gamma_2) = \gamma_2 P' + \beta_3 Q'$  where  $\alpha \in \mathbb{F}_p$  is chosen at random, and  $(\gamma_1, \gamma_2) = \alpha P$ . and 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 `pointAddition` (Figure 1) to prove that  $T = (\gamma_1, \gamma_2) - (x, y)$ .  
 $\pi : \text{PK}\{(x, y, \gamma_1, \gamma_2) : T = (\gamma_1, \gamma_2) - (x, y)\}$ . Send  $(z_1, \pi)$
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  $\pi$ .

Fig. 2:  $\text{ddlog} : \text{PK}\{(\lambda, x, y, r, r_1, r_2) : \text{Com}_p(\lambda) = \lambda P + r Q \wedge \text{Com}_q(x) = x P' + r_1 Q' \wedge \text{Com}_q(y) = y P' + r_2 Q' \wedge (x, y) = \lambda P\}$

### 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  $f : \mathbb{Z}_q^{n+m} \rightarrow \{0, 1\}$ , and let  $C$  be a commitment to the input  $x$ . Let  $f_{\text{com}}$  be the relation,  $f_{\text{com}} = \{(y, (x, b)) : ((x, y) \in \mathcal{L}_f \wedge b = 1) \vee (b = 0)\}$ . The Sigma protocol for the relation  $f_{\text{com}}$  is given by the proof  $\text{PK}\{(b, x) : f(x, y) = b \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\}$ . 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  $\Pi$  be a  $\Sigma$ -protocol for the relation  $f$ . The  $\Sigma$ -protocol for  $f_{\text{com}}$  is shown in Figure 3.

**Theorem 3.4.** *If  $\Pi$  is a  $\Sigma$ -protocol for  $f$ , then `comBitSigma` is a  $\Sigma$ -protocol for  $f_{\text{com}}$ .*

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

*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

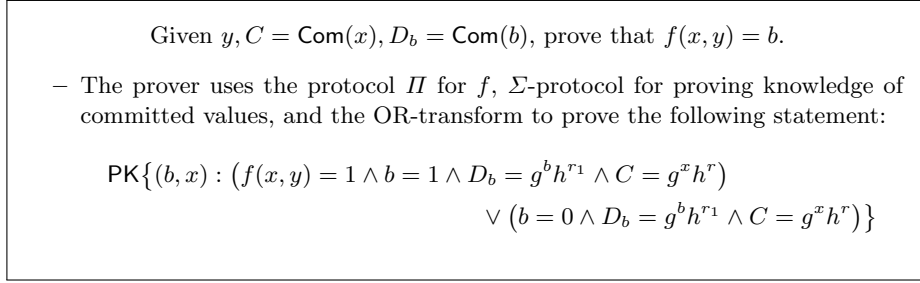


Fig. 3: **comBitSigma** :  $\text{PK}\{(b, x) : f(x, y) = b \wedge D_b = g^b h^{r_1} \wedge C = g^x h^r\}$

protocol to prove knowledge of discrete logarithms of two committed points such that their sum is another committed point which we do using **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** :  $\text{PK}\{(\gamma, \alpha, \beta, x_1, x_2) : \gamma = \alpha + \beta \wedge \alpha = x_1 P \wedge \beta = x_2 P\}$  for  $0 < x_1, x_2 < p$  is shown in Figure 4.

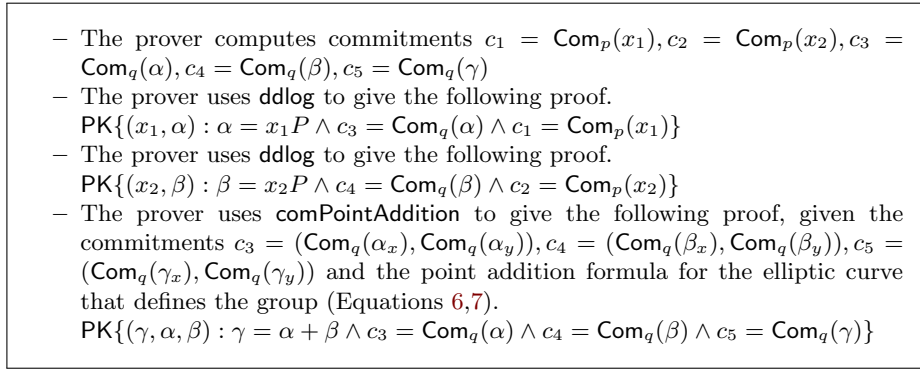


Fig. 4: **comSum** :  $\text{PK}\{(\gamma, \alpha, \beta, x_1, x_2) : \gamma = \alpha + \beta \wedge \alpha = x_1 P \wedge \beta = x_2 P\}$

