Prover-Efficient Commit-And-Prove Zero-Knowledge SNARKs

Abstract. Zk-SNARKs (succinct non-interactive zero-knowledge arguments of knowledge) are needed in many applications. Unfortunately, all previous zk-SNARKs for interesting languages are either inefficient for prover, or are non-adaptive and based on an commitment scheme that does depend both on the prover's input and on the language, i.e., they are not commit-and-prove (CaP) SNARKs. We propose a proof-friendly extractable commitment scheme, and use it to construct prover-efficient adaptive CaP succinct zk-SNARKs for different languages, that can all reuse committed data. In new zk-SNARKs, the prover computation is dominated by a linear number of cryptographic operations. We use batch-verification to decrease the verifier's computation.

Keywords: Commit-and-prove, CRS, NIZK, numerical NP-complete languages, range proof, Subset-Sum, zk-SNARK

1 Introduction

Recently, there has been a significant surge of activity in studying succinct non-interactive non-interactive zero knowledge (NIZK) arguments of knowledge (also known as zk-SNARKs) [5–7,9,19,21,26,28,36,37,40]. The prover of a zk-SNARK outputs a short (ideally, a small number of group elements) argument π that is used to convince many different verifiers in the truth of the same claim without leaking any side information. The verifiers can verify independently the correctness of π, without communicating with the prover. The argument must be efficiently verifiable. Constructing the argument can be less efficient, since it is only done once. Still, prover-efficiency is important, e.g., in a situation where a single server has to create many arguments to different clients or other servers.

Many known zk-SNARKs are non-adaptive, meaning that the common reference string, CRS, can depend on the concrete instance of the language (e.g., the circuit in the case of Circuit-SAT). In an adaptive zk-SNARK, the CRS is independent on the instance and thus can be reused many times. This distinction is important, since generation and distribution of the CRS must be done securely. The most efficient known *non-adaptive* zk-SNARKs for NP-complete languages from [26] are based on either Quadratic Arithmetic Programs (QAP, for arithmetic Circuit-SAT) or Quadratic Span Programs (QSP, for Boolean Circuit-SAT). There, the prover computation is dominated by $\Theta(n)$ cryptographic operations (see App. A for a clarification on cryptographic/non-cryptographic operations), where n is the number of the gates. QAP, QSP [26,37] and other related approaches like SSP [21] have the same asymptotic complexity.

QSP-based Circuit-SAT SNARK can be made adaptive by using universal circuits [47]. Then, the CRS depends on the construction of universal circuit and not on the concrete input circuit itself. However, since the size of a universal

circuit is $\Theta(n \log n)$, the prover computation in resulting adaptive zk-SNARKs is $\Theta(n \log^2 n)$ non-cryptographic operations and $\Theta(n \log n)$ cryptographic operations. (In the case of QAP-based arithmetic Circuit-SAT SNARK, one has to use universal arithmetic circuits [44] that have an even larger size $\Theta(r^4 n)$, where r is the degree of the polynomial computed by the arithmetic circuit. Thus, we will mostly give a comparison to the QSP-based approach.)

Since Valiant's universal circuits incur a large constant $c = 19$ in the $\Theta(\cdot)$ expression, a common approach [34,45] is to use universal circuits with the overhead of $\Theta(\log^2 n)$ but with a smaller constant $c = 1/2$ in $\Theta(\cdot)$. The prover computation in the resulting adaptive zk-SNARKs is $\Theta(n \log^3 n)$ non-cryptographic operations and $\Theta(n \log^2 n)$ cryptographic operations.[1]

Another important drawback of the QSP/QAP-based SNARKs is that they use a circuit-dependent commitment scheme. To use the same input data in multiple sub-SNARKs, one needs to construct a single large circuit that implements all sub-SNARKs, making the SNARK and the resulting *new* commitment scheme more complicated.[2] In particular, these SNARKs are not commit-and-prove (CaP [16, 33]) SNARKs. We recall that in CaP SNARKs, a commitment scheme C is fixed first, and the statement consists of commitments of the witness using C; see Sect. 2. Hence, a CaP commitment scheme is *instance-independent*. In addition, one would like the commitment scheme to be *language-independent*, enabling one to first commit to the data and only then to decide in what applications (e.g., verifiable computation of a later fixed function) to use it.

See Tbl. 1 for a brief comparison of the efficiency of proposed adaptive zk-SNARKs for NP-complete languages. Subset-Sum is here brought as an example of a wider family of languages; it can be replaced everywhere say with Partition or Knapsack, see Sect. 8. Here, $N = r_3^{-1}(n) = o(n2^{2\sqrt{2\log_2 n}})$, where $r_3(n)$ is the density of the largest progression-free set in $\{1, \ldots, n\}$. According to the current knowledge, $r_3^{-1}(n)$ is comparable to (or only slightly smaller than) n^2 for $n < 2^{12}$; this makes all known CaP SNARKs [23, 28, 36] arguably impractical unless n is really small. In all cases, the verifier's computation is dominated by either $\Theta(n)$ cryptographic or $\Theta(n \log n)$ non-cryptographic operations (with the verifier's online computation usually being $\Theta(1)$), and the communication consists of a small constant number of group elements.[3] Given all above, it is natural to ask the following question:

[1] Recently, [19] proposed an independent methodology to improve the prover's computational complexity in QAP-based arguments. However, [19] does not spell out their achieved prover's computational complexity. Hence, while we make use of some of their unrelated techniques (e.g., proof boot-strapping), we are unable to give a complete comparison between the current work and [19]. We hope that a good comparison can be given in a future work.

[2] Again, this seems to have been addressed by Geppetto, [19], however, it is not clear from [19] how the actual commitment scheme relates to the circuit

[3] We emphasize that Circuit-SAT is *not* our focus; the lines corresponding to Circuit-SAT are provided only for the sake of comparison. One can use proof boot-strapping [19] to decrease the length of the resulting Circuit-SAT argument from $\Theta(\log n)$, as stated in [38], to $\Theta(1)$; we omit further discussion.

Table 1. Prover-efficiency of known *adaptive* zk-SNARKs for NP-complete languages. Here, n is the number of the gates (in the case of Circuit-SAT) and the number of the integers (in the case of Subset-Sum). Green background denotes the best known asymptotic complexity of the *concrete* NP-complete language w.r.t. to the concrete parameter. The solutions marked with * use proof bootstrapping from [19]

Paper	Language	Prover computation		|CRS|
		non-crypt. op.	crypt. op.	
Not CaP-s				
QAP, QSP ([21, 26, 37])	Circuit-SAT	$\Theta(n \log^2 n)$	$\Theta(n \log n)$	$\Theta(n)$
CaP-s				
Gro10 ([28])	Circuit-SAT	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Lip12 ([36])	Circuit-SAT	$\Theta(n^2)$	$\Theta(N)$	$\Theta(N)$
Lip14 + Lip12 ([36, 38])*	Circuit-SAT	$\Theta(N \log^2 n)$	$\Theta(N \log n)$	$\Theta(N \log n)$
Lip14 + current paper ([38])*	Circuit-SAT	$\Theta(n \log^2 n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
FLZ13 ([23])	Subset-Sum	$\Theta(N \log n)$	$\Theta(N)$	$\Theta(N)$
Current paper	Subset-Sum	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(n)$

The Main Question of This Paper: *Is it possible to construct* adaptive CaP *zk-SNARKs for NP-complete languages where the prover computation is dominated by a linear number of cryptographic operations?*

Our Contributions. We answer the "main question" positively by improving on Groth's modular approach [28]. Using the modular approach allows us to modularize the security analysis, first proving the security of underlying building blocks (the product and the shift SNARKs), and then composing them to construct master SNARKs for even NP-complete languages. The security of master SNARKs follows easily from the security of the basic SNARKs. We also use batch verification to speed up verification of almost all known SNARKs.

All new SNARKs use the same commitment scheme, the interpolating commitment scheme. Hence, one can reuse their input data to construct CaP zk-SNARKs for different unrelated languages, chosen only after the commitment was done. Thus, one can first commit to some data, and only later decide in which application and to what end to use it. Importantly, by using CaP zk-SNARKs, one can guarantee that all such applications use exactly the same data.

The resulting SNARKs are not only commit-and-prove, but also very efficient, and often more efficient than any previously known SNARKs. The new CaP SNARKs have prover-computation dominated by $\Theta(n)$ cryptographic operations, with the constant in $\Theta(\cdot)$ being reasonably small. Importantly, we propose the most efficient known succinct range SNARK. Since the resulting zk-SNARKs are sufficiently different from QAP-based zk-SNARKs, we hope that our methodology by itself is of independent interest. Up to the current paper, Groth's modular approach has resulted in significantly less efficient zk-SNARKs than the QSP/QAP-based approach.

In Sect. 3, we construct a new natural extractable trapdoor commitment scheme (the interpolating commitment scheme). Here, commitment to $\boldsymbol{a} \in \mathbb{Z}_p^n$, where n is a power of 2, is a short garbled and randomized version $g_1^{L_{\boldsymbol{a}}(\chi)}(g_1^{\chi^n-1})^r$ of the Lagrange interpolating polynomial $L_{\boldsymbol{a}}(X)$ of $\boldsymbol{a}$, for a random secret key χ, together with a knowledge component. This commitment scheme is arguably a very natural one, and in particular its design is not influenced by the desire to tailor it to one concrete application. Nevertheless, as we will see, using it improves the efficiency of many constructions while allowing to reuse many existing results.

The new CaP zk-SNARKs are based on the interpolating commitment scheme and two CaP witness-indistinguishable SNARKs: a product SNARK (given commitments to vectors $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, it holds that $c_i = a_i b_i$; see [23, 28, 36]), and a shift SNARK (given commitments to $\boldsymbol{a}$, $\boldsymbol{b}$, it holds that $\boldsymbol{a}$ is a coordinate-wise shift of $\boldsymbol{b}$; see [23]). One can construct an adaptive Circuit-SAT CaP zk-SNARK from $\Theta(\log n)$ product and shift SNARKs [28, 38], or adaptive CaP zk-SNARKs for NP-complete languages like Subset-Sum (and a similar CaP range SNARK) by using a constant number of product and shift SNARKs [23].

In Sect. 4, we propose a CaP product SNARK, that is an argument of knowledge under a computational and a knowledge (needed solely to achieve extractability of the commitment scheme) assumption. Its prover computation is dominated by $\Theta(n \log n)$ non-cryptographic and $\Theta(n)$ cryptographic operations. This can be compared to $r_3^{-1}(n)$ non-cryptographic operations in [23]. The speed-up is mainly due to the use of the interpolating commitment scheme.

In Sect. 5, we propose a variant of the CaP shift SNARK of [23], secure when combined with the interpolating commitment scheme. We prove that this SNARK is an adaptive argument of knowledge under a computational and a knowledge assumption. It only requires the prover to perform $\Theta(n)$ cryptographic and non-cryptographic operations.

Product and shift SNARKs are already very powerful by itself. E.g., a prover can commit to her input vector $\boldsymbol{a}$. Then, after agreeing with the verifier on a concrete application, she can commit to a different yet related input vector (that say consists of certain permuted subset of $\boldsymbol{a}$'s coefficients), and then use the basic SNARKs to prove that this was done correctly. Here, she may use the permutation SNARK [38] that consists of $O(\log n)$ product and shift SNARKs. Finally, she can use another, application-specific, SNARK (e.g., a range SNARK) to prove that the new committed input vector has been correctly formed.

In Sect. 6, we describe a modular adaptive CaP zk-SNARK, motivated by [23], for the NP-complete language, Subset-Sum. (Subset-Sum was chosen by us mainly due to the simplicity of the SNARK; the rest of the paper considers more applications.) This SNARK consists of three commitments, one application of the shift SNARK, and three applications of the product SNARK. It is a zk-SNARK given that the commitment scheme, the shift SNARK, and the product SNARK are secure. Its prover computation is strongly dominated by $\Theta(n)$ cryptographic operations, where n is the instance size, the number of integers. More precisely, the prover has to perform only nine ($\approx n$)-wide multi-exponentiations, which makes the SNARK efficient not only asymptotically (to compare, the size

of Valiant's arithmetic circuit has constant 19, and this constant has to be multiplied by the overhead of non-adaptive QSP/QAP/SSP-based solutions). Thus, we answer positively to the stated main question of the current paper. Moreover, the prover computation is highly parallelizable, while the *online* verifier computation is dominated by 17 pairings (this number will be decreased later).

In Sect. 7, we propose a new CaP range zk-SNARK that the committed value belongs to a range $[L\,..\,H]$. This SNARK looks very similar to the Subset-Sum SNARK, but with the integer set $\boldsymbol{S}$ of the Subset-Sum language depending solely on the range length. Since here the prover has a committed input, the simulation of the range SNARK is slightly more complicated than of the Subset-Sum SNARK. Its prover-computation is similarly dominated by $\Theta(n)$ cryptographic operations, where this time $n := \lceil \log_2(H - L) \rceil$. Differently from the Subset-Sum SNARK, the verifier computation is dominated only by $\Theta(1)$ cryptographic operations, more precisely, by 19 pairings (also this number will be decreased later). Importantly, this SNARK is computationally more efficient than any of the existing *succinct* range SNARKs either in the standard model (i.e., randomoracle-less) or in the random oracle model. E.g., the prover computation in [35] is $\Theta(n^2)$ under the Extended Riemann Hypothesis, and the prover computation in [23] is $\Theta(r^{-3}(n) \log r^{-3}(n))$. It is also significantly simpler than the range SNARKs of [18,23], mostly since we do not have to consider different trade-offs between computation and communication.

In Sect. 8, we outline how to use the new basic SNARKs to construct efficient zk-SNARKs for several other NP-complete languages like Boolean and arithmetic Circuit-SAT, Two-Processor Scheduling, Subset-Product, Partition, and Knapsack [24]. Tbl. 1 includes the complexity of Subset-Sum and Circuit-SAT, the complexity of most other SNARKs is similar to that of Subset-Sum zk-SNARK. It is an interesting open problem why some NP-complete languages like Subset-Sum have more efficient zk-SNARKs in the modular approach (equivalently, why their verification can be performed more efficiently in the parallel machine model that consists of Hadamard product and shift) than languages like Circuit-SAT.

In Sect. 9, we show that by using batch-verification [4], one can decrease the verifier's computation of all presented SNARKs. In particular, one can decrease the verifier's computation in the new Range SNARK from 19 pairings to 8 pairings, one 4-way multi-exponentiation in $\mathbb{G}_1$, two 3-way multi-exponentiations in $\mathbb{G}_1$, one 2-way multi-exponentiation in $\mathbb{G}_1$, three exponentiations in $\mathbb{G}_1$, and one 3-way multi-exponentiation in $\mathbb{G}_2$.. Since one exponentiation is much cheaper than one pairing [15] and one m-way multi-exponentiation is much cheaper than m exponentiations [42,46], this results in a significant win for the verifier. A similar technique can be used to also speed up other SNARKs; a good example here is the Circuit-SAT argument from [38] that uses $\Theta(\log n)$ product and shift arguments. To compare, in Pinocchio [40] and Geppetto [19], the verifier has to execute 11 pairings; however, batch-verification can also be used to decrease this to 8 pairings and a small number of (multi-)exponentiations.

Finally, all resulting SNARKs work on data that has been committed to by using the interpolating commitment scheme. This means that one can repeatedly reuse committed data to compose different zk-SNARKs (e.g., to show that we know a satisfying input to a circuit, where the first coefficient belongs to a certain range). This is not possible with the known QSP/QAP-based zk-SNARKs where one would have to construct a single circuit of possibly considerable size, say n'. Moreover, in the QSP/QAP-based SNARKs, one has to commit to the vector, the length of which is equal to the total length of the input and witness (e.g., n' is the number of wires in the case of Circuit-SAT). By using a modular solution, one can instead execute several zk-SNARKs with smaller values of the input and witness size; this can make the SNARK more prover-efficient since the number of non-cryptographic operations is superlinear. This emphasizes another benefit of the modular approach: one can choose the value n, the length of the vectors, accordingly to the desired tradeoff, so that larger n results in faster verifier computation, while smaller n results in faster prover computation. We are not aware of such a tradeoff in the case of the QSP/QAP-based approach.

