A Subversion-Resistant SNARK[⋆]

May 26, 2017

Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michał Zając

University of Tartu, Estonia

Abstract. While succinct non-interactive zero-knowledge arguments of knowledge (zk-SNARKs) are widely studied, the question of what happens when the CRS has been subverted has received little attention. In ASIACRYPT 2016, Bellare, Fuchsbauer and Scafuro showed the first negative and positive results in this direction, proving also that it is impossible to achieve subversion soundness and (even non-subversion) zero knowledge at the same time. On the positive side, they constructed an involved sound and subversion zero-knowledge argument system for NP. We show that Groth's zk-SNARK for Circuit-SAT from EUROCRYPT 2016 can be made computationally knowledge-sound and perfectly composable Sub-ZK with minimal changes. We just require the CRS trapdoor to be extractable and the CRS to be publicly verifiable. To achieve the latter, we add some new elements to the CRS and construct an efficient CRS verification algorithm. We also provide a definitional framework for sound and Sub-ZK SNARKs and describe implementation results of the new Sub-ZK SNARK.

Keywords: Common reference string, generic group model, non-interactive zero knowledge, SNARK, subversion zero knowledge

1 Introduction

Combined effort of a large number of recent research papers (to only mention a few, [26,32,22,37,33,17,27]) has made it possible to construct very efficient succinct non-interactive zero-knowledge arguments of knowledge (zk-SNARKs) for both the Boolean and the Arithmetic Circuit-SAT and thus for NP. The most efficient known approach for constructing zk-SNARKs for the Arithmetic Circuit-SAT is based on Quadratic Arithmetic Programs (QAP, [22]).

In a QAP, the prover builds a set of polynomial equations that are then checked by the verifier by using a small number of pairings. QAP-based zk-SNARKs have additional nice properties that make them applicable in verifiable computation [21,22,37] where the client outsources some computation to the server, who returns the computation result together with a succinct efficiently-verifiable correctness argument. Especially due to this application, zk-SNARKs

[⋆] An earlier version of this paper was submitted to Crypto 2017. The current version includes implementation data and readability improvements have several heavily optimized implementations [37,8,9,15]. Other applications of zk-SNARK include cryptocurrencies [5]. See, e.g., [27] for more references.

One drawback of zk-SNARKs is that they are all based on non-falsifiable assumptions (like the knowledge assumptions [16], or the generic bilinear group model, GBGM [36,40,35,13]). In fact, Gentry and Wichs [23] showed that non-falsifiable assumptions are needed to construct zk-SNARKs for non-trivial languages. The currently most efficient zk-SNARK for Arithmetic CIRCUIT-SAT was proposed by Groth (EUROCRYPT 2016, [27]) who proved it to be knowledge-sound in the GBGM. In Groth's zk-SNARK, the argument consists of only 3 bilinear group elements and the verifier has to check a single pairing equation, dominated by the computation of only 3 bilinear (type III [20]) pairings and m_0 exponentiations, where m_0 is the statement size.

After the Snowden revelations, there has been a recent surge of interest in constructing cryptographic primitives and protocols secure against active subversion. In the context of zk-SNARKs, while the common reference string (CRS) model [12] is widely accepted to be the proper model, one has to be very careful to guarantee that the CRS has been created correctly. In [3], Bellare, Fuchsbauer and Scafuro tackled this problem by studying how much security one can still achieve when the CRS generator cannot be trusted. They proved several negative and positive results. In particular, they showed that it is impossible to achieve subversion soundness and (even non-subversion) zero knowledge simultaneously, the essential reason being that the zero knowledge simulator can be used to break subversion soundness.

In one of their positive solutions, Bellare *et al.* show that it is possible to get (non-subversion) soundness and computational subversion zero knowledge (Sub-ZK, ZK even if the the CRS is not trusted). Their main new idea is to use a knowledge assumption in the Sub-ZK proof, so that the simulator can extract a "trapdoor" from the untrusted CRS and then use this trapdoor to simulate the argument. While neat, the resulting argument system is quite complicated. Moreover, the non-interactive Sub-ZK argument system of [3] has linear communication; in the case of zk-SNARKs one presumably has to employ different techniques. We also need to take care to define and implement statistical Sub-ZK.