*When Committed Output is a Group Element.* In the following discussion, similar to before, for a group element  $\alpha = (\alpha_x, \alpha_y)$ , where  $\alpha_x, \alpha_y$  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.  $\text{Com}(\alpha) = (\text{Com}(\alpha_x), \text{Com}(\alpha_y))$ .

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  $f(x_1, \dots, x_n) = \Pi g_i^{P_i(x_i)}$ . We sketch the ideas at a high-level for some simple functions. Let  $f : \mathbb{Z}_p^n \rightarrow \mathcal{G}$ , 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  $\text{PK}\{(x, y) : g^{x^2} = y \wedge C_1 =$

$\text{Com}(x) \wedge C_2 = \text{Com}(y)\}$ , the prover computes the commitments  $C_1 = \text{Com}_p(x)$ ,  $C_2 = \text{Com}_p(x^2)$  and  $C_3 = \text{Com}_q(k) = (\text{Com}_q(k_x), \text{Com}_q(k_y))$ , 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.  $\text{PK}\{(x_2, k) : k = g^{x_2^2} \wedge C_2 = \text{Com}_p(x_2) \wedge C_3 = \text{Com}_q(k)\}$  using  $\text{ddlog}$ , and a Sigma protocol for  $\text{PK}\{(x_1, x_2) : x_2 = x_1^2 \wedge C_1 = \text{Com}_p(x_1) \wedge C_2 = \text{Com}_p(x_2)\}$ . Given the above building blocks, it is easy to see that we can extend the techniques to devise proofs  $\text{comSigma}$  for  $f(x_1, \dots, x_n) = \prod g_i^{P_i(x_i)}$ .

## 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 [24, 35], it has been in different contexts and for different purposes. We briefly discuss how our ideas relate to two such ideas.

In [24], 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 [35], 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 [35] 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 [57] 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  $(a_1, \dots, a_n)$  in  $y = \prod_{i=1}^n G_i^{a_i}$  and individual algebraic commitments to them. Using the standard notation, we denote the protocol by  $\text{PK}\{(a_1, \dots, a_n, r_1, \dots, r_n) : y = \prod_{i=1}^n G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \dots \wedge C_n = g^{a_n} h^{r_n}\}$ . We include the steps of the protocol in the full version.

#### 4.1 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  $f : \mathbb{F}^N \rightarrow \mathbb{F}^{n'}$  be a function with input/output values from  $\mathbb{F}$ , computed by an arithmetic circuit  $C$  with input wires labeled  $1, \dots, N$ , output wires labeled  $m - n' + 1, \dots, m$ . Let  $\mathcal{Q}$  be a QAP 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  $I_{com} \subseteq \{1, \dots, N\}$  be the set of indices corresponding to the private inputs  $a_1, \dots, a_n$ ,  $I_{pub}$  the indices for the public input wires, and  $I_{out}$  the indices for the public output. Then let  $I_{mid} = \{1, \dots, m\} \setminus (I_{pub} \cup I_{com} \cup I_{out})$  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  $i$ th input  $a_i$ . The construction  $\text{comInSnark} : \text{PK}\{(a_1, \dots, a_n, r_1, \dots, r_n) : f(a_1, \dots, a_n, z_1, \dots, z_{N-n}) = (b_1, \dots, b_{n'}) \wedge C_1 = g^{a_1} h^{r_1} \wedge \dots \wedge C_n = g^{a_n} h^{r_n}\}$  is given in Figure 5.