We provide some additional discussion (about the relation between n and then input length, and about possible QSP/QAP-based solutions) in Sect. 10.

2 Preliminaries

By default, all vectors have dimension n. Let $\boldsymbol{a} \circ \boldsymbol{b}$ denote the Hadamard (i.e., element-wise) product of two vectors, with $(\boldsymbol{a} \circ \boldsymbol{b})_i = a_i b_i$. We say that $\boldsymbol{a}$ is a *shift-right-by-z* of $\boldsymbol{b}$, $\boldsymbol{a} = \boldsymbol{b} \gg z$, iff $(a_n, \ldots, a_1) = (0, \ldots, 0, b_n, \ldots, b_{1+z})$. For a tuple of polynomials $\mathcal{F} \subseteq \mathbb{Z}_p[X, Y_1, \ldots, Y_{m-1}]$, define $Y_m\mathcal{F} = (Y_m \cdot f(X, Y_1, \ldots, Y_{m-1}))_{f\in\mathcal{F}} \subseteq \mathbb{Z}_p[X, Y_1, \ldots, Y_m]$. For a tuple of polynomials $\mathcal{F}$ that have the same domain, denote $h^{\mathcal{F}(\boldsymbol{a})} := (h^{f(\boldsymbol{a})})_{f\in\mathcal{F}}$. For a group $\mathbb{G}$, let $\mathbb{G}^*$ be the set of its invertible elements. Since the direct product $\mathbb{G}_1 \times \cdots \times \mathbb{G}_m$ of groups is also a group, we use notation like $(g_1, g_2)^c = (g_1^c, g_2^c) \in \mathbb{G}_1 \times \mathbb{G}_2$ without prior definition. Let κ be the security parameter. We denote $f(\kappa) \approx_\kappa g(\kappa)$ if $|f(\kappa) - g(\kappa)|$ is negligible in κ.

On input 1^κ, a *bilinear map generator* BP returns $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are three multiplicative cyclic groups of prime order p (with $\log p = \Omega(\kappa)$), and $\hat{e}$ is an efficient bilinear map $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies in particular the following two properties, where g_1 (resp., g_2) is an arbitrary generator of $\mathbb{G}_1$ (resp., $\mathbb{G}_2$): (i) $\hat{e}(g_1, g_2) \neq 1$, and (ii) $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$. Thus, if $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1^c, g_2^d)$ then $ab \equiv cd \pmod p$. We also give BP another input, n (intuitively, the input length), and allow p to depend on n. We assume that all algorithms that handle group elements verify by default that their inputs belong to corresponding groups and reject if they do not. In the case of many practically relevant pairings, arithmetic in (say) $\mathbb{G}_1$ is considerably cheaper than in $\mathbb{G}_2$; hence, we count separately exponentiations in both groups.

For $\kappa = 128$, the current recommendation is to use an optimal (asymmetric) Ate pairing [30] over Barreto-Naehrig curves [3, 41]. In that case, at security level of $\kappa = 128$, an element of $\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$ can be represented in respectively

256/512/3072 bits. To speed up interpolation, we will additionally need the existence of the n-th, where n is a power of 2, primitive root of unity modulo p (under this condition, one can interpolate in time $\Theta(n \log n)$, otherwise, interpolation takes time $\Theta(n \log^2 n)$, [25]). For this, it suffices that $(n+1) \mid (p-1)$ (recall that p is the elliptic curve group order). Fortunately, given κ and a practically relevant value of n, one can easily find a Barreto-Naehrig curve such that $(n+1) \mid (p-1)$ holds; such an observation was made also in [7]. For example, if $\kappa = 128$ and $n = 2^{10}$, one can use Alg. 1 of [3] to find an elliptic curve group of prime order $N(x_0)$ over a finite field of prime order $P(-x_0)$ for $x_0 = 1753449050$, where $P(x) = 36x^4+36x^3+24x^2+6x+1$, $T(x) = 6x^2+1$, and $N(x) = P(x)+1-T(x)$. One can then use the curve $E : y^2 = x^3 + 6$.

In proof bootstrapping [19], one needs an additional elliptic curve group $\tilde{E}$ over a finite field of order $N(x_0)$ (see [19] for additional details). Such elliptic curve group can be found by using the Cocks-Pinch method [10]; note that $\tilde{E}$ has somewhat less efficient arithmetic than E.

The security of the new commitment scheme and of the new SNARKs depends on the following q-type assumptions, variants of which have been used in many previous papers. The assumptions are parameterized but non-interactive in the sense that q is related to the parameters of the language (most generally, to the input length) and not to the number of the adversarial queries. All known (to us) adaptive zk-SNARKs are based on q-type assumptions about BP.

Let $d(n) \in poly(n)$ be a function. Then, BP is

- *$d(n)$-PDL (Power Discrete Logarithm, [36]) secure* if for any $n \in poly(\kappa)$ and any non-uniform probabilistic polynomial-time (NUPPT) adversary A,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa, n), (g_1, g_2, \chi) \leftarrow_r \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p : \\ \mathsf{A}(\mathsf{gk}; ((g_1, g_2)^{\chi^i})_{i=0}^{d(n)}) = \chi\end{array}\right] \approx_\kappa 0 \ .$$

- *n-TSDH (Target Strong Diffie-Hellman, [13, 40]) secure* if for any $n \in poly(\kappa)$ and any NUPPT adversary A,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa, n), (g_1, g_2, \chi) \leftarrow_r \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p : \\ \mathsf{A}(\mathsf{gk}; ((g_1, g_2)^{\chi^i})_{i=0}^{n}) = (r, \hat{e}(g_1, g_2)^{1/(\chi - r)})\end{array}\right] \approx_\kappa 0 \ .$$

For algorithms A and X_A, we write $(y; y') \leftarrow (\mathsf{A}||X_\mathsf{A})(\chi)$ if A on input χ outputs y, and X_A on the same input (including the random tape of A) outputs y' [1]. We will need knowledge assumptions w.r.t. several knowledge secrets γ_i. Let m be the number of different knowledge secrets in any concrete SNARK. Let $\mathcal{F} = (P_i)_{i=0}^n$ be a tuple of univariate polynomials, and $\mathcal{G}_1$ (resp., $\mathcal{G}_2$) be a tuple of univariate (resp., m-variate) polynomials. Let $i \in [1\,..\,m]$. Then, BP is *$(\mathcal{F}, \mathcal{G}_1, \mathcal{G}_2, i)$-PKE (Power Knowledge of Exponent, [28]) secure* if for any

NUPPT adversary A there exists an NUPPT extractor X_{A}, such that

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa, n), (g_1, g_2, \chi, \boldsymbol{\gamma}) \leftarrow_r \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p \times \mathbb{Z}_p^m, \\ \boldsymbol{\gamma_{-i}} \leftarrow (\gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+1}, \ldots, \gamma_m), \mathsf{aux} \leftarrow (g_1^{\mathcal{G}_1(\chi)}, g_2^{\mathcal{G}_2(\chi, \boldsymbol{\gamma_{-i}})}), \\ (h_1, h_2; (a_i)_{i=0}^n) \leftarrow (\mathsf{A}||X_{\mathsf{A}})(\mathsf{gk}; (g_1, g_2^{\gamma_i})^{\mathcal{F}(\chi)}, \mathsf{aux}) : \\ \hat{e}(h_1, g_2^{\gamma_i}) = \hat{e}(g_1, h_2) \wedge h_1 \neq g_1^{\sum_{i=0}^n a_i P_i(\chi)}\end{array}\right] \approx_\kappa 0 \ .$$

Here, aux can be seen as the common auxiliary input to A and X_{A} that is generated by using benign auxiliary input generation [8]. If $\mathcal{F} = (X^i)_{i=0}^d$ for some $d = d(n)$, then we replace the first argument in $(\mathcal{F}, \ldots)$-PKE with d. If $m = 1$, then we omit the last argument i in $(\mathcal{F}, \ldots, i)$-PKE. While knowledge assumptions are non-falsifiable, we recall that non-falsifiable assumptions are needed to design succincts SNARKs for interesting languages [27].

By generalizing [14, 28, 36], one can show that the TSDH, PDL, and PKE assumptions hold in the generic bilinear group model.

Within this paper, $m \leq 2$, and hence we denote γ_1 just be γ, and γ_2 by δ.

Commitment Schemes. *An extractable trapdoor commitment scheme* in the CRS model [12] consists of two efficient algorithms $\mathsf{G_{com}}$ (that outputs a CRS ck and a trapdoor) and C (that, given ck, a message m and a randomizer r, outputs a commitment $\mathsf{C_{ck}}(m; r)$), and must satisfy the following security properties.

Computational binding: without access to the trapdoor, it is intractable to open a commitment to two different messages.

Trapdoor: given access to the original message, the randomizer and the trapdoor, one can open the commitment to any other message.

Perfect hiding: commitments of any two messages have the same distribution.

Extractability: given access to the CRS, the commitment, and the random coins of the committer, one can open the commitment to the committed message.

See, e.g., [28] for formal definitions. In the context of the current paper, the message is a vector from $\mathbb{Z}_p^n$. We denote the randomizer space by $\mathfrak{R}$.

Commit-And-Prove SNARKs. Let $\mathcal{R} = \{(u, w)\}$ be an efficiently verifiable relation with $|w| = \mathrm{poly}(|u|)$. Here, u is a statement, and w is a witness. Let $\mathcal{L} = \{u : \exists w, (u, w) \in \mathcal{R}\}$ be an **NP**-language. Let $n = |u|$ be the input length. For fixed n, we have a relation $\mathcal{R}_n$ and a language $\mathcal{L}_n$.

Following [16,33], we will define commit-and-prove (CaP) argument systems. Intuitively, a CaP non-interactive zero knowledge argument system for $\mathcal{R}$ allows to create a common reference string (CRS) crs, commit to some values w_i (say, $u_i = \mathsf{C_{ck}}(w_i; r_i)$, where ck is a part of crs), and then prove that a subset $u := (u_{i_j}, w_{i_j}, r_{i_j})_{j=1}^{\ell_m(n)}$ (for publicly known indices i_j) satisfies that u_{i_j} is a commitment of w_{i_j} with randomizer r_{i_j}, and that $(w_{i_j}) \in \mathcal{R}$.

Differently from most of the previous work (but see also [19]), our CaP argument systems will use computationally binding trapdoor commitment schemes. This means that without their openings, commitments $u_i = \mathsf{C_{ck}}(a_i; r_i)$ themselves do not define a valid relation, since u_i can be a commitment to any a_i',

given a suitable r_i'. Rather, we define a new relation

$$\mathcal{R}_{\mathsf{ck}} := \{(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{r}) : (\forall i, u_i = \mathsf{C}_{\mathsf{ck}}(w_i; r_i)) \wedge \boldsymbol{w} \in \mathcal{R}\} \ ,$$

and construct argument systems for $\mathcal{R}_{\mathsf{ck}}$.

Within this subsubsection, we let vectors $\boldsymbol{u}$, $\boldsymbol{w}$, and $\boldsymbol{r}$ be of dimension $\ell_m(n)$ for some polynomial $\ell_m(n)$. However, we allow committed messages w_i themselves to be vectors of dimension n. Thus, $\ell_m(n)$ is usually very small. In some argument systems (like the Subset-Sum SNARK in Sect. 6), also the argument will include some commitments. In such cases, technically speaking, $\boldsymbol{w}$ and $\boldsymbol{r}$ are of higher dimension than $\boldsymbol{u}$. To simplify notation, we will ignore this issue.

A *commit-and-prove non-interactive zero-knowledge argument system* [16,33] Π for $\mathcal{R}$ consists of an ($\mathcal{R}$-independent) trapdoor commitment scheme $\Gamma = (\mathsf{G}_{\mathsf{com}}, \mathsf{C})$ and of a non-interactive zero-knowledge argument system $(\mathsf{G}, \mathsf{P}, \mathsf{V})$, that are combined as follows:

- the CRS generator G (that, in particular, invokes $(\mathsf{ck}, \mathsf{td}_{\mathsf{C}}) \leftarrow \mathsf{G}_{\mathsf{com}}(1^\kappa, n)$) outputs $(\mathsf{crs} = (\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{G}(1^\kappa, n)$, where both crs_p and crs_v include ck, and td includes td_{C}.
- the prover P produces an argument π, $\pi \leftarrow \mathsf{P}(\mathsf{crs}_p; \boldsymbol{u}; \boldsymbol{w}, \boldsymbol{r})$, where presumably $u_i = \mathsf{C}_{\mathsf{ck}}(w_i; r_i)$.
- the verifier V, $\mathsf{V}(\mathsf{crs}_v; \boldsymbol{u}, \pi)$, outputs either 1 (accept) or 0 (reject).

Now, Π is *perfectly complete*, if for all $n = \mathrm{poly}(\kappa)$,

$$\Pr\left[(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\kappa, n), (\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{r}) \leftarrow \mathcal{R}_{\mathsf{ck},n} : \mathsf{V}(\mathsf{crs}_v; \boldsymbol{u}, \mathsf{P}(\mathsf{crs}_p; \boldsymbol{u}, \boldsymbol{w}, \boldsymbol{r})) = 1\right] = 1 \ .$$

Since Γ is computationally binding and trapdoor (and hence u_i can be commitments to *any* messages), soundness of the CaP argument systems only makes sense together with the argument of knowledge property. (Another possibility is to define and prove culpable soundness [29], but the argument of knowledge property is more standard.)

Let $b(X)$ be a non-negative polynomial. Π is *a (b-bounded-auxiliary-input [8]) argument of knowledge* for $\mathcal{R}$, if for all $n = \mathrm{poly}(\kappa)$ and every NUPPT A, there exists an NUPPT extractor X_{A}, such that for every auxiliary input $\mathsf{aux} \in \{0,1\}^{b(\kappa)}$,

$$\Pr\left[\begin{array}{l}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{G}(1^\kappa, n), ((\boldsymbol{u}, \pi); \boldsymbol{w}, \boldsymbol{r}) \leftarrow (\mathsf{A}||X_{\mathsf{A}})(\mathsf{crs}; \mathsf{aux}) : \\ (\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{r}) \notin \mathcal{R}_{\mathsf{ck},n} \wedge \mathsf{V}(\mathsf{crs}_v; \boldsymbol{u}, \pi) = 1\end{array}\right] \approx_\kappa 0 \ .$$

See [8] for a motivation behind bounded auxiliary input. As in the definition of PKE, we can restrict the definition of a argument of knowledge to benign auxiliary information generators, where aux is known to come from; we omit further discussion.

Π is *perfectly witness-indistinguishable*, if for all $n = \mathrm{poly}(\kappa)$, it holds that if $(\mathsf{crs}, \mathsf{td}) \in \mathsf{G}(1^\kappa, n)$ and $((\boldsymbol{u}; \boldsymbol{w}, \boldsymbol{r}), (\boldsymbol{u}; \boldsymbol{w}', \boldsymbol{r}')) \in \mathcal{R}^2_{\mathsf{ck},n}$ with $r_i, r_i' \leftarrow_r \mathfrak{R}$, then the distributions $\mathsf{P}(\mathsf{crs}_p; \boldsymbol{u}; \boldsymbol{w}, \boldsymbol{r})$ and $\mathsf{P}(\mathsf{crs}_p; \boldsymbol{u}; \boldsymbol{w}', \boldsymbol{r}')$ are equal. Note that a witness-indistinguishable argument system does not have to have a trapdoor.