Our Contributions. We will take Groth's zk-SNARK from EUROCRYPT 2016 [27] as a starting point since, as mentioned before, it is currently the most efficient and thus the most attractive (for us) zk-SNARK. We propose a minimal modification to Groth's zk-SNARK that makes it computationally knowledge-sound in what we call the "subversion generic bilinear group model" (Sub-GBGM) and perfect composable Sub-ZK. In fact, we consider two different versions of perfect Sub-ZK: (i) the version with an efficient subverter, where we assume the existence of an efficient extractor and prove Sub-ZK under a knowledge assumption, and (ii) the version with a computationally unbounded subverter, where we assume the existence of a computationally unbounded extractor and prove Sub-ZK unconditionally.

We change Groth's zk-SNARK by adding extra elements to the CRS so that the CRS will become publicly verifiable; this minimal step (clearly, some public verifiability of the CRS is needed in the case the CRS generator cannot be trusted) will be sufficient to obtain Sub-ZK. However, choosing which elements to add to the CRS is not straightforward since the zk-SNARK must remain knowledge-sound even given enlarged CRS; adding too many (or just "wrong") elements to the CRS can break the knowledge-soundness. On the other hand, importantly, the prover and the verifier of the new zk-SNARK are unchanged compared to Groth's SNARK [27]. In the rest of the introduction, we will only outline the novel properties of the new SNARK as compared to [27].

We start by defining perfect subversion-complete (this includes the requirement that an honestly generated CRS is accepted by the CRS verification), computationally adaptively knowledge-sound, and statistically unbounded (or composable) Sub-ZK SNARKs. These definitions are similar to but subtly different from the non-subversion security definitions as given in, say, [25]. First, since one cannot check whether the subverter uses perfectly uniform coins (or, the *CRS trapdoor*) to generate the CRS, we divide the CRS generation into three different algorithms:

– generation of the CRS trapdoor tc (a probabilistic algorithm K_{tc}),
– creation of the CRS from tc (a deterministic algorithm $\mathsf{K}_{\mathsf{crs}}$), and
– creation of the simulation trapdoor from tc (a deterministic algorithm K_{ts}).

While we cannot check that K_{tc} works correctly, we will guarantee that given a fixed tc, $\mathsf{K}_{\mathsf{crs}}$ has been executed on this tc. More precisely, we require that a Sub-ZK SNARK satisfies a CRS trapdoor extractability property that allows one to extract tc used by the subverter, s.t. if the subverted CRS crs is accepted by the CRS verification algorithm (see below) then K_{tc} maps tc to crs. The extractability requirement forces our ZK proof to use either a computationally unbounded extractor or a knowledge assumption. While we use the Sub-ZK definition with an efficient subverter and extractor throughout this introduction and the paper (mainly, since it is actually more difficult to achieve), we will discuss the case of computationally unbounded subverter and extractor in Sect. 9.

In the proof of knowledge-soundness, we use (a version of) the GBGM. Using GBGM seems to be the best we can do since Groth's non-Sub zk-SNARK is proven knowledge-sound in GBGM and as mentioned above, the use of a knowledge assumption or the generic model in the knowledge-soundness proof is necessary due to the impossibility result of Gentry and Wichs [23]. However, following Bellare *et al.* [3], we weaken the usual definition of GBGM by allowing the generic adversary to create (under realistic restrictions) random elements in the source groups without knowing their discrete logarithms. We call the resulting somewhat weaker model the *subversion generic bilinear group model* (Sub-GBGM). Following Groth [27] (the main difference being that modeling a more powerful generic adversary and taking into account new CRS elements will complicate the proof somewhat), we prove that the new SNARK is (adaptively) knowledge-sound in the Sub-GBGM even in the case of type-I pairings. (We emphasize once more that Groth's zk-SNARK is proven knowledge-sound in the GBGM, and that Sub-GBGM is actually a weaker model than the GBGM.) This provides a hedge against possible future cryptanalysis that finds an efficient isomorphism between the two source groups.