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

1. CRS generation: Run  $\text{GroupGen}(1^\kappa)$  to get  $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, g, \tilde{g}, e)$ . 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:

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

Set the short verification CRS to be:

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

2. Prove: On input  $z_1, \dots, z_{N-n}$ , witness  $a_1, \dots, a_n$ , and  $\text{crs}$ , 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  $\pi$ :

$$\begin{aligned} \left( g_v^{v_{com}(s)}, g_v^{v_{mid}(s)}, \tilde{g}_w^{w_{com}(s)}, \tilde{g}_w^{w_{mid}(s)}, g_y^{y_{com}(s)}, g_y^{y_{mid}(s)}, g^{h(s)}, \right. \\ \tilde{g}_v^{\alpha_v v_{com}(s)}, \tilde{g}_v^{\alpha_v v_{mid}(s)}, g_w^{\alpha_w w_{com}(s)}, g_w^{\alpha_w w_{mid}(s)}, \tilde{g}_y^{\alpha_y y_{com}(s)}, \tilde{g}_y^{\alpha_y y_{mid}(s)}, \\ \left. g_v^{\beta v_{com}(s)} g_w^{\beta w_{com}(s)} g_y^{\beta y_{com}(s)}, g_v^{\beta v_{mid}(s)} g_w^{\beta w_{mid}(s)} g_y^{\beta y_{mid}(s)} \right) \end{aligned}$$

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

3. Verify:

– On input  $\text{shortcrs}$ ,  $z$ , and proofs  $\pi$ ,  $\pi_{in}$  parse  $\pi$  as

$$\begin{aligned} \pi = & \left( g^{V_{com}}, g^{V_{mid}}, \tilde{g}^{W_{com}}, \tilde{g}^{W_{mid}}, g^{Y_{com}}, g^{Y_{mid}}, \tilde{g}^H, \right. \\ & \left. \tilde{g}^{V'_{com}}, \tilde{g}^{V'_{mid}}, g^{W'_{com}}, g^{W'_{mid}}, \tilde{g}^{Y'_{com}}, \tilde{g}^{Y'_{mid}}, g^{Z_{com}}, g^{Z_{mid}} \right) \end{aligned}$$

- 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

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

- Verify that the linear combinations are in correct spans.
 
$$e(g^{V_{com}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{com}}), \quad e(g^{V_{mid}}, \tilde{g}^{\alpha_v}) = e(g, \tilde{g}^{V'_{mid}}),$$

$$e(g^{W'_{com}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{com}}), \quad e(g^{W'_{mid}}, \tilde{g}) = e(g^{\alpha_w}, \tilde{g}^{W_{mid}}),$$

$$e(g^{Y_{com}}, \tilde{g}^{\alpha_y}) = e(g, \tilde{g}^{Y'_{com}}), \quad e(g^{Y_{mid}}, \tilde{g}^{\alpha_y}) = e(g, \tilde{g}^{Y'_{mid}}).$$
- Verify same coefficients in all linear combinations.
  - (a)  $e(g^{Z_{com}}, \tilde{g}^\gamma) = e(g^{V_{com}} g^{Y_{com}}, \tilde{g}^{\beta\gamma}) \cdot e(g^{\beta\gamma}, \tilde{g}^{W_{com}})$
  - (b)  $e(g^{Z_{mid}}, \tilde{g}^\gamma) = e(g^{V_{mid}} g^{Y_{mid}}, \tilde{g}^{\beta\gamma}) \cdot e(g^{\beta\gamma}, \tilde{g}^{W_{mid}})$
- 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  $\pi_{in}$ .

Fig. 5:  $\text{comInSnark} : \text{PK}\{(a_1, \dots, a_n, r_1, \dots, r_n) : f(a_1, \dots, a_n, z_1, \dots, z_{N-n}) = (b_1, \dots, b_n) \wedge C_1 = g^{a_1} h^{r_1} \wedge \dots \wedge C_n = g^{a_n} h^{r_n}\}$

*Zero-knowledge.* We make our construction zero-knowledge, and obtain  $\text{zkcomInSnark}$ , by randomizing the elements in the proof  $\pi$  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  $\text{crs}$ :  $g_v^{t(s)}$ ,  $\tilde{g}_w^{t(s)}$ ,  $g_y^{t(s)}$ ,  $g_v^{\alpha_v t(s)}$ ,  $g_w^{\alpha_w t(s)}$ ,  $g_y^{\alpha_y t(s)}$ ,  $g_v^{\beta t(s)}$ ,  $g_w^{\beta t(s)}$ ,  $g_y^{\beta t(s)}$  ( $g_v^{t(s)}$  is also added to  $\text{shortcrs}$ ). Prover can now compute the new values in  $\pi$  from  $\text{crs}$ , and they are verified in the same manner as before. The proof  $\pi_{in}$  now proves a slightly different statement:  $\text{PK}\{(a_1, \dots, a_n, \delta, r_1, \dots, r_n) : y = H^\delta \prod_{i=1}^n G_i^{a_i} \wedge C_1 = g^{a_1} h^{r_1} \wedge \dots \wedge C_n = g^{a_n} h^{r_n}\}$ . To verify it, the verifier uses  $g_v^{t(s)}$  from  $\text{shortcrs}$ .