Π is *perfectly composable zero-knowledge*, if there exists a probabilistic polytime simulator S, s.t. for all stateful NUPPT adversaries A and $n = \mathrm{poly}(\kappa)$,

$$\Pr\left[\begin{array}{l}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{G}(1^\kappa, n),\\ (\boldsymbol{u},\boldsymbol{w},\boldsymbol{r}) \leftarrow \mathsf{A}(\mathsf{crs}),\\ \pi \leftarrow \mathsf{P}(\mathsf{crs}_p;\boldsymbol{u};\boldsymbol{w},\boldsymbol{r}) :\\ (\boldsymbol{u},\boldsymbol{w},\boldsymbol{r}) \in \mathcal{R}_{\mathsf{ck},n} \wedge \mathsf{A}(\pi) = 1\end{array}\right] = \Pr\left[\begin{array}{l}(\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{G}(1^\kappa, n),\\ (\boldsymbol{u},\boldsymbol{w},\boldsymbol{r}) \leftarrow \mathsf{A}(\mathsf{crs}),\\ \pi \leftarrow \mathsf{S}(\mathsf{crs};\boldsymbol{u},\mathsf{td}) :\\ (\boldsymbol{u},\boldsymbol{w},\boldsymbol{r}) \in \mathcal{R}_{\mathsf{ck},n} \wedge \mathsf{A}(\pi) = 1\end{array}\right].$$

Here, the prover and the simulator use the same CRS, and thus we have *same-string zero knowledge* [22]. Same-string statistical zero knowledge allows to use the same CRS an unbounded number of times [22].

An argument system that satisfies above requirements is known as *adaptive*. An argument system where the CRS depends on the statement is often called *non-adaptive*. It is not surprising that non-adaptive SNARKs can be much more efficient than adaptive SNARKs; see [31, 32] for some examples.

A non-interactive argument system is *succinct* if the output length of P and the running time of V are polylogarithmic in the P's input length (and polynomial in the security parameter). A succinct non-interactive argument of knowledge is usually called *SNARK*. A zero-knowledge SNARK is abbreviated to *zk-SNARK*.

3 New Extractable Trapdoor Commitment Scheme

We now define a new extractable trapdoor commitment scheme. It uses the following polynomials. Assume n is a power of two, and let ω be the n-th primitive root of unity modulo p. Then,

- $Z(X) := \prod_{i=1}^{n}(X - \omega^{i-1}) = X^n - 1$ is the unique degree n monic polynomial, such that $Z(\omega^{i-1}) = 0$ for all $i \in [1\,..\,n]$.
- $\ell_i(X) := \prod_{j\neq i}((X-\omega^{j-1})/(\omega^{i-1}-\omega^{j-1}))$, the *ith Lagrange basis polynomial*, is the unique degree $n-1$ polynomial, such that $\ell_i(\omega^{i-1}) = 1$ and $\ell_i(\omega^{j-1}) = 0$ for $j \neq i$.

Clearly,

$$L_{\boldsymbol{a}}(X) = \sum_{i=1}^{n} a_i \ell_i(X)$$

is the interpolating polynomial of $\boldsymbol{a}$ at points ω^{i-1}, with $L_{\boldsymbol{a}}(\omega^{i-1}) = a_i$, and can thus be computed by executing an inverse Fast Fourier Transform. Moreover, $(\ell_i(\omega^{j-1}))_{j=1}^n = \boldsymbol{e_i}$ and $(Z(\omega^{j-1}))_{j=1}^n = \mathbf{0}_n$. Thus, $Z(X)$ and $(\ell_i(X))_{i=1}^n$ are $n+1$ linearly independent degree $\leq n$ polynomials, and hence

$$\mathcal{F}_{\mathsf{C}} := (Z(X), (\ell_i(X))_{i=1}^n)$$

is a basis of such polynomials. Clearly, $Z^{-1}(0) = \{j : Z(j) = 0\} = \{\omega^{i-1}\}_{i=1}^n$.

Definition 1 (Interpolating Commitment Scheme). *Let $n = \mathrm{poly}(\kappa)$, $n > 0$, be a power of two. First, $\mathsf{G}_{\mathsf{com}}(1^\kappa, n)$ sets $\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa, n)$, picks $g_1 \leftarrow_r \mathbb{G}_1^*$, $g_2 \leftarrow_r \mathbb{G}_2^*$, and then outputs the CRS*

$$\mathsf{ck} \leftarrow (\mathsf{gk}; (g_1^{f(\chi)}, g_2^{\gamma f(\chi)})_{f\in\mathcal{F}_{\mathsf{C}}})$$

for $\chi \leftarrow_r \mathbb{Z}_p \setminus Z^{-1}(0)$ *and* $\gamma \leftarrow_r \mathbb{Z}_p^*$. *The trapdoor is equal to* χ.

The commitment of $\boldsymbol{a} \in \mathbb{Z}_p^n$, *given a randomizer* $r \leftarrow_r \mathbb{Z}_p$, *is*

$$\mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r) := (g_1^{Z(\chi)}, g_2^{\gamma Z(\chi)})^r \cdot \prod_{i=1}^{n} (g_1^{\ell_i(\chi)}, g_2^{\gamma \ell_i(\chi)})^{a_i} \in \mathbb{G}_1 \times \mathbb{G}_2 \ , \ \textit{i.e.,}$$
$$\mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r) := (g_1, g_2^{\gamma})^{r(\chi^n - 1) + L_{\boldsymbol{a}}(\chi)} \ .$$

The validity of a commitment (A_1, A_2^{γ}) *is checked by verifying that* $\hat{e}(A_1, g_2^{\gamma Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, A_2^{\gamma})$. *To open a commitment, the committer sends* $(\boldsymbol{a}, r)$ *to the verifier.*

The condition $Z(\chi) \neq 0$ is needed in Thm. 1 to get perfect hiding and the trapdoor property. The condition $\gamma \neq 0$ is only needed in Thm. 5 to get perfect zero knowledge. Also, (a function of) γ is a part of the trapdoor in the range SNARK of Sect. 7.

Clearly, $\log_{g_1} A_1 = \log_{g_2^{\gamma}} A_2^{\gamma} = rZ(\chi) + \sum_{i=1}^{n} a_i \ell_i(\chi)$. The second element, A_2^{γ}, of the commitment is known as the knowledge component [20].

Theorem 1. *The interpolating commitment scheme is perfectly hiding and trapdoor. If* BP *is* n*-PDL secure, then it is computationally binding. If* BP *is* $(n, \emptyset, \emptyset)$*-PKE secure, then it is extractable.*

Proof. Perfect Hiding: since $Z(\chi) \neq 0$, then $rZ(\chi)$ (and thus also $\log_{g_1} A_1$) is uniformly random in $\mathbb{Z}_p$. Hence, (A_1, A_2^{γ}) is a uniformly random element of the multiplicative subgroup $\langle (g_1, g_2^{\gamma}) \rangle \subset \mathbb{G}_1^* \times \mathbb{G}_2^*$ generated by (g_1, g_2^{γ}), independently of the committed value. Extractability: clear from the statement.

Computational Binding: assume that there exists an adversary A_{C} that outputs $(\boldsymbol{a}, r_a)$ and $(\boldsymbol{b}, r_b)$ with $(\boldsymbol{a}, r_a) \neq (\boldsymbol{b}, r_b)$, s.t. the polynomial $d(X) := (r_a Z(X) + \sum_{i=1}^{n} a_i \ell_i(X)) - (r_b Z(X) + \sum_{i=1}^{n} b_i \ell_i(X))$ has a root at χ.

Construct now the following adversary A_{pdl} that breaks the PDL assumption. Given an n-PDL challenge, since $\mathcal{F}_{\mathsf{C}}$ consists of degree $\leq n$ polynomials, A_{pdl} can compute a valid ck from (a distribution that is statistically close to) the correct distribution. He sends ck to A_{C}. If A_{C} is successful, then $d(X) \in \mathbb{Z}_p[X]$ is a non-trivial degree-$\leq n$ polynomial. Since the coefficients of d are known, A_{pdl} can use an efficient polynomial factorization algorithm to compute all roots r_i of $d(X)$. One of these roots has to be equal to χ. A_{pdl} can establish which one by comparing each (say) $g_1^{\ell_1(r_i)}$ to the element $g_1^{\ell_1(\chi)}$ given in the CRS. Clearly, $g_1^{\ell_1(r_i)}$ is computed from g_1 (which can be computed, given the CRS, since $1 \in \mathrm{span}(\mathcal{F}_{\mathsf{C}})$), the coefficients of $\ell_1(X)$, and r_i. A_{pdl} has the same success probability as A_{C}, while her running time is dominated by that of A_{C} plus the time to factor a degree-$\leq n$ polynomial.

Trapdoor: given χ, $\boldsymbol{a}$, r, $\boldsymbol{a}^*$, and $c = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r)$, we compute r^* such that $(r^* - r)Z(\chi) + \sum_{i=1}^{n} (a_i^* - a_i)\ell_i(\chi) = 0$. This is possible since $Z(\chi) \neq 0$. Clearly, $c = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}^*; r^*)$. □

Thm. 1 also holds when instead of $Z(X)$ and $\ell_i(X)$ one uses any $n+1$ linearly independent low-degree polynomials (say) $P_0(X)$ and $P_i(X)$. Given the statement of Thm. 1, this choice of the concrete polynomials is very natural: $\ell_i(X)$ interpolate linearly independent vectors (and thus are linearly independent; in fact, they constitute a basis), and the choice to interpolate unit vectors is the conceptually clearest way of choosing $P_i(X)$. Another natural choice of independent polynomials is to set $P_i(X) = X^i$ as in [28], but that choice has resulted in much less efficient (CaP) SNARKs.

In Sect. 9 we show how to use batch-verification techniques to speed up simultaneous validity verification of many commitments.

4 New Product SNARK

Assume the use of the interpolating commitment scheme. In a *CaP product SNARK* [28], the prover aims to convince the verifier that she knows how to open three commitments (A, A^γ), (B, B^γ), and (C, C^γ) to vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ (together with the used randomizers), such that $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$. Thus,

$$\mathcal{R}^{\times}_{\mathsf{ck},n} := \left\{ \begin{array}{l} (u_\times, w_\times, r_\times) : u_\times = ((A_1, A_2^\gamma), (B_1, B_2^\gamma), (C_1, C_2^\gamma)) \wedge \\ w_\times = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \wedge r_\times = (r_a, r_b, r_c) \wedge (A_1, A_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r_a) \wedge \\ (B_1, B_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{b}; r_b) \wedge (C_1, C_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{c}; r_c) \wedge \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c} \end{array} \right\} .$$

Next, we propose an efficient CaP product SNARK. For this, we need Lem. 1.

Lemma 1. *Let $A(X)$, $B(X)$ and $C(X)$ be polynomials with $A(\omega^{i-1}) = a_i$, $B(\omega^{i-1}) = b_i$ and $C(\omega^{i-1}) = c_i$. Let $Q(X) = A(X)B(X) - C(X)$. Assume that (i) $A(X)$, $B(X)$ and $C(X)$ belong to the span of $(\ell_i(X))$, and (ii) there exists a degree $n-2$ polynomial $\pi(X)$, s.t. $\pi(X) = Q(X)/Z(X)$. Then $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$.*

Proof. From (i) it follows that $A(X) = L_{\boldsymbol{a}}(X)$, $B(X) = L_{\boldsymbol{b}}(X)$, and $C(X) = L_{\boldsymbol{c}}(X)$, and thus $Q(\omega^{i-1}) = a_i b_i - c_i$ for all $i \in [1\,..\,n]$. But (ii) iff $Z(X) \mid Q(X)$, which holds iff $Q(X)$ evaluates to 0 at all n values ω^{i-1}. Thus, $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$. Finally, if (i) holds then $\deg Q(X) = 2n - 2$ and thus $\deg \pi(X) = n - 2$. □

If privacy and succinctness are not needed, one can think of the product argument being equal to $\pi(X)$. We achieve privacy by picking $r_a, r_b, r_c \leftarrow_r \mathbb{Z}_p$, and defining $Q_{wi}(X) := (L_{\boldsymbol{a}}(X) + r_a Z(X))\,(L_{\boldsymbol{b}}(X) + r_b Z(X)) - (L_{\boldsymbol{c}}(X) + r_c Z(X))$. Here, the new addends of type $r_a Z(X)$ guarantee hiding. On the other hand, $Q_{wi}(X)$ remains divisible by $Z(X)$ iff $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$. Thus, $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$ iff

(i') $Q_{wi}(X)$ can be expressed as $Q_{wi}(X) = A(X)B(X) - C(X)$ for some polynomials $A(X)$, $B(X)$ and $C(X)$ that belong to the span of $\mathcal{F}_{\mathsf{C}}$, and

(ii') there exists a polynomial $\pi_{wi}(X)$, such that

$$\pi_{wi}(X) = Q_{wi}(X)/Z(X) \ . \tag{1}$$

The degree of $Q_{wi}(X)$ is $2n$, thus, if $\pi_{wi}(X)$ exists, then it has degree n.

However, $|\pi_{wi}(X)|$ is not sublinear in n. To minimize communication, we let the prover transfer a "garbled" evaluation of $\pi_{wi}(X)$ at a random secret point χ. More precisely, the prover computes $\pi_\times := g_1^{\pi_{wi}(\chi)}$, using the values $g_1^{\chi^i}$ (given in the CRS) and the coefficients π_i of $\pi_{wi}(X) = \sum_{i=0}^{n} \pi_i X^i$, as follows:

$$\pi_\times := g_1^{\pi_{wi}(\chi)} \leftarrow \prod_{i=0}^{n} (g_1^{\chi^i})^{\pi_i} \ . \tag{2}$$

Similarly, instead of (say) $L_{\boldsymbol{a}}(X) + r_a Z(X)$, the verifier has the succinct interpolating commitment $\mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r_a) = (g_1, g_2^\gamma)^{L_{\boldsymbol{a}}(\chi) + r_a Z(\chi)}$ of $\boldsymbol{a}$.

We now give a full description of the new product SNARK $\Pi_\times$, given the interpolating commitment scheme $(\mathsf{G}_{\mathsf{com}}, \mathsf{C})$ and the following tuple of algorithms, $(\mathsf{G}_\times, \mathsf{P}_\times, \mathsf{V}_\times)$. Note that $\mathsf{C}_{\mathsf{ck}}(\mathbf{1}_{\boldsymbol{n}}; 0) = (g_1, g_2^\gamma)$.

CRS generation $\mathsf{G}_\times(1^\kappa, n)$**:** Let $\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa)$, $(g_1, g_2, \chi, \gamma) \leftarrow_r \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p^2$ with $Z(\chi) \neq 0$ and $\gamma \neq 0$. Let $\mathsf{crs}_p = \mathsf{ck} \leftarrow (\mathsf{gk}; (g_1, g_2^\gamma)^{\mathcal{F}_{\mathsf{C}}(\chi)})$ and $\mathsf{crs}_v \leftarrow (\mathsf{gk}; g_2^{\gamma Z(\chi)})$. Output $\mathsf{crs}_\times = (\mathsf{crs}_p, \mathsf{crs}_v)$.

Common input: $u_\times = ((A_1, A_2^\gamma), (B_1, B_2^\gamma), (C_1, C_2^\gamma))$.

Proving $\mathsf{P}_\times(\mathsf{crs}_p; u_\times; w_\times = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}), r_\times = (r_a, r_b, r_c))$**:** Compute $\pi_{wi}(X) = \sum_{i=0}^{n} \pi_i X^i$ as in Eq. (1) and $\pi_\times$ as in Eq. (2). Output $\pi_\times$.

Verification $\mathsf{V}_\times(\mathsf{crs}_v; u_\times; \pi_\times)$**:** accept if $\hat{e}(A_1, B_2^\gamma) = \hat{e}(g_1, C_2^\gamma) \cdot \hat{e}(\pi_\times, g_2^{\gamma Z(\chi)})$.