Consider the case of efficient subverter. In the proof of perfect composable Sub-ZK, we use a well-known knowledge assumption (see, e.g., [17]) that we call BDH-KE. We argue that BDH-KE makes sense in the Sub-GBGM. The Sub-ZK proof of the only previously known non-interactive Sub-ZK argument system by Bellare *et al.* [3] also relies on knowledge assumptions. We follow the main idea of [3] by first using BDH-KE to extract the CRS trapdoor tc from the CRS and then construct a non-subversion simulator (that gets a part of the tc as an input) to simulate the argument. However, since we construct a zk-SNARK, our concrete approach is different from [3].

Also here, we rely on the existence of the efficient CRS verification algorithm CV. We show that if CV accepts a crs, then crs has been computed correctly by $\mathsf{K}_{\mathsf{crs}}$ from a tc bijectively fixed by crs. From this, it follows under the BDH-KE assumption that for any subverter that produces a crs accepted by CV, there exists an extractor that produces tc such that $\mathsf{K}_{\mathsf{crs}}$ given tc outputs crs.

We emphasize that our security proofs of knowledge-soundness and of Sub-ZK are in incomparable models. The knowledge-soundness proof uses the full power of Sub-GBGM in the case of any pairings (including type-I). The Sub-ZK proof, on the other hand, uses a concrete standard-looking knowledge assumption BDH-KE that holds in the the GBGM but does not hold in the Sub-GBGM in the case of type-I pairings. (In the case of computationally unbounded subverter, we even do not need BDH-KE.) This enables us to construct an efficient Sub-ZK SNARK that uses type-III pairings, while guaranteeing its knowledge-soundness even in the case of type-I pairings.

General Design Recommendations. We do not expect that constructing Sub-ZK SNARKs can be done automatically, in particular since our framework points to the necessity of making CRS publicly verifiable which means adding new elements to the CRS. Since knowledge-soundness proofs of many SNARKs are very subtle, it seems to be difficult to give a general "theorem" about which SNARKs can be modified to be Sub-ZK or even whether their CRS can be made verifiable without a major reconstruction. Whether a SNARK remains sound after that must be proven separately in each case.

However, we can still give a few recommendations for designing a Sub-ZK SNARK from a non subversion-secure SNARK (or from scratch) when using the same approach as the current paper:

1. Division of duties: make sure that K can be divided into randomized K_{tc}, deterministic K_{ts}, and deterministic $\mathsf{K}_{\mathsf{crs}}$.
2. CRS trapdoor extractability: for each element of tc, make sure that it can be extracted from the CRS. For this, one can use a generic proof of knowledge, a specific knowledge-assumption, or a computationally unbounded extractor. A few additional properties described in Sect. 7 can also help.
3. CRS verifiability: the CRS must be publicly verifiable.

4. Sound approach: make sure that the previous steps do not hurt the knowledge-soundness. To achieve it, one should aim at designing a SNARK with a very simple CRS or where CRS verifiability comes naturally. Depending on the SNARK in question, this step may be the most difficult one.

On Efficiency. Since the new zk-SNARK is closely based on the most efficient known non-subversion zk-SNARK of Groth [27], it has comparable efficiency. Importantly, the new CRS verification algorithm CV has to be executed only by the prover (this is since we achieve Sub-ZK and non-subversion knowledge-soundness). This means that it suffices for CV to have the same computational complexity as the prover's algorithm. The initial CV we describe in Fig. 1 is quite inefficient. However, as we show in Sect. 8 (see App. B for more information), by using batching techniques the CV algorithm can be sped up to be faster than the prover's algorithm for circuits of size 30 000 or more (at the information-theoretical security level 2^{-80}) and even faster at the information-theoretical security level 2^{-40}. We implemented the new SNARK by using the `libsnark` [9] library and in App. B, we back up the last claim by concrete numbers.