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

A proof of Theorem 4.1 can be found in the full version. Similarly, by separating the circuit wires into private input, public input, intermediate values and private output, we obtain  $\text{zk-SNARK}$  on committed input and output. This construction,  $\text{comIOSnark}$ , can be found in the full version. 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 the full version, 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. 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  $\text{PK}\{(x_1, \dots, x_n, y_1, \dots, y_m) : f_1(x_1, \dots, x_n, f_2(y_1, \dots, y_m)) = z\}$  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  $f_1 : \mathbb{F}_p^2 \rightarrow \mathbb{F}_p$ , and  $f_2 : \mathbb{F}_p \rightarrow \mathbb{F}_p$ , 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  $c_1 = \text{Com}_p(x_1), c_2 = \text{Com}_p(x_2), c_3 = \text{Com}_p(x_3)$ . 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  $c_3$ .  $\text{PK}\{(x_2, x_3, r_2, r_3) : f_2(x_2) = x_3 \wedge c_2 = \text{Com}_p(x_2) \wedge c_3 = \text{Com}_p(x_3)\}$ .
- The prover uses zk-comInSnark to give a proof that  $f_1(x_1, x_3) = z$  given  $c_1, c_3$  and  $z$ .  $\text{PK}\{(x_1, x_3, r_1, r_3) : f_1(x_1, x_3) = z \wedge c_1 = \text{Com}_p(x_1) \wedge c_3 = \text{Com}_p(x_3)\}$ .

2.  $f_1$  is an arithmetic circuit and  $f_2$  is algebraic. Let  $f_1 : \mathbb{F}_p^3 \rightarrow \mathbb{F}_p, f_2 : \mathbb{Z}_q \rightarrow \mathcal{G}$  and  $T : \mathcal{G} \rightarrow \mathbb{F}_p^2$ . 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  $\text{PK}\{(x_1, x_2) : f_1(x_1, T(f_2(x_2))) = z\}$ . 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  $c_1 = \text{Com}_p(x_1), c_2 = \text{Com}_q(x_2), c_3 = \text{Com}_p(k) = (\text{Com}_p(k_x), \text{Com}_p(k_y))$ , 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:  $\text{PK}\{(x_2, k, r_2, r_3) : f_2(x_2) = k \wedge c_2 = \text{Com}_q(x_2) \wedge c_3 = \text{Com}_p(k)\}$ .
- The prover uses zk-comInSnark to prove  $f_1(x_1, T(k)) = z$  given  $c_1, c_3, c_4$ .  $\text{PK}\{(x_1, k, r_1, r_3) : f_1(x_1, T(k)) = z \wedge c_1 = \text{Com}_p(x_1) \wedge c_3 = \text{Com}_p(k)\}$ .

3.  $f_1$  is algebraic, and  $f_2$  is an arithmetic circuit. Let  $f_1 : \mathbb{Z}_q^2 \rightarrow \mathcal{G}, f_2 : \mathbb{F}_p \rightarrow \mathbb{F}_p$ . Let  $\Pi$  be a  $\Sigma$ -protocol for  $f_1$ . The following is a protocol for  $\text{PK}\{(x_1, x_2) : f_1(x_1, f_2(x_2)) = z\}$ . 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  $c_1 = \text{Com}_q(x_1), c_2 = \text{Com}_p(x_2), c_3 = \text{Com}_q(x_3), c'_3 = \text{Com}_p(x_3)$ .  $c_3$  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  $c'_3$ .  $\text{PK}\{(x_2, x'_3, r_2, r'_3) : f_2(x_2) = x'_3 \wedge c_2 = \text{Com}_p(x_2) \wedge c'_3 = \text{Com}_p(x'_3)\}$ .
- The prover uses the sigma protocol  $\Pi$  to give the following proof.  $\text{PK}\{(x_1, x_3, r_1, r_3) : f_1(x_1, x_3) = z \wedge c_1 = \text{Com}_q(x_1) \wedge c_3 = \text{Com}_q(x_3)\}$ .
- The prover uses the protocol `Eq` to prove that  $c'_3$  and  $c_3$  are commitments to the same value.  $\text{PK}\{(x_3, x'_3, r_3, r'_3) : x_3 \equiv x'_3 \pmod{q} \wedge c_3 = \text{Com}_q(x_3) \wedge c'_3 = \text{Com}_p(x'_3)\}$