Since one can recompute it from ck, inclusion of $g_2^{\gamma Z(\chi)}$ in the CRS is only needed to speed up the verification. Here as in the shift SNARK of Sect. 5, validity of the commitments will be verified in the master SNARK. This is since the master SNARKs use some of the commitments in several sub-SNARKs, while it suffices to verify the validity of every commitment only once.

To obtain an argument of knowledge, we use knowledge assumptions in all following proofs. This SNARK is not zero-knowledge since the possible simulator gets three commitments as inputs but not their openings; to create an accepting argument the simulator must at least know how to open the commitment $(A_1 B_1 / C_1, A_2^\gamma B_2^\gamma / C_2^\gamma)$ to $\boldsymbol{a} \circ \boldsymbol{b} - \boldsymbol{c}$. It is witness-indistinguishable, and this suffices for the Subset-Sum and other master SNARKs to be zero-knowledge.

Theorem 2. *$\Pi_\times$ is perfectly complete and witness-indistinguishable. If the input consists of valid commitments, and* BP *is n-TSDH and $(n, \emptyset, \emptyset)$-PKE secure, then $\Pi_\times$ is an ($\Theta(n)$-bounded-auxiliary-input) adaptive argument of knowledge.*

Proof. Perfect completeness: follows from the discussion in the beginning of this section. Perfect witness-indistinguishability: since the argument $\pi_\times$ that satisfies the verification equations is unique, all witnesses result in the same argument, and thus this argument is witness-indistinguishable.

Argument of knowledge: Assume that A_{aok} is an adversary that, given $\mathsf{crs}_\times$, returns $(u_\times, \pi)$ such that $\mathsf{V}_\times(\mathsf{crs}_v; u_\times, \pi) = 1$. Assume that the PKE assumption holds, and let X_{A} be the extractor thatreturns openings of the commitments in $u_\times$, i.e., $(\boldsymbol{a}, r_a)$, $(\boldsymbol{b}, r_b)$, and $(\boldsymbol{c}, r_c)$. We now claim that X_{A} is also the extractor needed to achieve the argument of knowledge property.

Assume that this is not the case. We now construct the next adversary A_{tsdh} against n-TSDH. Given an n-TSDH challenge $ch = (\mathsf{gk}, ((g_1, g_2)^{\chi^i})_{i=0}^n)$, A_{tsdh} first generates $\gamma \leftarrow_r \mathbb{Z}_p^*$, and then computes (this is possible since $\mathcal{F}_{\mathsf{C}}$ consists of degree $\leq n$ polynomials) and sends $\mathsf{crs}_\times$ to A_{aok}. Assume $(\mathsf{A}_{aok}||X_{\mathsf{A}})(\mathsf{crs}_\times)$ returns $((u_\times = ((A_1, A_2^\gamma), (B_1, B_2^\gamma), (C_1, C_2^\gamma)), \pi), (w_\times = (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}), r_\times = (r_a, r_b, r_C)))$, s.t. $u_i = \mathsf{C}_{\mathsf{ck}}(w_i; r_i)$ but $(u_\times, w_\times, r_\times) \notin \mathcal{R}_{\mathsf{ck},n}^\times$. Since the openings are correct, $\boldsymbol{a} \circ \boldsymbol{b} \neq \boldsymbol{c}$ but π is accepting. According to Lem. 1, thus $Z(X) \nmid Q_{wi}(X)$.

Since $Z(X) \nmid Q_{wi}(X)$, then for some $i \in [1\,..\,n]$, $(X - \omega^{i-1}) \nmid Q_{wi}(X)$. Write $Q_{wi}(X) = q(X)(X - \omega^{i-1}) + r$ for $r \in \mathbb{Z}_p^*$. Clearly, $\deg q(X) \leq 2n-1$. Moreover, we write $q(X) = q_1(X)Z(X) + q_2(X)$ with $\deg q_i(X) \leq n-1$. Since the verification succeeds, $\hat{e}(g_1, g_2^\gamma)^{Q_{wi}(\chi)} = \hat{e}(\pi_\times, g_2^{\gamma Z(\chi)})$, or $\hat{e}(g_1, g_2^\gamma)^{q(\chi)(\chi - \omega^{i-1}) + r} = \hat{e}(\pi_\times, g_2^{\gamma Z(\chi)})$, or $\hat{e}(g_1, g_2^\gamma)^{q(\chi) + r/(\chi - \omega^{i-1})} = \hat{e}(\pi_\times, g_2^{\gamma Z(\chi)/(\chi - \omega^{i-1})})$, or

$$\hat{e}(g_1, g_2^\gamma)^{1/(\chi - \omega^{i-1})} = (\hat{e}(\pi_\times, g_2^{\gamma Z(\chi)/(\chi - \omega^{i-1})}) / \hat{e}(g_1^{q(\chi)}, g_2^\gamma))^{r^{-1}} \quad .$$

Now, $\hat{e}(g_1^{q(\chi)}, g_2^\gamma) = \hat{e}(g_1^{q_1(\chi)}, g_2^{\gamma Z(\chi)}) \hat{e}(g_1^{q_2(\chi)}, g_2^\gamma)$, and thus it can be efficiently computed from $((g_1^{\chi^i})_{i=0}^{n-1}, g_2^\gamma, g_2^{\gamma Z(\chi)}) \subset \mathsf{crs}$. Moreover, $Z(X)/(X - \omega^{i-1}) = \ell_i(X) \cdot \prod_{j \neq i} (\omega^{i-1} - \omega^{j-1})$, and thus $g_2^{\gamma Z(\chi)/(\chi - \omega^{i-1})}$ can be computed from $g_2^{\gamma \ell_i(\chi)}$ by using generic group operations. Hence, $\hat{e}(g_1, g_2^\gamma)^{1/(\chi - \omega^{i-1})}$ can be computed from $((g_1^{\chi^i})_{i=0}^{n-1}, g_2^\gamma, g_2^{\gamma Z(\chi)}, (g_2^{\gamma \ell_i(\chi)})_{i=1}^n)$ (that can be computed from ch), by using generic group operations. Thus, the adversary has computed $(r = \omega^{i-1}, \hat{e}(g_1, g_2^\gamma)^{1/(\chi - r)})$, for $r \neq \chi$. Since A_{tsdh} knows $\gamma \neq 0$, he can finally compute $(r, \hat{e}(g_1, g_2)^{1/(\chi - r)})$, and thus break the n-TSDH assumption.

Hence, the argument of knowledge property follows. □

We remark that the product SNARK (but not the shift SNARK of Sect. 5) can be seen as a QAP-based SNARK [26], namely for the relation $\boldsymbol{a} \circ \boldsymbol{b} - \boldsymbol{c}$. (Constructing a QAP-based shift SNARK is possible, but results in using different polynomials and thus in a different commitment scheme.)

Efficiency. The prover computation is dominated by the computation of

(i) one $(n+1)$-wide multi-exponentiation in $\mathbb{G}_1$. By using the Pippenger's multi-exponentiation algorithm for *large* n this means approximately $n+1$ bilinear-group multiplications, see [42]. For small values of n, one can use the algorithm by Straus [46]; then one has to execute $\Theta(n/\log n)$ bilinear-group exponentiations.

(ii) three polynomial interpolations, one polynomial multiplication, and one polynomial division to compute the coefficients of the polynomial $\pi_{wi}(X)$. Since polynomial division can be implemented as 2 polynomial multiplications (by using pre-computation and storing some extra information in the CRS, [25, 37]), this part is dominated by two inverse FFT-s and three polynomial multiplications. Other savings are possible.

The verifier computation is dominated by 3 pairings. (We will count the cost of validity verifications separately in the master SNARKs.) In the special case $C_1 = A_1$ (e.g., in the *Boolean SNARK*, where we need to prove that $\boldsymbol{a} \circ \boldsymbol{a} = \boldsymbol{a}$,

or in the *restriction SNARK* [28], where we need to prove that $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{a}$ for a *public* Boolean vector $\boldsymbol{b}$), the verification equation can be simplified to $\hat{e}(A_1, B_2^\gamma / g_2^\gamma) = \hat{e}(\pi_\times, g_2^{\gamma Z(\chi)})$, which saves one more pairing. In Sect. 9, we will describe a batch-verification technique that allows to speed up simultaneous verification of several product SNARKs.

Excluding gk, the prover CRS together with ck consists of $2(n+1)$ group elements, while the verifier CRS consists of 1 group element. The CRS can be computed in time $\Theta(n)$, by using an algorithm from [5].

5 New Shift SNARK

In a *shift-right-by-z* SNARK [23] (shift SNARK, for short), the prover aims to convince the verifier that for 2 commitments (A, A^γ) and (B, B^γ), he knows how to open them as $(A, A^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r_a)$ and $(B, B^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{b}; r_b)$, s.t. $\boldsymbol{a} = \boldsymbol{b} \gg z$. I.e., $a_i = b_{i+z}$ for $i \in [1\,..\,n-z]$ and $a_i = 0$ for $i \in [n-z+1\,..\,n]$. Thus,

$$\mathcal{R}^{\mathsf{rsft}}_{\mathsf{ck},n} := \left\{ \begin{array}{l} (u_\times, w_\times, r_\times) : u_\times = ((A_1, A_2^\gamma), (B_1, B_2^\gamma)) \wedge w_\times = (\boldsymbol{a}, \boldsymbol{b}) \wedge \\ r_\times = (r_a, r_b) \wedge (A_1, A_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r_a) \wedge \\ (B_1, B_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{b}; r_b) \wedge (\boldsymbol{a} = \boldsymbol{b} \gg z) \end{array} \right\} .$$

An efficient shift SNARK was described in [23]. We now reconstruct this SNARK so that it can be used together with the interpolating commitment scheme. We can do it since the shift SNARK of [23] does *almost* not depend on the commitment scheme. We also slightly optimize the resulting SNARK; in particular, the verifier has to execute one less pairing compared to [23].

Our strategy of constructing a shift SNARK follows the strategy of [28, 36]. We start with a concrete verification equation that also contains the argument π_1. We write the discrete logarithm of π_1 (that follows from this equation) as $F_\pi(\chi) + F_{con}(\chi)$, where χ is a secret key, and $F_\pi(X)$ and $F_{con}(X)$ are two polynomials. The first polynomial, $F_\pi(X)$, is identically zero iff the prover is honest. Since the spans of certain two polynomial sets do not intersect, this results in an efficient adaptive shift SNARK that is an argument of knowledge under (two) PKE assumptions.

Now, for a non-zero polynomial $Z^*(X)$ to be defined later, consider the verification equation $\hat{e}(A_1, g_2^{\gamma Z^*(\chi)}) / \hat{e}(B_1 \pi_1, g_2^\gamma) = 1$ (due to the properties of pairing, this is equivalent to verifying that $\pi_1 = A_1^{Z^*(\chi)} / B_1$), with (A_1, A_2^γ) and (B_1, B_2^γ) being interpolating commitments to $\boldsymbol{a}$ and $\boldsymbol{b}$, and $\pi_1 = g_1^{\pi(\chi)}$ for some polynomial $\pi(X)$. Denote $r(X) := (r_a Z^*(X) - r_b) Z(X)$. Taking a discrete logarithm of the verification equation, we get that

$$\begin{aligned} \pi(X) &= (r_a Z(X) + \textstyle\sum_{i=1}^n a_i \ell_i(X))\, Z^*(X) - (r_b Z(X) + \sum_{i=1}^n b_i \ell_i(X)) \\ &= Z^*(X) \textstyle\sum_{i=1}^n a_i \ell_i(X) - \sum_{i=1}^n b_i \ell_i(X) + r(X) \\ &= \left(\textstyle\sum_{i=1}^{n-z} a_i \ell_i(X) + \sum_{i=n-z+1}^{n} a_i \ell_i(X)\right) Z^*(X) + r(X) - \\ &\qquad \left(\textstyle\sum_{i=1}^{n-z} b_{i+z} \ell_{i+z}(X) + \sum_{i=1}^{z} b_i \ell_i(X)\right) . \end{aligned}$$

Hence, $\pi(X) = F_\pi(X) + F_{con}(X)$, where

$$F_\pi(X) = \left(\sum_{i=1}^{n-z}(a_i - b_{i+z})\ell_i(X) + \sum_{i=n-z+1}^{n} a_i\ell_i(X)\right) \cdot Z^*(X) \ ,$$
$$F_{con}(X) = \left(\sum_{i=z+1}^{n} b_i(\ell_{i-z}(X)Z^*(X) - \ell_i(X)) - \sum_{i=1}^{z} b_i\ell_i(X)\right) + r(X) \ .$$

Clearly, the prover is honest iff $F_\pi(X) = 0$, which holds iff $\pi(X) = F_{con}(X)$, i.e., $\pi(X)$ belongs to the span of

$$\mathcal{F}_{z-\mathsf{rsft}} := (\ell_{i-z}(X)Z^*(X) - \ell_i(X))_{i=z+1}^{n}, (\ell_i(X))_{i=1}^{z}, Z(X)Z^*(X), Z(X)) \ .$$

For the shift SNARK to be an argument of knowledge, we need that
(i) $(\ell_i(X)Z^*(X))_{i=1}^n$ is linearly independent, and
(ii) $F_\pi(X) \cap \mathrm{span}(\mathcal{F}_{z-\mathsf{rsft}}) = \emptyset$.
Together, (i) and (ii) guarantee that from $\pi(X) \in \mathrm{span}(\mathcal{F}_{z-\mathsf{rsft}})$ it follows that $\boldsymbol{a}$ is a shift of $\boldsymbol{b}$.

We guarantee that $\pi(X) \in \mathrm{span}(\mathcal{F}_{z-\mathsf{rsft}})$ by a knowledge assumption (w.r.t. another knowledge secret δ); for this we will also show that $\mathcal{F}_{z-\mathsf{rsft}}$ is linearly independent. As in the case of the product SNARK, we also need that (A_1, A_2^γ) and (B_1, B_2^γ) are actually commitments of n-dimensional vectors (w.r.t. γ), i.e., we rely on two PKE assumptions.

Denote $\mathcal{F}_\pi := \{\ell_i(X)Z^*(X)\}_{i=1}^n$. For a certain choice of $Z^*(X)$, both (i) and (ii) follow from the next lemma.

Lemma 2. *Let $Z^*(X) = Z(X)^2$. Then $\mathcal{F}_\pi \cup \mathcal{F}_{z-\mathsf{rsft}}$ is linearly independent.*

Proof. Assume that there exist $\boldsymbol{a} \in \mathbb{Z}_p^n$, $\boldsymbol{b} \in \mathbb{Z}_p^n$, $c \in \mathbb{Z}_p$, and $d \in \mathbb{Z}_p$, s.t. $f(X) := \sum_{i=1}^n a_i\ell_i(X)Z^*(X) + \sum_{i=z+1}^n b_i\left(\ell_{i-z}(X)Z^*(X) - \ell_i(X)\right) - \sum_{i=1}^z b_i\ell_i(X) + cZ(X)Z^*(X) + dZ(X) = 0$. But then also $f(\omega^{j-1}) = 0$, for $j \in [1\,..\,n]$. Thus, due to the definition of $\ell_i(X)$ and $Z(X)$, $\sum_{i=1}^z b_i\boldsymbol{e_i} = \mathbf{0}_n$ which is only possible if $b_i = 0$ for all $j \in [1\,..\,n]$. Thus also $f'(X) := f(X)/Z(X) = \sum_{i=1}^n a_i\ell_i(X)Z^*(X)/Z(X) + cZ^*(X) + d = 0$. But then also $f'(\omega^{j-1}) = 0$ for $j \in [1\,..\,n]$. Hence, $cZ^*(\omega^{j-1}) + d = d = 0$. Finally, $f''(X) := f(X)/Z^*(X) = \sum_{i=1}^n a_i\ell_i(X) + cZ(X) = 0$, and from $f''(\omega^{j-1}) = 0$ for $j \in [1\,..\,n]$, we get $\boldsymbol{a} = \mathbf{0}_n$. Thus also $c = 0$. This finishes the proof. □

Since the argument of knowledge property of the new shift SNARK relies on $\pi(X)$ belonging to a certain span, similarly to [23], we will use an additional knowledge assumption. That is, it is necessary that there exists an extractor that outputs a witness that $\pi(X) = F_{con}(X)$ belongs to the span of $\mathcal{F}_{z-\mathsf{rsft}}$.