An interesting open question is to minimize the computational complexity of the CRS verification. In particular, in most of the known zero-knowledge argument systems, either the argument length is at least linear and hence the verification algorithm takes at least linear time (e.g., the Groth-Sahai proofs [28]) or the CRS length is at least linear and hence the CRS verification algorithm takes at least linear time. A SNARK where both CRS and the argument are succinct is called *fully succinct*. See [26,32,10,8,15] for work on zk-SNARKs with sublinear CRS. However, existing fully succinct zk-SNARK are not really practical, see [8] for discussions. Moreover, it is not clear a priori how to make it Sub-ZK. An important open research topic hence is to construct an efficient fully succinct non-interactive Sub-ZK argument system.

Related Work. Ben-Sasson *et al.* [7] proposed an efficient MPC approach to achieve security in the case one can trust at least one CRS creator. We emphasize that [3] and the current paper study the scenario where you can trust none. In such a case, the approach of [7] still works but it is not efficient. For example, the computational cost of CV (see Sect. 8) in the new SNARK is very small compared to the cost of the joint CRS creation and verification protocol in [7]. Nevertheless, while the starting point of their approach is different, it actually resulted in a somewhat similar solution. See App. A for a longer comparison, and App. B.2 for a brief comparison of the implementation results.

2 Preliminaries

For a matrix $M \in \mathbb{Z}_p^{n \times m}$, we denote by $\boldsymbol{M}_i$ the ith row of M and by $\boldsymbol{M}^{(j)}$ the jth column of M.

PPT stands for probabilistic polynomial-time. We write $f(\kappa) \approx_\kappa g(\kappa)$, if $f(\kappa) - g(\kappa)$ is negligible as a function of κ. For an algorithm A, let $\operatorname{range}(\mathsf{A})$

Algorithm 1: Computing $(\ell_i(\chi))_{i=1}^n$

```
1 $\zeta \leftarrow (\chi^n - 1)/n$; $\omega' \leftarrow 1$; $\ell_1(\chi) \leftarrow \zeta/(\chi - \omega')$;
2 **for** $i = 2$ **to** $n$ **do** $\zeta \leftarrow \omega\zeta$; $\omega' \leftarrow \omega\omega'$; $\ell_i(\chi) \leftarrow \zeta/(\chi - \omega')$ ;
```

be the range of A, i.e., the set of of valid outputs of A. For an algorithm A, let $\mathsf{RND}(\mathsf{A})$ denote the random tape of A, and let $r \leftarrow_r \mathsf{RND}(\mathsf{A})$ denote the random choice of the randomizer r from $\mathsf{RND}(\mathsf{A})$. By $y \leftarrow \mathsf{A}(\mathsf{x}; r)$ we denote the fact that A, given an input x and a randomizer r, outputs y. We emphasize that when we use this notation, then r represents the full random tape of A. For algorithms A and X_A, we write $(y \,\|\, y') \leftarrow (\mathsf{A} \,\|\, \mathsf{X}_\mathsf{A})(\chi; r)$ as a shorthand for $y \leftarrow \mathsf{A}(\chi; r)$, $y' \leftarrow \mathsf{X}_\mathsf{A}(\chi; r)$. In both cases, X_A and A use internally the same randomness r.

In the new argument system, we will use a small number of indeterminates, and we assume that each indeterminate has a canonical concrete value (a uniformly random element of $\mathbb{Z}_p$ or $\mathbb{Z}_p^*$). The canonical value of $X, X_\alpha, X_\beta, X_\gamma, X_\delta$ (resp., Y_i) will be $\chi, \alpha, \beta, \gamma, \delta$ (resp., υ_i, for $i \in \{1, 2, 3\}$). The canonical value of the vector $\boldsymbol{X} = (X, X_\alpha, X_\beta, X_\gamma, X_\delta)$ (resp., $\boldsymbol{Y} = (Y_1, Y_2, Y_3)$) will be $\boldsymbol{x} = (\chi, \alpha, \beta, \gamma, \delta)$ (resp., $\boldsymbol{y} = (\upsilon_1, \upsilon_2, \upsilon_3)$).