4.  $f_1$  and  $f_2$  are algebraic. Let  $f_1 : \mathbb{Z}_p^3 \rightarrow \mathcal{G}_1, f_2 : \mathbb{Z}_q \rightarrow \mathcal{G}_2$ , 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  $\Pi_1$  be a  $\Sigma$ -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  $c_1 = \text{Com}_p(x_1), c_2 = \text{Com}_q(x_2), c_3 = \text{Com}_p(k) = (\text{Com}_p(k_x), \text{Com}_p(k_y))$ , 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:  $\text{PK}\{(x_2, k, r_2, r_3) : f_2(x_2) = k \wedge c_2 = \text{Com}_q(x_2) \wedge c_3 = \text{Com}_p(k)\}$ .
- The prover uses the sigma protocol  $\Pi_1$  to give the following proof.  $\text{PK}\{(x_1, k, r_1, r_3) : f_1(x_1, T(k)) = z \wedge c_1 = \text{Com}_p(x_1) \wedge c_3 = \text{Com}_p(k)\}$ .

**Theorem 5.1 (Function Composition).** *The constructions composition are non-interactive zero-knowledge arguments  $\text{PK}\{(x_1, \dots, x_n, y_1, \dots, y_m) : f_1(x_1, \dots, x_n, f_2(y_1, \dots, y_m)) = z\}$ , as per Definition 2.2, for any  $f_1, f_2 \in \{\text{algebraic, arithmetic}\}$  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`:  $\text{PK}\{(x_1, x_2, x_3) : f_1(x_1, x_2) \vee f_2(x_1, x_3) = 1\}$ , where the  $f_i$ s 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  $f_1 : \mathbb{F}_p^2 \rightarrow \{0, 1\}$ , and  $f_2 : \mathbb{F}_q^2 \rightarrow \{0, 1\}$ ,  $q < p$ . An example is composing proofs for two SNARK systems that work over different elliptic curve groups.

- The prover commits to the inputs by computing,  $c_1 = \text{Com}_p(x_1), c'_1 = \text{Com}_q(x_1), c_2 = \text{Com}_p(x_2), c_3 = \text{Com}_q(x_3)$ , and to the output bits  $b_1 = f_1(x_1, x_2), b_2 = f_2(x_1, x_3), c_4 = \text{Com}_p(b_1), c_5 = \text{Com}_q(b_2), c'_5 = \text{Com}_p(b_2)$ .  $x_1$  and  $b_2$  are committed to in both groups of order  $p$  and  $q$ .
- The prover uses `zk-comIOSnark` to give proofs.  
 $\text{PK}\{(x_1, x_2, b_1, r_1, r_2, r_4) : f_1(x_1, x_2) = b_1 \wedge c_1 = \text{Com}_p(x_1) \wedge c_2 = \text{Com}_p(x_2) \wedge c_4 = \text{Com}_p(b_1)\}$ .  
 $\text{PK}\{(x'_1, x_3, b_2, r'_1, r_3, r_5) : f_2(x'_1, x_3) = b_2 \wedge c'_1 = \text{Com}_q(x'_1) \wedge c_3 = \text{Com}_q(x_3) \wedge c_5 = \text{Com}_q(b_2)\}$ .

- The prover uses the protocol **Eq** to prove that  $c'_1$  and  $c_1$  are commitments to the same value.  
 $\text{PK}\{(x_1, x'_1, r_1, r'_1) : x_1 \equiv x'_1 \pmod{q} \wedge c_1 = \text{Com}_p(x_1) \wedge c'_1 = \text{Com}_q(x_1)\}$
- The prover uses the protocol **Eq** to prove that  $c'_5$  and  $c_5$  are commitments to the same value.  
 $\text{PK}\{(b_2, b'_2, r_5, r'_5) : b_2 \equiv b'_2 \pmod{q} \wedge c_5 = \text{Com}_q(b_2) \wedge c'_5 = \text{Com}_p(b'_2)\}$
- The prover uses the Sigma protocol OR-transform to give the following proof.  
 $\text{PK}\{(b_1, b_2, r_4, r_5) : (b_1 = 1 \wedge c_4 = \text{Com}_p(b_1)) \vee (b_2 = 1 \wedge c'_5 = \text{Com}_p(b_2))\}$