Similarly to the product SNARK, the shift SNARK does not contain $\pi(X) = F_{con}(X)$, but the value $\pi_{\mathsf{rsft}} = (g_1, g_2^\delta)^{\pi(\chi)}$ for random χ and δ (necessary due to the use of the second PKE assumption), computed as

$$\begin{aligned}\pi_{\mathsf{rsft}} \leftarrow & (\pi_1, \pi_2^\delta) = (g_1, g_2^\delta)^{\pi(\chi)} \\ & = \textstyle\prod_{i=z+1}^n ((g_1, g_2^\delta)^{\ell_{i-z}(\chi)Z^*(\chi) - \ell_i(\chi)})^{b_i} \cdot \prod_{i=1}^z ((g_1, g_2^\delta)^{\ell_i(\chi)})^{-b_i} \cdot \\ & \quad ((g_1, g_2^\delta)^{Z(\chi)Z^*(\chi)})^{r_a} \cdot ((g_1, g_2^\delta)^{Z(\chi)})^{-r_b} \ .\end{aligned} \tag{3}$$

We are now ready to state the new shift-right-by-z SNARK Π_{rsft}. It consists of the interpolating commitment scheme and of the following three algorithms:

CRS generation $\mathsf{G}_{\mathsf{rsft}}(1^\kappa, n)$**:** Let $Z^*(X) = Z(X)^2$. Let $\mathsf{gk} \leftarrow \mathsf{BP}(1^\kappa)$, $(g_1, g_2, \chi, \gamma, \delta) \leftarrow \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p^3$, s.t. $Z(\chi) \neq 0$, $\gamma \neq 0$. Set $\mathsf{ck} \leftarrow (\mathsf{gk}; (g_1, g_2^\gamma)^{\mathcal{F}_{\mathsf{C}}(\chi)})$, $\mathsf{crs}_p \leftarrow (\mathsf{gk}; (g_1, g_2^\delta)^{\mathcal{F}_{z-\mathsf{rsft}}(\chi)})$, $\mathsf{crs}_v \leftarrow (\mathsf{gk}; (g_1, g_2^\delta)^{Z(\chi)}, g_2^{\delta Z(\chi) Z^*(\chi)})$. Return $\mathsf{crs}_{\mathsf{rsft}} = (\mathsf{ck}, \mathsf{crs}_p, \mathsf{crs}_v)$.

Common input: $u_{\mathsf{rsft}} = ((A_1, A_2^\gamma), (B_1, B_2^\gamma))$.

Proving $\mathsf{P}_{\mathsf{rsft}}(\mathsf{crs}_p; u_{\mathsf{rsft}}; w_{\mathsf{rsft}} = (\boldsymbol{a}, \boldsymbol{b}), r_{\mathsf{rsft}} = (r_a, r_b))$**:** return $\pi_{\mathsf{rsft}} \leftarrow (\pi_1, \pi_2^\delta)$ from Eq. (3).

Verification $\mathsf{V}_{\mathsf{rsft}}(\mathsf{crs}_v; u_{\mathsf{rsft}}; \pi_{\mathsf{rsft}} = (\pi_1, \pi_2^\delta))$**:** accept if $\hat{e}(\pi_1, g_2^{\delta Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, \pi_2^\delta)$ and $\hat{e}(B_1 \pi_1, g_2^{\delta Z(\chi)}) = \hat{e}(A_1, g_2^{\delta Z(\chi) Z^*(\chi)})$.

Since crs_v can be recomputed from $\mathsf{ck} \cup \mathsf{crs}_p$, then clearly it suffices to take CRS to be $\mathsf{crs}_{\mathsf{rsft}} = (\mathsf{gk}; g_1^{\mathcal{F}_{\mathsf{C}}(\chi) \cup \mathcal{F}_{z-\mathsf{rsft}}(\chi)}, g_2^{\gamma \mathcal{F}_{\mathsf{C}}(\chi) \cup \delta \mathcal{F}_{z-\mathsf{rsft}}(\chi)})$.

Theorem 3. *Let $Z^*(X) = Z(X)^2$, $y = \deg(Z(X)Z^*(X)) = 3n$. Π_{rsft} is perfectly complete and witness-indistinguishable. If the input consists of valid commitments, and BP is y-PDL, $(n, \mathcal{F}_{z-\mathsf{rsft}}, Y_2 \mathcal{F}_{z-\mathsf{rsft}}, 1)$-PKE, and $(\mathcal{F}_{z-\mathsf{rsft}}, \mathcal{F}_{\mathsf{C}}, Y_1 \mathcal{F}_{\mathsf{C}}, 2)$-PKE secure, then Π_{rsft} is an ($\Theta(n)$-bounded-auxiliary-input) adaptive argument of knowledge.*

Proof. Perfect completeness: follows from the derivation of the SNARK. Note that we added extra $Z(\chi)$ to exponents so that the verification equation be computable from $\mathsf{crs}_{\mathsf{rsft}}$. Perfect witness-indistinguishability: since argument π_{rsft} that satisfies the verification equations is unique, all witnesses result in the same argument, and hence the shift SNARK is witness-indistinguishable.

Argument of knowledge: Assume that A_{aok} is an adversary sucuh that $\mathsf{A}_{aok}(\mathsf{crs}_{\mathsf{rsft}})$ outputs, with all but a negligible probability, the input $u_{\mathsf{rsft}} = ((A_1, A_2^\gamma), (B_1, B_2^\gamma))$ and an accepting argument $\pi_{\mathsf{rsft}} \leftarrow (\pi_1, \pi_2^\delta)$, s.t. $\boldsymbol{a} \neq \boldsymbol{b} \gg z$. Observe that A_{aok} gets no auxiliary input.

We now construct the following adversary A_{pdl} that breaks PDL. It gets as an input an y-PDL challenge $ch = (\mathsf{gk}, ((g_1, g_2)^{\chi^i})_{i=0}^y)$. He chooses random $\gamma \neq 0$, δ, and then computes the CRS $\mathsf{crs}_{\mathsf{rsft}}$ of the shift SNARK (this is possible since all involved polynomials are of degree $\leq \deg(Z(X)Z^*(X)) = y$). Now,

(a) from the $(n, \mathcal{F}_{z-\mathsf{rsft}}, Y_2 \mathcal{F}_{z-\mathsf{rsft}}, 1)$-PKE assumption it follows that there exists an extractor that, given access to A_{aok}'s secret coins and input $\mathsf{crs}_{\mathsf{rsft}}$, returns $(w_{\mathsf{rsft}} = (\boldsymbol{a}, \boldsymbol{b}), r_{\mathsf{rsft}} = (r_a, r_b))$ such that $(A_1, A_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{a}; r_a)$, and $(B_1, B_2^\gamma) = \mathsf{C}_{\mathsf{ck}}(\boldsymbol{b}; r_b)$, and

(b) from the $(\mathcal{F}_{z-\mathsf{rsft}}, \mathcal{F}_{\mathsf{C}}, Y_1 \mathcal{F}_{\mathsf{C}}, 2)$-PKE assumption it follows that there exists an extractor that, given access to A_{aok}'s secret coins and input $\mathsf{crs}_{\mathsf{rsft}}$, returns a polynomial $\pi^*(X)$, such that $\pi_{\mathsf{rsft}} = (\pi_1, \pi_2^\delta) = (g_1, g_2^\delta)^{\pi^*(\chi)}$ where $\pi^*(X) = \sum_{f \in \mathcal{F}_{z-\mathsf{rsft}}} \pi_f^* \cdot f(X)$.

Then, given A_{aok} and those two extractors, we can construct another adversary A_{aok}^* that, given access to A_{aok}'s secret coins and input, can — with all but a negligible probability — output u_{rsft}, an accepting statement π_{rsft}, and an extended witness $w_{\mathsf{rsft}}^* = (\boldsymbol{a}, \boldsymbol{b}, r_a, r_b, \pi^*(X))$.

Thus, A_{pdl} forwards $\mathsf{crs}_{\mathsf{rsft}}$ to A^*_{aok}, who returns u_{rsft}, an accepting argument π_{rsft}, and w^*_{rsft}. Thus, A_{pdl} has obtained all coefficients of $\pi^*(X)$, such that $\pi_1 = g_1^{\pi^*(\chi)}$. Since $\hat{e}(B_1\pi_1, g_2^{\delta Z(\chi)}) = \hat{e}(A_1, g_2^{\delta Z(\chi)Z^*(\chi)})$, $\pi^*(\chi) = F_\pi(\chi) + F_{con}(\chi)$. Since $\pi^*(X) \in \mathrm{span}(\mathcal{F}_{z-\mathsf{rsft}})$, we have by preceding discussion that $\pi^*(X) = F_{con}(X)$ and thus $\pi^*(\chi) = F_{con}(\chi)$ and hence $F_\pi(\chi) = 0$. Since A_{aok} succeeded in cheating, $F_\pi(X)$ is a non-zero polynomial with $F_\pi(\chi) = 0$. But A_{pdl} can use an efficient polynomial factorization algorithm to find all roots r_i of $F_\pi(X)$. One of those roots has to be χ; this can be tested by comparing say the values $g_1^{Z(r_i)Z^*(\chi)}$ with the value $g_1^{Z(\chi)Z^*(\chi)}$ given in the CRS. The claim follows. □

The prover computation is dominated by two $(n+2)$-wide multi-exponentiations (one in $\mathbb{G}_1$ and one in $\mathbb{G}_2$); there is no need for polynomial interpolation, multiplication or division. The communication is 2 group elements. The verifier computation is dominated by 4 pairings. In Sect. 9, we will describe a batch-verification technique that allows to speed up simultaneous verification of several shift SNARKs. Apart from gk, the prover CRS and the ck together contain $4n+6$ group elements, and the verifier CRS contains 3 group elements.

A shift-left-by-z (necessary in [38] to construct a permutation SNARK) SNARK can be constructed similarly. A rotation-left/right-by-z SNARK (one committed vector is a *rotation* of another committed vector) requires only small modifications, see [23])

6 New Subset-Sum SNARK

For fixed n and $p = n^{\omega(1)}$, the NP-complete language Subset-Sum over $\mathbb{Z}_p$ is defined as the language $\mathcal{L}_n^{\text{Subset-Sum}}$ of tuples $(\boldsymbol{S} = (S_1, \ldots, S_n), s)$, with $S_i, s \in \mathbb{Z}_p$, such that there exists a vector $\boldsymbol{b} \in \{0,1\}^n$ with $\sum_{i=1}^n S_i b_i = s$ in $\mathbb{Z}_p$. Subset-Sum can be solved in pseudo-polynomial time $O(pn)$ by using dynamic programming [43]; however, for $p = n^{\omega(1)}$. In the current paper, since $n = \kappa^{o(1)}$ and $p = 2^{O(\kappa)}$), pn is not polynomial in the input size $n \log_2 p$.

In a Subset-Sum SNARK, the prover aims to convince the verifier that he knows how to open commitment (B_1, B_2^γ) to a vector $\boldsymbol{b} \in \{0,1\}^n$, such that $\sum_{i=1}^n S_i b_i = s$. Next, we show that by using the new product and shift SNARKs, one can design a prover-efficient adaptive Subset-Sum zk-SNARK $\mathit{\Pi}_{\mathsf{ssum}}$. We emphasize that Subset-Sum is just one of the languages for which we can construct an efficient zk-SNARK; Sect. 7 and Sect. 8 have more examples.

First, we again use the interpolating commitment scheme. The CRS generation $\mathsf{G}_{\mathsf{ssum}}$ invokes CRS generations of the commitment scheme, the product SNARK and the shift SNARK, sharing the same gk, g_1, g_2, γ, and trapdoor $\mathsf{td} = \chi$ between the different invocations. (Since here the argument must be zero knowledge, it needs a trapdoor.) Thus, $\mathsf{crs}_{\mathsf{ssum}} = \mathsf{crs}_{\mathsf{rsft}}$ for $z = 1$.

The prover's actions are depicted by Fig. 1 (a precise explanation of this SNARK will be given in the concise completeness proof in Thm. 4). This SNARK, even without taking into account the differences in the product and shift SNARKs, is both simpler and moth efficient than the Subset-Sum SNARK presented in [23] where one say needed an additional step of proving that $\boldsymbol{b} \neq \boldsymbol{0}_n$.

Let $\boldsymbol{b} \in \{0,1\}^n$ be such that $\sum_{i=1}^n S_i b_i = s$.
Let (B_1, B_2^γ) be a commitment to $\boldsymbol{b}$.
Construct an argument π_1 to show that $\boldsymbol{b} \circ \boldsymbol{b} = \boldsymbol{b}$.
Let (C_1, C_2^γ) be a commitment to $\boldsymbol{c} \leftarrow \boldsymbol{S} \circ \boldsymbol{b}$.
Construct an argument π_2 to show that $\boldsymbol{c} = \boldsymbol{S} \circ \boldsymbol{b}$.
Let (D_1, D_2^γ) be a commitment to $\boldsymbol{d}$, where $d_i = \sum_{j \geq i} c_j$.
Construct an argument $(\pi_{31}, \pi_{32}^\delta)$ to show that $\boldsymbol{d} = (\boldsymbol{d} - \boldsymbol{c}) \gg 1$.
Construct an argument π_4 to show that $\boldsymbol{e_1} \circ (\boldsymbol{d} - s\boldsymbol{e_1}) = \boldsymbol{0}_n$.
Output $\pi_{\mathsf{ssum}} = (B_1, B_2^\gamma, C_1, C_2^\gamma, D_1, D_2^\gamma, \pi_1, \pi_2, \pi_{31}, \pi_{32}^\delta, \pi_4)$.

Fig. 1. The new SUBSET-SUM SNARK Π_{ssum} (prover's operations)

We remark that the vector $\boldsymbol{d}$, with $d_i = \sum_{j \geq i} c_j$, is called either a *vector scan*, an *all-prefix-sums*, or a *prefix-sum* of $\boldsymbol{c}$ [11], and $(\pi_{31}, \pi_{32}^\delta)$ can be thought of as a *scan SNARK* [23] that $\boldsymbol{d}$ is a correct scan of $\boldsymbol{c}$.

After receiving π_{ssum}, the verifier computes $S_1' \leftarrow \prod_i (g_1^{\ell_i(\chi)})^{S_i}$ as the first half of a commitment to $\boldsymbol{S}$, and then performs the following verifications:

- Three commitment validations: $\hat{e}(B_1, g_2^\gamma) = \hat{e}(g_1, B_2^\gamma)$, $\hat{e}(C_1, g_2^\gamma) = \hat{e}(g_1, C_2^\gamma)$, $\hat{e}(D_1, g_2^\gamma) = \hat{e}(g_1, D_2^\gamma)$.
- Three product argument verifications: $\hat{e}(B_1/g_1, B_2^\gamma) = \hat{e}(\pi_1, g_2^{\gamma Z(\chi)})$, $\hat{e}(S_1', B_2^\gamma) = \hat{e}(g_1, C_2^\gamma) \cdot \hat{e}(\pi_2, g_2^{\gamma Z(\chi)})$, $\hat{e}(g_1^{\ell_1(\chi)}, D_2^\gamma / (g_2^{\gamma \ell_1(\chi)})^s) = \hat{e}(\pi_4, g_2^{\gamma Z(\chi)})$.
- One shift argument verification, consisting of two equality tests: $\hat{e}(\pi_{31}, g_2^{\delta Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, \pi_{32}^\delta)$, $\hat{e}(D_1 / C_1 \pi_{31}, g_2^{\delta Z(\chi)}) = \hat{e}(D_1, g_2^{\delta Z(\chi) Z^*(\chi)})$.