Lemma 1 (Schwartz-Zippel [41,39]). *Let $f \in \mathbb{F}[X_1, \dots, X_n]$ be a non-zero polynomial of total degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$, and let $x_1, \dots, x_n$ be selected at random independently and uniformly from S. Then* $\Pr[f(x_1, \dots, x_n) = 0] \leq d/|S|$.

Interpolation. Assume n is a power of two, and let ω be the n-th primitive root of unity modulo p. Such ω exists, given that $n \mid (p-1)$. Then,

- $\ell(X) := \prod_{i=1}^n (X - \omega^{i-1}) = X^n - 1$ is the unique degree n monic polynomial such that $\ell(\omega^{i-1}) = 0$ for all $i \in [1 \mathinner{..} n]$.
- For $i \in [1 \mathinner{..} n]$, let

$$\ell_i(X) := \frac{\ell(X)}{\ell'(\omega^{i-1})(X - \omega^{i-1})} = \frac{(X^n - 1)\omega^{i-1}}{n(X - \omega^{i-1})} \tag{1}$$

  be the *ith Lagrange basis polynomial*, that is, the unique degree $n-1$ polynomial such that $\ell_i(\omega^{i-1}) = 1$ and $\ell_i(\omega^{j-1}) = 0$ for $i \neq j$.

Given any $\chi \in \mathbb{Z}_p$, Alg. 1 (see, e.g., [6]) computes $\ell_i(\chi)$ for $i \in [1 \mathinner{..} n]$. It can be implemented by using $4n - 2$ multiplications and divisions in $\mathbb{Z}_p$.

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 its coefficients can thus be computed by executing an inverse Fast Fourier Transform in time $\Theta(n \log n)$. Moreover, $(\ell_j(\omega^{i-1}))_{i=1}^n = \boldsymbol{e}_j$ (the jth unit vector) and $(\ell(\omega^{i-1}))_{i=1}^n = \boldsymbol{0}_n$.

Elliptic Curves and Bilinear Maps. On input 1^κ, a *bilinear map generator* genbp returns $\mathsf{gk} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \mathfrak{g}_1, \mathfrak{g}_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are three additive cyclic groups of prime order p (with $\log p = \Omega(\kappa)$) and $\mathfrak{g}_z$ is a random generator of $\mathbb{G}_z$ for $z \in \{1, 2, T\}$. We denote the elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ by using

Fraktur as in $\mathfrak{g}_1$. Additionally, $\hat{e}$ 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: (i) $\hat{e}(\mathfrak{g}_1, \mathfrak{g}_2) \neq 1$, and (ii) $\hat{e}(a\mathfrak{g}_1, b\mathfrak{g}_2) = (ab)\hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)$.

We give genbp 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 not. Usually, arithmetic in (say) $\mathbb{G}_1$ is considerably cheaper than in $\mathbb{G}_2$; hence, we count separately exponentiations in both groups.

We use the bracket notation [18]. That is, for an integer x, we denote $[x]_z := x\mathfrak{g}_z$ even when x is unknown. We denote $[x]_1 \bullet [y]_2 := \hat{e}([x]_1, [y]_2)$, hence $[xy]_T = [x]_1 \bullet [y]_2$. We denote $[a_1, \dots, a_s]_z = ([a_1]_z, \dots, [a_s]_z)$.

The current recommendation is to use an optimal (asymmetric) Ate pairing [29] over Barreto-Naehrig curves [2,38]. In that case, at security level of $\kappa = 99$, an element of $\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$ can be represented in respectively 256/512/3072 bits.[1] To speed up interpolation and other related computation, we will need the existence of the n-th, where n is a power of 2, primitive root of unity modulo p. 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 [9].

Quadratic Arithmetic Programs. Quadratic Arithmetic Program (QAP) was introduced by Gennaro *et al.* [22] as a language where for an input x and witness w, $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$ can be verified by using a parallel quadratic check, and that has an efficient reduction from the well-known language (either Boolean or Arithmetic) Circuit-SAT. Hence, an efficient zk-SNARK for QAP results in an efficient zk-SNARK for Circuit-SAT.