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  $f_1 : \mathbb{F}_p^2 \rightarrow \{0, 1\}$ ,  $f_2 : \mathbb{Z}_q^2 \rightarrow \{0, 1\}$ ,  $q < p$ . Let  $\Pi$  be a  $\Sigma$ -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,  $c_1 = \text{Com}_q(x_1)$ ,  $c'_1 = \text{Com}_p(x_1)$ ,  $c_2 = \text{Com}_p(x_2)$ ,  $c_3 = \text{Com}_q(x_3)$ . 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  $c_4 = \text{Com}_p(b_1)$ ,  $c_5 = \text{Com}_q(b_2)$ ,  $c'_5 = \text{Com}_p(b_2)$ .
- The prover uses **comIOSnark** to give the following proof.  
 $\text{PK}\{(x'_1, x_2, b_1, r'_1, r_2, r_4) : f_1(x'_1, x_2) = b_1 \wedge c'_1 = \text{Com}_p(x_1) \wedge c_2 = \text{Com}_p(x_2) \wedge c_4 = \text{Com}_p(b_1)\}$ .
- The prover uses the protocol  $\Pi$  and protocol **comBitSigma** (Figure 3) to prove the following.  
 $\text{PK}\{(x_1, x_3, b_2, r_1, r_3, r_5) : f_2(x_1, x_3) = b_2 \wedge c_1 = \text{Com}_q(x_1) \wedge c_3 = \text{Com}_q(x_3) \wedge c_5 = \text{Com}_q(b_2)\}$
- The prover uses the protocol **Eq** to prove that  $c'_1$  and  $c_1$  are commitments to the same value.  
 $\text{PK}\{(x_1, x'_1, r_1, r'_1) : x_1 \equiv x'_1 \pmod{q} \wedge c_1 = \text{Com}_q(x_1) \wedge c'_1 = \text{Com}_p(x_1)\}$
- The prover uses the protocol **Eq** to prove that  $c'_5$  and  $c_5$  are commitments to the same value.  
 $\text{PK}\{(b_2, b'_2, r_5, r'_5) : b_2 \equiv b'_2 \pmod{q} \wedge c_5 = \text{Com}_q(b_2) \wedge c'_5 = \text{Com}_p(b'_2)\}$
- The prover uses the Sigma protocol OR-transform to prove the following.  
 $\text{PK}\{(b_1, b_2, r_4, r_5) : (b_1 = 1 \wedge c_4 = \text{Com}_p(b_1)) \vee (b_2 = 1 \wedge c_5 = \text{Com}_q(b_2))\}$ .

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

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

### 5.3 AND Composition

Techniques shown in Section 5.2 extend for proofs of the form,  $\text{PK}\{(x_1, x_2, x_3) : f_1(x_1, x_2) \wedge f_2(x_1, x_3) = 1\}$  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 `comInSnark` for the arithmetic part,  $\Sigma$ -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  $\text{Com}(k)$  to mean a commitment to the coordinates of  $k$ , i.e.  $\text{Com}(k) = (\text{Com}(k_x), \text{Com}(k_y))$ . The Bitcoin address corresponding to a key  $y$  is given by  $\mathbf{h} = H(y)$ , where  $H$  hashes  $y$  to a more compact representation. We denote the balance associated with an address  $\mathbf{h}$  by  $\text{bal}(\mathbf{h})$ .

**Proof of assets** We give the proof of assets in Figure 6, 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:  $\mathcal{PK} = \{\mathbf{h}_1, \dots, \mathbf{h}_n\}$  from the public data available on the blockchain. Let  $x_1, \dots, x_n$  be the corresponding secret keys, so that  $\mathbf{h}_i = H(g^{x_i})$ ,  $s_i$  indicates whether the exchange knows the  $i$ th secret key. The total assets can now be expressed as  $\text{Assets} = \sum_{i=1}^n s_i \cdot \text{bal}(\mathbf{h}_i)$ . The public data available on the blockchain is  $\mathbf{h}_i = H(y_i), p_i = g^{\text{bal}(\mathbf{h}_i)}$  for all  $i \in [1, n]$ .