Theorem 4. *Π_{ssum} is perfectly complete and perfectly composable zero-knowledge. It is an ($\Theta(n)$-bounded-auxiliary-input) adaptive argument of knowledge if* BP *satisfies n-TSDH and the same assumptions as in Thm. 3 (for $z = 1$).*

Proof. COMPLETENESS: $\boldsymbol{S} \in$ SUBSET-SUM iff there exists $\boldsymbol{b} \in \{0,1\}^n$ such that $\sum_{i=1}^n S_i b_i = s$. Here, π_1 proves that $b_i \in \{0,1\}$, π_2 proves that $c_i = S_i b_i$, $(\pi_{31}, \pi_{32}^\delta)$ proves that $d_j - c_j = d_{j+1}$ for $j < n$ and $d_n - c_n = 0$ (and thus $d_n = c_n$, $d_{n-1} = c_{n-1} + d_n$ and in general $d_j = \sum_{i=j}^n c_i = \sum_{i=j}^{n-1} S_i b_i$), and finally π_4 proves that $d_1 = s$. Thus, $\sum_{i=1}^n S_i b_i = s$. Thus, if the prover and verifier are honest then all arguments are accepted.

ARGUMENT OF KNOWLEDGE: follows, under the corresponding assumptions on BP, from the argument of knowledge property of every basic SNARK. Assume that both PKE assumptions hold and that there exists an adversary $\mathsf{A} = \mathsf{A}_{aok}$ that breaks argument of knowledge property of Π_{ssum}. We construct an adversary A_{dl} (resp., A_{tsdh}) that breaks the PDL (resp., TSDH) assumption as follows:

(a) since BP is $(n, \mathcal{F}_{1-\mathsf{rsft}}, Y_2\mathcal{F}_{1-\mathsf{rsft}}, 1)$-PKE secure, there exists an extractor that obtains $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$ (and the used randomizers r_b, r_c and r_d) from (B_1, B_2^γ), (C_1, C_2^γ), and (D_1, D_2^γ).
(b) From π_1: due to (a), A_{tsdh} has access to $\boldsymbol{b}$, and hence by the TSDH assumption and since the product SNARK is an argument of knowledge, $b_i \in \{0,1\}$.

(c) From π_2: due to (a), A_{tsdh} has access to $\boldsymbol{b}$ and $\boldsymbol{c}$, and hence by the TSDH assumption and since the product SNARK is an argument of knowledge, $\boldsymbol{c} = \boldsymbol{S} \circ \boldsymbol{b}$.
(d) From $(\pi_{31}, \pi_{32}^{\delta})$: since BP is $(\mathcal{F}_{1-\mathsf{rsft}}, Y_1\mathcal{F}_{\mathsf{C}}, Y_1\mathcal{F}_{\mathsf{C}}, 2)$-PKE secure, there exists an extractor that obtains a witness $(\pi_f^*)_f$ that the argument belongs to the correct span. Due to this, A_{dl} has access to all values required in Thm. 3, and hence by the $3n$-PDL assumption and since the shift SNARK is an argument of knowledge, $d_i = \sum_{j \geq i} c_j = \sum_{j \geq i} S_j b_j$ for all i.
(e) From π_4: due to (a), A_{tsdh} has access to $\boldsymbol{d}$, and hence by the TSDH assumption and since the product SNARK is an argument of knowledge, $\boldsymbol{e_1} \circ (\boldsymbol{d} - s\boldsymbol{e_1}) = \mathbf{0}_n$ and thus $d_1 = \sum_{i=1}^{n} S_i b_i = s$.

Hence, this argument system is a bounded-auxiliary-input argument of knowledge.

We note that this argument uses the extractors in the security proofs of Thm. 2 and Thm. 3, giving both times to the adversary an auxiliary input (the part of $\mathsf{crs}_{\mathsf{ssum}}$ that is not $\mathsf{crs}_{\times}$ or $\mathsf{crs}_{\mathsf{rsft}}$) of length $\Theta(n)$.

Perfect composable zero-knowledge: follows from the presence of the trapdoor (this allows the simulator to open all commitments to $\mathbf{0}_n$), and from the facts that $\mathbf{0}_n \circ \mathbf{0}_n = \mathbf{0}_n$ and that $\mathbf{0}_n$ is a shift of $\mathbf{0}_n$. The simulator creates (B_1, B_2^{γ}), (C_1, C_2^{γ}) and (D_1, D_2^{γ}) as random commitments to $\mathbf{0}_n$, and uses the knowledge of td (and the trapdoor property of the commitment scheme) to compute r_e^* such that $\mathsf{C}_{\mathsf{ck}}(\boldsymbol{e_1}; \mathbf{0}) = \mathsf{C}_{\mathsf{ck}}(\mathbf{0}_n; r_e^*)$. She then simulates all four basic arguments, based on her knowledge of the trapdoor. All product arguments are obviously correct when the committed values are equal to $\mathbf{0}_n$: $\mathbf{0}_n = \mathbf{0}_n \circ \mathbf{0}_n$ (this takes care of π_1), $\mathbf{0}_n = \boldsymbol{S} \circ \mathbf{0}_n$ (this takes care of π_2), $\mathbf{0}_n$ is a right shift of $\mathbf{0}_n$ (this takes care of $(\pi_{31}, \pi_{32}^{\delta})$), and $\mathbf{0}_n \circ (\mathbf{0}_n - s\mathbf{0}_n) = \mathbf{0}_n$ (this takes care of π_4). It is composable zero-knowledge, because it is same-string zero knowledge.

Witness-indistinguishability of the basic SNARKs guarantees that the simulated argument comes from the same distribution as the real argument. □

Efficiency. The prover computation is dominated by three commitments and the application of 3 product SNARKs and 1 shift SNARK, i.e., by $\Theta(n \log n)$ non-cryptographic operations and $\Theta(n)$ cryptographic operations. The latter is dominated by nine ($\approx n$)-wide multi-exponentiations (2 in commitments to $\boldsymbol{c}$ and $\boldsymbol{d}$ and in the shift argument, and 1 in each product argument), 7 in $\mathbb{G}_1$ and 4 in $\mathbb{G}_2$. The argument size is constant (11 group elements), and the verifier computation is dominated by *offline* computation of two $(n+1)$-wide multi-exponentiations (needed to once commit to $\boldsymbol{S}$) and *online* computation of 17 pairings (3 pairings to verify π_2, 2 pairings to verify each of the other product arguments, 4 pairings to verify the shift argument, and 6 pairings to verify the validity of 3 commitments). In Sect. 9, we will describe a batch-verification technique that allows to speed up on-line part of the verification of the Subset-Sum SNARK.

As always, multi-exponentiation can be sped up by using algorithms from [42, 46]; it can also be highly parallelized, potentially resulting in very fast parallel implementations of the zk-SNARK.

Let $a = \sum_{i=1}^{n} S_i b_i$ for $b_i \in \{0,1\}$.
Let (B_1, B_2^γ) be a commitment to $\boldsymbol{b}$.
Construct a product argument π_1 to show that $\boldsymbol{b} = \boldsymbol{b} \circ \boldsymbol{b}$.
Let (C_1, C_2^γ) be a commitment to $\boldsymbol{c} \leftarrow \boldsymbol{S} \circ \boldsymbol{b}$.
Construct a product argument π_2 to show that $\boldsymbol{c} = \boldsymbol{S} \circ \boldsymbol{b}$.
Let (D_1, D_2^γ) be a commitment to $\boldsymbol{d}$, where $d_i = \sum_{j \geq i} c_i$.
Construct a shift argument $(\pi_{31}, \pi_{32}^\delta)$ to show that $\boldsymbol{d} = (\boldsymbol{d} - \boldsymbol{c}) \gg 1$.
Construct a product argument π_4 to show that $\boldsymbol{e_1} \circ (\boldsymbol{d} - \boldsymbol{a}) = \mathbf{0}_n$.
Output $\pi_{\mathsf{rng}} = (B_1, B_2^\gamma, C_1, C_2^\gamma, D_1, D_2^\gamma, \pi_1, \pi_2, \pi_{31}, \pi_{32}^\delta, \pi_4)$.

Fig. 2. The new range argument Π_{rng}

7 New Range SNARK

In a *range SNARK*, given public range $[L\,..\,H]$, the prover aims to convince the verifier that he knows how to open commitment (A_1, A_2^γ) to a value $a \in [L\,..\,H]$. That is, that the common input (A_1, A_2^γ) is a commitment to vector $\boldsymbol{a}$ with $a_1 = a$ and $a_i = 0$ for $i > 1$.

We first remark that instead of the range $[L\,..\,H]$, one can consider the range $[0\,..\,H-L]$, and then use the homomorphic properties of the commitment scheme to add L to the committed value. Hence, we will just assume that the range is equal to $[0\,..\,H]$ for some $H \geq 1$. Moreover, the efficiency of the following SNARK depends on the range length.

The new range SNARK Π_{rng} is very similar to Π_{ssum}, except that one has to additionally commit to a value $a \in [0\,..\,H]$, use a specific sparse $\boldsymbol{S}$ with $S_i = \left\lfloor (H + 2^{i-1})/2^i \right\rfloor$ [17,39], and prove that $a = \sum_{i=1}^{n} S_i b_i$ for the committed a. Since $\boldsymbol{S} = (S_i)_{i=1}^n$ does not depend on the instance (i.e., on a), the verifier computation is $\Theta(1)$. On the other hand, since the commitment to a is given as an input to the prover (and not created by prover as part of the argument), Π_{rng} has a more complex simulation strategy, with one more element in the trapdoor.

Let $n = \lfloor \log_2 H \rfloor + 1$. Define $S_i = \left\lfloor (H + 2^{i-1})/2^i \right\rfloor$ for $i \in [1\,..\,n]$ and $\boldsymbol{S} = (S_i)$. We again use the interpolating commitment scheme. To prove that $a \in [0\,..\,H]$, we do the following.

The CRS generation $\mathsf{G}_{\mathsf{rng}}$ invokes the CRS generations of the commitment scheme, the product SNARK and the shift SNARK, sharing the same gk and trapdoor $\mathsf{td} = (\chi, \delta/\gamma)$ between the different invocations. In this case, the trapdoor has to include δ/γ (which is well defined, since $\gamma \neq 0$) since the simulator does not know how to open (A_1, A_2^γ); see the proof of Thm. 5 for more details. We note that the trapdoor only has to contain δ/γ, and not γ and δ separately. The CRS also contains the first half of a commitment $S_1' \leftarrow \prod (g_1^{\ell_i(\chi)})^{S_i}$ to $\boldsymbol{S}$, needed for a later efficient verification of the argument π_2. Clearly, the CRS can be computed efficiently from $\mathsf{crs}_{\mathsf{rsft}}$ (for $z = 1$).

The prover's actions on input (A_1, A_2^γ) are depicted by Fig. 2 (further explanations are given in the concise completeness proof in Thm.5). The only differences, compared to the prover computation of Π_{ssum}, are the computation

of b_i on step 1, and of π_4 on step 2. After receiving π_{rng}, the verifier performs the following checks:

- Four commitment validations: $\hat{e}(A_1, g_2^\gamma) = \hat{e}(g_1, A_2^\gamma)$, $\hat{e}(B_1, g_2^\gamma) = \hat{e}(g_1, B_2^\gamma)$, $\hat{e}(C_1, g_2^\gamma) = \hat{e}(g_1, C_2^\gamma)$, $\hat{e}(D_1, g_2^\gamma) = \hat{e}(g_1, D_2^\gamma)$.
- Three product argument verifications: $\hat{e}(B_1/g_1, B_2^\gamma) = \hat{e}(\pi_1, g_2^{\gamma Z(\chi)})$, $\hat{e}(S_1', B_2^\gamma) = \hat{e}(g_1, C_2^\gamma) \cdot \hat{e}(\pi_2, g_2^{\gamma Z(\chi)})$, $\hat{e}(g_1^{\ell_1(\chi)}, D_2^\gamma / A_2^\gamma) = \hat{e}(\pi_4, g_2^{\gamma Z(\chi)})$.
- One shift argument verification, consisting of two equality tests: $\hat{e}(\pi_{31}, g_2^{\delta Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, \pi_{32}^\delta)$, $\hat{e}(D_1/C_1\pi_{31}, g_2^{\delta Z(\chi)}) = \hat{e}(D_1, g_2^{\delta Z(\chi) Z^*(\chi)})$.

Theorem 5. *Π_{rng} is perfectly complete and composable zero-knowledge. If BP satisfies n-TSDH and the assumptions of Thm. 3, then Π_{rng} is an adaptive ($\Theta(n)$-bounded-auxiliary-input) argument of knowledge.*

Proof (Sketch). Completeness: $a \in [0\,..\,H]$ iff $a = \sum_{i=1}^n S_i b_i$ for some $b_i \in \{0,1\}$ [39] (see [17] for a formal proof). Here, π_1 proves that b_i are Boolean, π_2 proves that $c_i = S_i b_i$, $(\pi_{31}, \pi_{32}^\delta)$ proves that $d_j - c_j = d_{j+1}$ for $j < n$ and $d_n - c_n = 0$ (and thus $d_n = c_n$, $d_{n-1} = c_{n-1} + d_n$ and in general $d_j = \sum_{i=j}^n c_i = \sum_{i=j}^{n-1} S_i b_i$), and finally π_4 proves that $\boldsymbol{a} = (0, \ldots, 0, a)$ with $d_1 - a = \sum_{i=1}^n S_i b_i - a = 0$. Thus, $a = \sum_{i=1}^n S_i b_i$ and therefore, $a \in [0\,..\,H]$.

Perfect composable zero-knowledge: follows from the presence of the trapdoor (this allows the simulator to open all commitments to $\mathbf{0}_n$), and from the facts that $\mathbf{0}_n \circ \mathbf{0}_n = \mathbf{0}_n$ and that $\mathbf{0}_n$ is a shift of $\mathbf{0}_n$. The simulator creates (B_1, B_2^γ) and (C_1, C_2^γ) as random commitments to $\mathbf{0}_n$, computes $(D_1, D_2^\gamma) \leftarrow (A_1, A_2^\gamma) \cdot \mathsf{C}_{\mathsf{ck}}(\mathbf{0}_n; r_d^*)$ for a random r_d^*, and uses the knowledge of χ (and the trapdoor property of the commitment scheme) to compute r_e^* such that $\mathsf{C}_{\mathsf{ck}}(\boldsymbol{e_1}; \mathbf{0}) = \mathsf{C}_{\mathsf{ck}}(\mathbf{0}_n; r_e^*)$. She then simulates the basic arguments, based on her knowledge of χ. All product arguments are correct when the committed values are equal to $\mathbf{0}_n$: $\mathbf{0}_n = \mathbf{0}_n \circ \mathbf{0}_n$ (this takes care of π_1), $\mathbf{0}_n = \boldsymbol{S} \circ \mathbf{0}_n$ (this takes care of π_2), $\mathbf{0}_n$ is a right shift of $\mathbf{0}_n$, and $\mathbf{0}_n \circ \mathbf{0}_n = \mathbf{0}_n$ (this takes care of π_4).

To simulate the shift argument, the simulator uses the knowledge of δ/γ to set $(\pi_{31}, \pi_{32}^\delta) \leftarrow (D_1, (D_2^\gamma)^{\delta/\gamma})^{Z(\chi)-1} \cdot (C_1, (C_2^\gamma)^{\delta/\gamma})$. Clearly, the first verification of the shift argument succeeds. For the second verification, note that $\hat{e}((D_1/C_1) \cdot \pi_{31}, g_2^{\delta Z(\chi)}) = \hat{e}(D_1^{Z(\chi)}, g_2^{\delta Z(\chi)}) = \hat{e}(D_1, g_2^{\delta Z(\chi) Z^*(\chi)})$. Thus $(\pi_{31}, \pi_{32}^\delta)$ is an accepting shift argument.

Witness-indistinguishability of the basic SNARKs guarantees that the simulated argument comes from the same distribution as the real argument. Finally, this SNARK is composable zero-knowledge, because it is same-string zero knowledge. This finishes the proof.

Argument of knowledge: This part of the proof is very similar to the proof of Thm. 4 and thus omitted. □

The prover computation is dominated by three commitments and the application of three product arguments and one shift argument, that is, by $\Theta(n \log n)$ non-cryptographic operations and $\Theta(n)$ cryptographic operations. The latter is dominated by nine ($\approx n$)-wide multi-exponentiations (2 in commitments to $\boldsymbol{c}$ and $\boldsymbol{d}$ and in the shift argument, and 1 in each product argument), seven in

$\mathbb{G}_1$ and four in $\mathbb{G}_2$. The argument size is constant (11 group elements), and the verifier computation is dominated by 19 pairings (3 pairings to verify π_2, 2 pairings to verify each of the other product arguments, 4 pairings to verify the shift argument, and 8 pairings to verify the validity of 4 commitments). In this case, since the verifier does not have to commit to $\boldsymbol{S}$, the verifier computation is dominated by $\Theta(1)$ cryptographic operations.

The new range SNARK is significantly more computation-efficient for the prover than the previous range SNARKs [18, 23] that have prover computation $\Theta(r_3^{-1}(n) \log n)$. Π_{rng} has better communication (11 versus 31 group elements in [23]), and verification complexity (19 versus 65 pairings in [23]). Moreover, Π_{rng} is also simpler: since the prover computation is quasi-linear, we do not have to consider various trade-offs (though they are still available) between computation and communication as in [18, 23]. In Sect. 9, we will show that by using batch verification, one can further speed up verification of the Range SNARK.

8 Other SNARKs

As shown in [23], SNARKs for other interesting languages can be constructed, given efficient product and shift SNARKs. This includes NP-complete languages like Partition. One can plug in the interpolating commitment scheme and the new product SNARK to speed up corresponding SNARKs. Most of such SNARKs will have prover computation dominated by $\Theta(n \log n)$ non-cryptographic operations and $\Theta(n)$ cryptographic operations. Here, n is a language-dependent parameter (e.g., the size of the integer set in Subset-Sum). It is unknown how to achieve similar efficiency by using any other techniques. On the other hand, resulting Circuit-SAT and arithmetic Circuit-SAT arguments are slower by a factor of $\Theta(\log n)$.

In what follows, we give a brief description of some other NP-complete languages for which one can construct efficient adaptive zk-SNARKs by using the product-and-shift framework. We define the languages by using notation that facilitates design of such SNARKs.

Partition *[24, p. 223].* Partition is the set of all vectors $\boldsymbol{S} = (S_1, \ldots, S_n)$, such that there exist $\boldsymbol{b} \in \{0,1\}^n$ with $\sum_{i=1}^n S_i b_i = \frac{1}{2} \cdot \sum_{i=1}^n S_i$. Thus, Partition is a special case of Subset-Sum with $s = \frac{1}{2} \cdot \sum_{i=1}^n S_i$. A Partition SNARK follows trivially from Π_{ssum}.

Subset-Product *[24, p. 224].* Subset-Product is the set of all vectors $(\boldsymbol{S} = (S_1, \ldots, S_n), s)$, such that there exist $\boldsymbol{b} \in \{0,1\}^n$ with $\prod_{i:b_i=1} S_i b_i = s$. The resulting SNARK is almost the same as Π_{ssum}, except one step. Namely, the prover computes still a vector $\boldsymbol{c} = \boldsymbol{S} \circ \boldsymbol{b}$. However, differently from Π_{ssum} of Sect. 6, the prover now lets $\boldsymbol{d}$ to be the multiplicative scan of $\boldsymbol{c}$ with $d_i = \prod_{j \geq i} c_j$, and then proves the correctness of $\boldsymbol{d}$ by using a product SNARK.

Two-Processor Scheduling *(*2PS *for short) [24, p. 65].* 2PS is the set of all vectors $(\boldsymbol{S} = (S_1, \ldots, S_n), s)$, such that there exist $\boldsymbol{b} \in \{0,1\}^n$ with $\sum_{i=1}^n S_i b_i \leq s$ and $\sum_{i=1}^n S_i (1 - b_i) \leq s$. To construct a zk-SNARK for 2PS, we show that the scan of $\boldsymbol{S} \circ \boldsymbol{b}$ is in the range $[0 \,..\, s]$. Here, we use the range SNARK of Sect. 7 twice.

One can similarly construct a zk-SNARK for multiprocessor scheduling [24, p. 65] for m processors, but the complexity of the SNARK will depend linearly on m.
KNAPSACK *[24, p. 65].* KNAPSACK is the set of vectors $(\boldsymbol{S} = (S_1, \ldots, S_n), \boldsymbol{V} = (V_1, \ldots, V_n), s, v)$, such that there exist $\boldsymbol{b} \in \{0,1\}^n$ with $\sum_{i=1}^n S_i b_i \leq s$ and $\sum_{i=1}^n V_i b_i \geq v$. To construct a zk-SNARK for KNAPSACK, we show that the scan of $\boldsymbol{S} \circ \boldsymbol{b}$ is in the range $[0\,..\,s]$ and the scan of $\boldsymbol{V} \circ \boldsymbol{b}$ is in the range of (say) $[v\,..\,p-1]$. Here, we use the range SNARK of Sect. 7 twice.
CIRCUIT-SAT. Recently, Lipmaa [38] used Beneš network to construct a permutation SNARK out of $\Theta(\log n)$ product and shift SNARKs, and then Groth's CIRCUIT-SAT zk-SNARK [28] to construct a CIRCUIT-SAT out of a small constant number of product and permutation SNARKs. By using the new product and shift SNARKs of the current paper together with proof bootstrapping [6,19], one can construct an adaptive CIRCUIT-SAT zk-SNARK with complexity parameters as stated in Tbl. 1. Without proof bootstrapping, the communication will be $\Theta(\log n)$ group elements. (In the case of [38], Tbl. 1 includes its combination with two different product SNARKs.)
Arithmetic CIRCUIT-SAT. Arithmetic CIRCUIT-SAT is the set of all circuits $\mathcal{C}$, such that for some secret input vector $\boldsymbol{x}$ and public output vector $\boldsymbol{y}$, $\mathcal{C}(\boldsymbol{x}) = \boldsymbol{y}$. One can construct a zk-SNARK for arithmetic CIRCUIT-SAT by using the same ideas as in the case of Boolean CIRCUIT-SAT, see [28]. Here, we just outline the differences compared to the Boolean CIRCUIT-SAT zk-SNARK in [28]. First, in the case of arithmetic CIRCUIT-SAT, there is no need to verify that $\boldsymbol{x}$ is Boolean. Second, the output of the circuit can be equal to any $\boldsymbol{y}$ (not just 0). Third, the step that verifies that all gates are correctly executed is somewhat more complicated. In the case of Boolean CIRCUIT-SAT, one can assume that each gate is a NAND gate, and thus one just must verify that $\boldsymbol{x} \circ \boldsymbol{y} = \boldsymbol{1} - \boldsymbol{z}$, where $\boldsymbol{x}$ is the vector of left inputs to all gates, $\boldsymbol{y}$ is the vector of right inputs, and $\boldsymbol{z}$ is the vector of outputs. In the case of arithmetic CIRCUIT-SAT, one has to check that $\boldsymbol{z} = \boldsymbol{m} \circ (\boldsymbol{x} + \boldsymbol{y}) + (\boldsymbol{1} - \boldsymbol{m}) \circ (\boldsymbol{x} \circ \boldsymbol{y})$, where $m_i = 1$ iff the ith gate is the addition gate and $m_i = 0$ otherwise.

9 Batch Verification

In many known applications (however, up to now SNARK has not been one of them, one can speed up the verification process by applying batch verification [4]. As already mentioned, [19] proposed proof bootstrapping to shorten the arguments and improve verification time, however, proof bootstrapping only results in the efficient provided by QAP [26]. Batch verification enables us to improve verification time even further.

One of the known batch verification techniques is based on the following idea: instead of testing P equalities $X_i = Y_i$ for $i \in [1\,..\,P]$ (in a cyclic group of order p), the verifier generates random values $\tau_i \leftarrow \mathbb{Z}_p$ for $i \in [1\,..\,P-1]$, and then tests whether $\prod_{i=1}^{P-1} X_i^{\tau_i} \cdot X_P = \prod_{i=1}^{P-1} Y_i^{\tau_i} \cdot Y_P$. Clearly, if the latter holds, then with probability $1 - 1/p$ over the choice of $(\tau_i)_{i=1}^{P-1}$, $X_i = Y_i$ holds for each $i \in [1\,..\,P]$.

Importantly, if X_i and Y_i have a suitable structure, such a simple batch verification can save a significant amount of computation. For example, if $X_i = \hat{e}(U_i, V)$ for some U_i and V where V does not depend on i, then the computation of $\prod_{i=1}^{P-1} X_i^{\tau_i} \cdot X_P = \hat{e}(\prod_{i=1}^{P-1} U_i^{\tau_i} \cdot U_P, V)$ is dominated by just using a single pairing and one $(P-1)$-wide multi-exponentiation. We will next give some examples how to use batch-verification together with the new SNARKs.

Batched Commitment Validity Verification. Assume that the verifier is given commitments $(A_{i1}, A_{i2}^{\gamma})$, $i \in [1\,..\,P]$, and she has to verify that all of them are valid. In a naive verification, she checks that $\hat{e}(A_{i1}, g_2^{\gamma}) = \hat{e}(g_1, A_{i2}^{\gamma})$ for $i \in [1\,..\,P]$, and thus computes $2P$ bilinear pairings. In a batched version, she checks that $\prod_{i=1}^{P} \hat{e}(A_{i1}, g_2^{\gamma})^{\tau_i} = \prod_{i=1}^{P} \hat{e}(g_1, A_{i2}^{\gamma})^{\tau_i}$, for $\tau_i \leftarrow \mathbb{Z}_q$ (and $\tau_P = 1$). This can be simplified to

$$\hat{e}(\prod_{i=1}^{P} A_{i1}^{\tau_i}, g_2^{\gamma}) = \hat{e}(g_1, \prod_{i=1}^{P} A_{i2}^{\gamma\tau_i}) \ ,$$

where again $\tau_P = 1$. Thus, the verifier has to compute 2 pairings, one $(P-1)$-wide multi-exponentiation in $\mathbb{G}_1$, and one $(P-1)$-wide multi-exponentiation in $\mathbb{G}_2$.

Batched Product Argument Verification. Assume that the verifier is given P product arguments π_i, each for some triple $((A_{i1}, A_{i2}^{\gamma}), (B_{i1}, B_{i2}^{\gamma}), (C_{i1}, C_{i2}^{\gamma}))$ of commitments. In a naive verification, she checks that $\hat{e}(A_{i1}, B_{i2}^{\gamma}) = \hat{e}(g_1, C_{i2}^{\gamma}) \cdot \hat{e}(\pi_i, g_2^{\gamma Z(\sigma)})$ for $i \in [1\,..\,P]$, and thus computes $3P$ bilinear pairings. In a batched version, she checks that $\prod_{i=1}^{P} \hat{e}(A_{i1}, B_{i2}^{\gamma})^{\tau_i} = \prod_{i=1}^{P} (\hat{e}(g_1, C_{i2}^{\gamma}) \cdot \hat{e}(\pi_i, g_2^{\gamma Z(\sigma)}))^{\tau_i}$, for $\tau_i \leftarrow \mathbb{Z}_q$ (and $\tau_P = 1$), which can be simplified to

$$\prod_{i=1}^{P} \hat{e}(A_{i1}, B_{i2}^{\gamma})^{\tau_i} = \hat{e}(g_1, \prod_{i=1}^{P} C_{i2}^{\gamma\tau_i}) \cdot \hat{e}(\prod_{i=1}^{P} \pi_i^{\tau_i}, g_2^{\gamma Z(\sigma)}) \ ,$$

where again $\tau_P = 1$. Hence, the verifier only has to compute $P+2$ pairings, one P-wide multi-exponentiation in $\mathbb{G}_1$, one $(P-1)$-wide multi-exponentiation in $\mathbb{G}_2$, and one $(P-1)$-wide multi-exponentiation in $\mathbb{G}_T$.

In the special case where the values A_{i1} are all equal to some A_1, the verification can be simplified to

$$\hat{e}(A_1, \prod_{i=1}^{P} B_{i2}^{\gamma\tau_i}) = \hat{e}(g_1, \prod_{i=1}^{P} C_{i2}^{\gamma\tau_i}) \cdot \hat{e}(\prod_{i=1}^{P} \pi_i^{\tau_i}, g_2^{\gamma Z(\sigma)}) \ ,$$

where again $\tau_P = 1$. In this case, the verifier only has to compute 3 pairings, one $(P-1)$-wide multi-exponentiation in $\mathbb{G}_1$ and two $(P-1)$-wide multi-exponentiations in $\mathbb{G}_2$. The case where all values B_{i2}^{γ} are equal is dual.

Batched Shift Argument Verification. Assume that the verifier is given P shift arguments $\pi_i = (\pi_{i1}, \pi_{i2}^{\delta})$, each for a pair $((A_{i1}, A_{i2}^{\gamma}), (B_{i1}, B_{i2}^{\gamma})$ of commitments. In a naive verification, she checks that $\hat{e}(\pi_{i1}, g_2^{\delta Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, \pi_{i2}^{\delta})$ and $\hat{e}(B_{i1}\pi_{i1}, g_2^{\delta Z(\chi)}) = \hat{e}(A_{i1}, g_2^{\delta Z(\chi) Z^*(\chi)})$ for $i \in [1\,..\,P]$, and thus computes $4P$ pairings. In a batched version, she checks that $\prod_{i=1}^{P} \hat{e}(\pi_{i1}, g_2^{\delta Z(\chi)})^{\tau_i} \prod_{i=P+1}^{2P} \hat{e}(B_{i1}\pi_{i1}, g_2^{\delta Z(\chi)})^{\tau_i} = \prod_{i=1}^{P} \hat{e}(g_1^{Z(\chi)}, \pi_{i2}^{\delta})^{\tau_i} \cdot$

$\prod_{i=P+1}^{2P} \hat{e}(A_{i1}, g_2^{\delta Z(\chi) Z^*(\chi)})^{\tau_i}$, for $\tau_i \leftarrow \mathbb{Z}_q$ (and $\tau_P = 1$), which simplifies to

$$\hat{e}(\prod_{i=1}^{P} \pi_{i1}^{\tau_i} \cdot \prod_{i=P+1}^{2P} (B_{i1}\pi_{i1})^{\tau_i}, g_2^{\delta Z(\chi)}) = \hat{e}(g_1^{Z(\chi)}, \prod_{i=1}^{P} \pi_{i2}^{\delta\tau_i}) \cdot \hat{e}(\prod_{i=P+1}^{2P} A_{i1}^{\tau_i}, g_2^{\delta Z(\chi) Z^*(\chi)}) \ ,$$

where again $\tau_P = 1$. Hence, the verifier only has to compute 3 pairings, one $(P-1)$-wide and one $(2P-1)$-wide multi-exponentiation in $\mathbb{G}_1$, and one P-wide multi-exponentiation in $\mathbb{G}_2$.