Zero-knowledge and soundness of the proof of assets follow from properties of our constructions for compound statements (Theorems 5.1, 5.2) and properties of the Sigma protocols used. Proofs of liabilities and solvency have been moved to the full version because they are very similar to Provisions. 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 7.



- The exchange computes the commitments. For  $i \in [1, n]$ , commit to  $x_i$  by publishing  $\alpha_i = \text{Com}_q(x_i) = g^{x_i} h^{r_i}$ , and commit to  $y_i$  by publishing  $\beta_i = \text{Com}_q(y_i)$ .
- The exchange commits to the balance in each address for the public keys he controls and to 0 otherwise, by publishing  $u_i = \text{Com}_q(s_i \cdot \text{bal}(h_i)) = g^{s_i \cdot \text{bal}(h_i)} h^{t_i}$ ,  $s_i \in \{0, 1\}$ , 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  $i$ ,

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

where  $f_2(x) = g^x$  and  $f_1(y, h) = 1$  if  $H(y) = h$  and 0 otherwise.

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

Fig. 6: Proof of assets

## 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 [30] 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  $\sigma$ , and an RSA modulus  $N$ , verification involves checking whether  $\sigma^e \bmod n = h$ , where  $h = \text{padding}(\text{SHA}(m))$ . The construction given in Figure 7 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 7. 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  $\sigma$  by computing  $c_1 = \text{Com}_p(m)$ ,  $c_2 = \text{Com}_p(h)$ ,  $c_3 = \text{Com}_n(\sigma)$ ,  $c_4 = \text{Com}_n(h)$  for  $p < n$ .
- The prover uses zk-comIOSnark to give a proof that the hash digest is correct, given  $c_1$  and  $c_2$ .  
 $\text{PK}\{(m, h, r_1, r_2) : \text{padding}(\text{SHA}(m)) = h \wedge c_1 = \text{Com}_p(m) \wedge c_2 = \text{Com}_p(h)\}$ .
- The prover uses a sigma protocol to prove knowledge of  $e$ -th root of a committed value [19].  
 $\text{PK}\{(h, \sigma, r_2, r_3) : \sigma^e \bmod n = h \wedge c_2 = \text{Com}_n(h) \wedge c_3 = \text{Com}_n(\sigma)\}$ .
- The prover uses the protocol Eq to prove that the commitments  $c_2$  and  $c_4$  are to the same value:  $\text{PK}\{(h, h', r_2, r_4) : c_2 = \text{Com}_p(h) \wedge c_4 = \text{Com}_n(h') \wedge h \equiv h' \bmod p\}$ .

Fig. 7: RSA Signature Verification

## References

1. Secp256k1. <https://en.bitcoin.it/wiki/Secp256k1>
2. Technical background of version 1 bitcoin addresses. [https://en.bitcoin.it/wiki/Technical\\_background\\_of\\_version\\_1\\_Bitcoin\\_addresses](https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses)
3. Zcash 1.0 “Sprout” Guide. <https://github.com/zcash/zcash/wiki/1.0-User-Guide>
4. Zcash Parameter Generation. <https://z.cash/technology/paramgen.html>
5. Aho, A. (ed.): 19th ACM STOC. ACM Press (May 1987)
6. Ames, S., Hazay, C., Ishai, Y., Venkatasubramanian, M.: Liger: Lightweight sub-linear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17. pp. 2087–2104. ACM Press (Oct / Nov 2017)
7. Ben-Or, M., Goldreich, O., Goldwasser, S., Håstad, J., Kilian, J., Micali, S., Rogaway, P.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO’88. LNCS, vol. 403, pp. 37–56. Springer, Heidelberg (Aug 1990)
8. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014)
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (Aug 2013)
10. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 781–796. USENIX Association, San Diego, CA (2014)
11. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Goldwasser, S. (ed.) ITCS 2012. pp. 326–349. ACM (Jan 2012)
12. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (Mar 2013)

13. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC. pp. 103–112. ACM Press (May 1988)
14. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Peneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (May 2000)
15. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Efficient range proofs for confidential transactions. Cryptology ePrint Archive, Report 2017/1066 (2017), <https://eprint.iacr.org/2017/1066>
16. Camenisch, J., Chaabouni, R., shelat, a.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (Dec 2008)
17. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001)
18. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT’99. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (May 1999)
19. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr. [49], pp. 410–424
20. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1825–1842. ACM (2017)
21. Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 499–530. Springer, Heidelberg (Aug 2016)
22. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO’82. pp. 199–203. Plenum Press, New York, USA (1982)
23. Cohen, J.D., Fischer, M.J.: A robust and verifiable cryptographically secure election scheme (extended abstract). In: 26th FOCS. pp. 372–382. IEEE Computer Society Press (Oct 1985)
24. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270. IEEE Computer Society Press (May 2015)
25. Cramer, R. (ed.): TCC 2012, LNCS, vol. 7194. Springer, Heidelberg (Mar 2012)
26. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO’94. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994)
27. Dagher, G.G., Bünz, B., Boneh, D., Clark, J., Boneh, D.: Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 15. pp. 720–731. ACM Press (Oct 2015)
28. Dãmgård, I.: On Sigma Protocols. <http://www.cs.au.dk/~ivan/Sigma.pdf>
29. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer [25], pp. 54–74
30. Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Parno, B.: Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of

- verifiable computation. In: 2016 IEEE Symposium on Security and Privacy. pp. 235–254. IEEE Computer Society Press (May 2016)
31. Edwards, H.: A normal form for elliptic curves. *Bulletin of the American Mathematical Society* 44(3), 393–422 (2007)
  32. Feige, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Aho [5], pp. 210–217
  33. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS. pp. 308–317. IEEE Computer Society Press (Oct 1990)
  34. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO’86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
  35. Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: Adaptive verifiable computations on outsourced data. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 16. pp. 1304–1316. ACM Press (Oct 2016)
  36. Fortnow, L.: The complexity of perfect zero-knowledge (extended abstract). In: Aho [5], pp. 204–209
  37. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr. [49], pp. 16–30
  38. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology* 19(2), 169–209 (Apr 2006)
  39. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013)
  40. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. (2016)
  41. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS. pp. 174–187. IEEE Computer Society Press (Oct 1986)
  42. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho [5], pp. 218–229
  43. Goldwasser, S., Lin, H., Rubinfeld, A.: Delegation of computation without rejection problem from designated verifier CS-Proofs. *Cryptology ePrint Archive*, Report 2011/456 (2011), <http://eprint.iacr.org/2011/456>
  44. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010)
  45. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 253–280. Springer, Heidelberg (Apr 2015)
  46. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: Günther, C.G. (ed.) EUROCRYPT’88. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (May 1988)
  47. 2013 IEEE Symposium on Security and Privacy. IEEE Computer Society Press (May 2013)
  48. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007)

49. Kaliski Jr., B.S. (ed.): CRYPTO'97, LNCS, vol. 1294. Springer, Heidelberg (Aug 1997)
50. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732. ACM (1992)
51. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer [25], pp. 169–189
52. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg (Dec 2013)
53. Micali, S.: Computationally sound proofs. SIAM Journal on Computing 30(4), 1253–1298 (2000)
54. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed E-cash from Bitcoin. In: IEEE S&P 2013 [47], pp. 397–411
55. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: 22nd ACM STOC. pp. 427–437. ACM Press (May 1990)
56. Noether, S., Mackenzie, A., Team, M.C.: Ring confidential transactions. <https://lab.getmonero.org/pubs/MRL-0005.pdf>
57. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE S&P 2013 [47], pp. 238–252
58. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992)
59. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (May 1996)
60. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
61. Vadhan, S.P.: A study of statistical zero-knowledge proofs. Ph.D. thesis, Massachusetts Institute of Technology (1999)
62. Wilcox, Z.: Proving bitcoin reserves. <https://iwilcox.me.uk/2014/proving-bitcoin-reserves>

## 7 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 [15], 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	$( H  + p^3)n + c$ exp.
Our composition techniques	pay-to-pub, pay-to-hash	$2396n + \log p + \log n$	$( H  + 30p + 1800)n + c$ 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  $\lceil \log \text{Max} \rceil = 51$ ,  $p$  is the bit length of the modulus for exponentiation (size of the field over which 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.)

*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 [21], and our composition.

zk technique	Feature	Proof size	Prover
Cinderella	non-interactive	7	$ H $ + additional 164,826 equations for RSA (as optimized in Cinderella)
GC + Sigma [21]	interactive	$ H $	$ m  +  h $ exp. + $ H $ symmetric-key operations
Our composition techniques	non-interactive	$42 + \log p$	$ H  + \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.)