Batched Subset-Sum SNARK. We now show how to speed up the online part of the verification of the Subset-Sum SNARK from Sect. 6. In the batch-verification, the verifier generates new random values $\tau_i \leftarrow \mathbb{Z}_q$ for $i \in [1\,..\,8]$ except that $\tau_6 = \tau_8 \leftarrow 1$, and then checks that

$$\begin{aligned} \hat{e}(B_1^{\tau_1} C_1^{\tau_2} D_1^{\tau_3}, g_2^{\gamma}) \cdot \hat{e}((B_1/g_1)^{\tau_4} (S_1')^{\tau_5}, B_2^{\gamma}) \cdot \hat{e}((g_1^{\ell_1(\chi)})^{\tau_6}, D_2^{\gamma}/(g_2^{\gamma \ell_1(\chi)})^{s}) \cdot \\ = \hat{e}(g_1, B_2^{\gamma\tau_1} C_2^{\gamma(\tau_2+\tau_5)} D_2^{\gamma\tau_3}) \cdot \hat{e}(\pi_1^{\tau_4} \pi_2^{\tau_5} \pi_4^{\tau_6}, g_2^{\gamma Z(\chi)}) \ , \\ \hat{e}(\pi_{31}^{\tau_7} (D_1/C_1\pi_{31})^{\tau_8}, g_2^{\delta Z(\chi)}) = \hat{e}((g_1^{Z(\chi)})^{\tau_7}, \pi_{32}^{\delta}) \cdot \hat{e}(D_1^{\tau_8}, g_2^{\delta Z(\chi) Z^*(\chi)}) \ . \end{aligned}$$

Hence, the verifier only requires 8 pairings, two 3-way multi-exponentiations in $\mathbb{G}_1$, two 2-way multi-exponentiations in $\mathbb{G}_1$, three exponentiations in $\mathbb{G}_1$, one 3-way multi-exponentiation in $\mathbb{G}_2$, and one exponentiation in $\mathbb{G}_2$.

Batched Range SNARK. We now speed up the verification of the Range SNARK from Sect. 7. In the batch-verification, the verifier generates new random values $\tau_i \leftarrow \mathbb{Z}_q$ for $i \in [0\,..\,8]$ except that $\tau_6 = \tau_8 \leftarrow 1$, and then checks that

$$\begin{aligned} \hat{e}(A_1^{\tau_0} B_1^{\tau_1} C_1^{\tau_2} D_1^{\tau_3}, g_2^{\gamma}) \cdot \hat{e}(g_1^{-\tau_1} (B_1/g_1)^{\tau_4} (S_1')^{\tau_5}, B_2^{\gamma}) \cdot \hat{e}((g_1^{\ell_1(\chi)})^{\tau_6}, D_2^{\gamma}/A_2^{\gamma}) \\ = \hat{e}(g_1, A_2^{\gamma\tau_0} C_2^{\gamma(\tau_2+\tau_5)} D_2^{\gamma\tau_3}) \cdot \hat{e}(\pi_1^{\tau_4} \pi_2^{\tau_5} \pi_4^{\tau_6}, g_2^{\gamma Z(\chi)}) \ , \\ \hat{e}(\pi_{31}^{\tau_7} (D_1/C_1\pi_{31})^{\tau_8}, g_2^{\delta Z(\chi)}) = \hat{e}((g_1^{Z(\chi)})^{\tau_7}, \pi_{32}^{\delta}) \cdot \hat{e}(D_1^{\tau_8}, g_2^{\delta Z(\chi) Z^*(\chi)}) \ . \end{aligned}$$

Since $\tau_6 = \tau_8 = 1$, the verifier only requires 8 pairings, one 4-way multi-exponentiation in $\mathbb{G}_1$, two 3-way multi-exponentiations in $\mathbb{G}_1$, one 2-way multi-exponentiation in $\mathbb{G}_1$, three exponentiations in $\mathbb{G}_1$, and one 3-way multi-exponentiation in $\mathbb{G}_2$.

Barreto-Naehrig Curves. We use exponentiation speed records from [15] and pairing speed records from [2] for Barreto-Naehrig curves. According to Tbl. 4 in [15], a pairing, exponentiation in $\mathbb{G}_1$, exponentiation in $\mathbb{G}_2$, and exponentiation in $\mathbb{G}_T$ take respectively 7.0, 0.9, 1.8, and 3.1 millions of clock cycles on the Core i7-3520M CPU. We also assume that one P-way multi-exponentiation in a group can be computed in the time of $P/\log_2 P$ exponentiations in the same group.

Taken millions of clock cycles as the basic unit, in a non-batched Subset-Sum argument the verifier's online computation is dominated by $17 \cdot 7.0 + 1 \cdot 0.9 = 119.9$ units. In the batched variant, the verifier's online computation is dominated by $8 \cdot 7.0 + (2 \cdot (3/\log_2 3) + 2 \cdot (2/\log_2 2) + 3) \cdot 0.9 + (1 \cdot 3/\log_2 3 + 1) \cdot 1.8 = 70.9$ units. Hence, in this case this simple batching technique results in an approximately 1.7 times speed-up.

Similarly, in a non-batched RANGE argument the verifier's computation is dominated by $19 \cdot 7.0 = 133.0$ units. In the batched variant, the verifier's online computation is dominated by $8 \cdot 7.0 + (1 \cdot 4/\log_2 4 + 2 \cdot 3/\log_2 3 + 2/\log_2 2 + 3) * 0.9 + (1 \cdot 3/\log_2 3) \cdot 1.8 = 69.1$ units. Hence, in this case this simple batching technique results in an approximately 1.9 times speed-up.

Other SNARKs. The described batch-verification can also be used with other SNARKs of the current and preceding papers. For example, in Pinocchio [40] the verifier executes 11 pairings. Batch-verification reduces this number to 8 pairings and a small number of exponentiations; this means that the verifier of the new SUBSET-SUM and Range SNARKs is essentially as fast as the verifier of Pinocchio. Batch-verification is especially effective in the cases when P is large. It can also be used as an additional step after proof boot-strapping [6].

10 Additional Discussion

On Input Size and n. When using Groth's modular approach, n is the dimension of the vectors, and every coefficient is an element of $\mathbb{Z}_p$. Hence, it is to be expected that n is smaller than the input size. In the case of SUBSET-SUM, n is the number of the integers in the base set, and the input size is $N = \Theta(\kappa n)$ bits, where κ is the security parameter. Hence, in the SUBSET-SUM zk-SNARK, the prover executes $\Theta(N/\kappa)$ cryptographic operations; since $n = \kappa^{O(1)}$, this is sublinear in the input size. For example, if $\kappa = n^{1/c}$, then the prover executes $\Theta(N^{1/(c+1)})$ cryptographic operations. In the case of the range proof $n \approx \log_2(H - L)$, which can be significantly smaller than the input length $N = \lceil \log_2 L \rceil + n$.

To compare, in Boolean (resp., arithmetic) CIRCUIT-SAT zk-SNARKs (see, e.g. [21, 26, 28, 36, 37]), n is the number of the gates, while the input size is $N = \Theta(n \log n)$ (resp., $N = \Theta(n \log n \cdot \kappa)$) bits. Thus, in the QSP-based adaptive Boolean CIRCUIT-SAT zk-SNARK, the prover computation is $\Theta(N)$ cryptographic operations. (As before, adaptive arithmetic CIRCUIT-SAT zk-SNARK is less efficient due to less efficient known universal arithmetic circuits.) According to this comparison, one of the main results of the current paper is the first adaptive zk-SNARK for an NP-complete language where the prover computation is dominated by $O(N/\kappa)$ cryptographic operations.

On Possible QAP-Based Solutions. While there might exist QAP-based adaptive zk-SNARKs for NP-complete languages with similar prover-complexity, we are not aware of any previous research on this topic. Even if such zk-SNARKs were constructed in the future, the modular approach has advantages that we already emphasized, like reliance on an instance (and language) independent commitment scheme or the need to commit to shorter vectors.

For example, one can construct a QAP to implement the relation $\boldsymbol{c} = d(\boldsymbol{a} \circ \boldsymbol{b}) + (1 - d)(\boldsymbol{a} \gg z)$, where $d \in \{0, 1\}$. However, this would result in a QAP $C\boldsymbol{d} = A_d\boldsymbol{a} \circ B_d\boldsymbol{b}$, where the matrices A_d and B_d depend on d and moreover, $A_d \neq B_d$. This means that one would have to commit to $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ by using different commitment schemes. One runs into additional complications when the

same vector are used instead of $\boldsymbol{a}$, $\boldsymbol{b}$, or $\boldsymbol{b}$, with different values of d, in different instantiations of this relation. While such complications are probably solvable, they result in a less intuitive and less efficient commitment scheme than proposed in the current paper.

In particular, while [19] proposes a QAP-based CaP zk-SNARK, it is very different from the zk-SNARKs of the current paper. Intuitively, by using sophisticated techniques, they combine QAP-s of small circuits together with commitments to shared data (i.e., input and wire values) to obtain a QAP of a large circuit. Unfortunately, [19] does neither specify the achieved prover's computational complexity or the used commitment scheme. Due to the lack of space, we leave precise comparison to a future work.

References

1. Abe, M., Fehr, S.: Perfect NIZK with Adaptive Soundness. In: TCC 2007. LNCS, vol. 4392, pp. 118–136
2. Aranha, D.F., Barreto, P.S.L.M., Longa, P., Ricardini, J.E.: The Realm of the Pairings. In: SAC 2013. LNCS, vol. 8282, pp. 3–25
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: SAC 2005. LNCS, vol. 3897, pp. 319–331
4. Bellare, M., Garay, J.A., Rabin, T.: Batch Verification with Applications to Cryptography and Checking. In: LATIN 1998. LNCS, vol. 1380, pp. 170–191
5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In: CRYPTO (2) 2013. LNCS, vol. 8043, pp. 90–108
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable Zero Knowledge via Cycles of Elliptic Curves. In: CRYPTO (2) 2014. LNCS, vol. 8617, pp. 276–294
7. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In: USENIX 2014, pp. 781–796
8. Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the Existence of Extractable One-Way Functions. In: STOC 2014, pp. 505–514
9. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct Non-interactive Arguments via Linear Interactive Proofs. In: TCC 2013. LNCS, vol. 7785, pp. 315–333
10. Blake, I.F., Seroussi, G., Smart, N.P., eds.: Advances in Elliptic Curves in Cryptography. 2 edn. London Mathematical Society Lecture Note Series. Cambridge Univ Press (2005)
11. Blelloch, G.: Vector Models for Data-Parallel Computing. MIT Press (1990)
12. Blum, M., Feldman, P., Micali, S.: Non-Interactive Zero-Knowledge and Its Applications. In: STOC 1988, pp. 103–112
13. Boneh, D., Boyen, X.: Secure Identity Based Encryption Without Random Oracles. In: CRYPTO 2004. LNCS, vol. 3152, pp. 443–459
14. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. J. Cryptology **21**(2) (2008) pp. 149–177
15. Bos, J.W., Costello, C., Naehrig, M.: Exponentiating in Pairing Groups. In: SAC 2013. LNCS, vol. 8282, pp. 438–455
16. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally Composable Two-Party and Multi-Party Secure Computation. In: STOC 2002, pp. 494–503

17. Chaabouni, R., Lipmaa, H., shelat, a.: Additive Combinatorics and Discrete Logarithm Based Range Protocols. In: ACISP 2010. LNCS, vol. 6168, pp. 336–351
18. Chaabouni, R., Lipmaa, H., Zhang, B.: A Non-Interactive Range Proof with Constant Communication. In: FC 2012. LNCS, vol. 7397, pp. 179–199
19. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile Verifiable Computation. In: IEEE SP 2015, pp. 253–270
20. Damgård, I.: Towards Practical Public Key Systems Secure against Chosen Ciphertext Attacks. In: CRYPTO 1991. LNCS, vol. 576, pp. 445–456
21. Danezis, G., Fournet, C., Groth, J., Kohlweiss, M.: Square Span Programs with Applications to Succinct NIZK Arguments. In: ASIACRYPT 2014 (1). LNCS, vol. 8873, pp. 532–550
22. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust Non-interactive Zero Knowledge. In: CRYPTO 2001. LNCS, vol. 2139, pp. 566–598
23. Fauzi, P., Lipmaa, H., Zhang, B.: Efficient Modular NIZK Arguments from Shift and Product. In: CANS 2013. LNCS, vol. 8257, pp. 92–121
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Series of Books in the Mathematical Sciences. W. H. Freeman (1979)
25. Gathen, J., Gerhard, J.: Modern Computer Algebra. 2 edn. Cambridge University Press (2003)
26. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and NIZKs without PCPs. In: EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645
27. Gentry, C., Wichs, D.: Separating Succinct Non-Interactive Arguments from All Falsifiable Assumptions. In: STOC 2011, pp. 99–108
28. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340
29. Groth, J., Ostrovsky, R., Sahai, A.: New Techniques for Noninteractive Zero-Knowledge. Journal of the ACM **59**(3) (2012)
30. Hess, F., Smart, N.P., Vercauteren, F.: The Eta Pairing Revisited. IEEE Trans. on Inf. Theory **52**(10) (2006) pp. 4595–4602
31. Jutla, C.S., Roy, A.: Shorter Quasi-Adaptive NIZK Proofs for Linear Subspaces. In: ASIACRYPT 2013 (1). LNCS, vol. 8269, pp. 1–20
32. Jutla, C.S., Roy, A.: Switching Lemma for Bilinear Tests and Constant-Size NIZK Proofs for Linear Subspaces. In: CRYPTO (2) 2014. LNCS, vol. 8617, pp. 295–312
33. Kilian, J.: Uses of Randomness in Algorithms and Protocols. PhD thesis, Massachusetts Institute of Technology, USA (1989)
34. Kolesnikov, V., Schneider, T.: A Practical Universal Circuit Construction and Secure Evaluation of Private Functions. In: FC 2008. LNCS, vol. 5143, pp. 83–97
35. Lipmaa, H.: On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In: ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415
36. Lipmaa, H.: Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In: TCC 2012. LNCS, vol. 7194, pp. 169–189
37. Lipmaa, H.: Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes. In: ASIACRYPT 2013 (1). LNCS, vol. 8269, pp. 41–60
38. Lipmaa, H.: Efficient NIZK Arguments via Parallel Verification of Benes Networks. In: SCN 2014. LNCS, vol. 8642, pp. 416–434
39. Lipmaa, H., Asokan, N., Niemi, V.: Secure Vickrey Auctions without Threshold Trust. In: FC 2002. LNCS, vol. 2357, pp. 87–101

40. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: IEEE SP 2013, pp. 238–252
41. Pereira Geovandro, C.C.F., Simplício Jr., M.A., Naehrig, M., Barreto, P.S.L.M.: A Family of Implementation-Friendly BN Elliptic Curves. Journal of Systems and Software **84**(8) (2011) pp. 1319–1326
42. Pippenger, N.: On the Evaluation of Powers and Monomials. SIAM J. Comput. **9**(2) (1980) pp. 230–250
43. Pisinger, D.: Linear Time Algorithms for Knapsack Problems with Bounded Weights. J. Algorithms **33**(1) (1999) pp. 1–14
44. Raz, R.: Elusive Functions and Lower Bounds for Arithmetic Circuits. Theory of Computing **6**(1) (2010) pp. 135–177
45. Sadeghi, A.R., Schneider, T.: Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In: ICISC 2008. LNCS, vol. 5461, pp. 336–353
46. Straus, E.G.: Addition Chains of Vectors. Amer. Math. Monthly **70** (1964) pp. 806–808
47. Valiant, L.G.: Universal Circuits (Preliminary Report). In: STOC 1976, pp. 196–203

A On Cryptographic and Non-Cryptographic Operations

We count communication often implicitly in group elements, and verifier computation in the number of cryptographic operations. In the case of prover computation (which is the focus of the current work), very often the number of non-cryptographic operations and cryptographic operations differs, and thus we count them separately. According to the usual but somewhat informal practice, non-cryptographic operations count cheap operations (additions or multiplications) in $\mathbb{Z}_p$, while cryptographic operations count more expensive operations (exponentiations or pairings) in a cryptographic group. The basic difference is that non-cryptographic operations are significantly (usually by more than a factor of $\log n$) more efficient than cryptographic operations.