For an m-dimensional vector $\boldsymbol{A}$, let $\mathsf{aug}(\boldsymbol{A}) = \left(\begin{smallmatrix} 1 \\ \boldsymbol{A} \end{smallmatrix}\right)$. For an n-dimensional vector $\boldsymbol{M}^{(0)}$ and an $n \times m$ matrix M over finite field $\mathbb{F}$, let $\mathsf{aug}(M) := (\boldsymbol{M}^{(0)}, M)$. Let $m_0 < m$ be a non-negative integer. An instance $\mathcal{Q}$ of the QAP language is specified by $(\mathbb{F}, m_0, \mathsf{aug}(U), \mathsf{aug}(V), \mathsf{aug}(W))$ where $U, V, W \in \mathbb{F}^{n \times m}$.

In the case of Arithmetic Circuit-SAT, n is the number of multiplication gates and m to the number of wires in the circuit. Here, we consider arithmetic circuits that consist only of fan-in-2 multiplication gates, but either input of each multiplication gate can be a weighted sum of some wire values, [22].

For a fixed instance $\mathcal{Q}$ of QAP, define the relation $\mathbf{R}$ as follows:

$$\mathbf{R}_{\mathcal{Q}} = \left\{ \begin{array}{l} (\mathsf{x}, \mathsf{w})\colon \mathsf{x} = (A_1, \dots, A_{m_0})^\top \wedge \mathsf{w} = (A_{m_0+1}, \dots, A_m)^\top \wedge \\ (\mathsf{aug}(U) \cdot \mathsf{aug}(\boldsymbol{A})) \circ (\mathsf{aug}(V) \cdot \mathsf{aug}(\boldsymbol{A})) = \mathsf{aug}(W) \cdot \mathsf{aug}(\boldsymbol{A}) \end{array} \right\}$$

where $\boldsymbol{a} \circ \boldsymbol{b} = (a_i b_i)_{i=1}^n$ denotes the entrywise product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

In a cryptographic setting, it is more convenient to work with the following alternative definition of QAP and of the relation $\mathbf{R}_{\mathcal{Q}}$. (This corresponds to

[1] The value $\kappa = 99$ takes account recent cryptanalysis of the Barreto-Naehrig curves by Kim and Barbulescu [31,1]. One can use different settings for 128-bit security. Since we use the library `libsnark` [9] that currently offers the mentioned security level, we just refer the reader to [31,1] for more discussion.

the original definition of QAP in [22].) Let $\mathbb{F} = \mathbb{Z}_p$, such that ω is the n-th primitive root of unity modulo p. (This requirement is needed for the sake of efficiency, and we will make it implicitly throughout the current paper.) Let $\mathcal{Q} = (\mathbb{Z}_p, m_0, \mathsf{aug}(U), \mathsf{aug}(V), \mathsf{aug}(W))$ be a QAP instance. For $j \in [0 \mathinner{..} m]$, define $u_j(X) := L_{\boldsymbol{U}^{(j)}}(X)$, $v_j(X) := L_{\boldsymbol{V}^{(j)}}(X)$, and $w_j(X) := L_{\boldsymbol{W}^{(j)}}(X)$. Thus, $u_j, v_j, w_j \in \mathbb{Z}_p^{(\leq n-1)}[X]$.

An QAP instance $\mathcal{Q}_p$ is specified by the so defined $(\mathbb{Z}_p, m_0, \{u_j, v_j, w_j\}_{j=0}^m)$. This instance defines the following relation, where we assume that $A_0 = 1$:

$$\mathbf{R}_{\mathcal{Q}_p} = \left\{ \begin{array}{l} (\mathsf{x}, \mathsf{w}) \colon \mathsf{x} = (A_1, \dots, A_{m_0})^\top \wedge \mathsf{w} = (A_{m_0+1}, \dots, A_m)^\top \wedge \\ \left(\sum_{j=0}^m A_j u_j(X)\right) \left(\sum_{j=0}^m A_j v_j(X)\right) \equiv \sum_{j=0}^m A_j w_j(X) \pmod{\ell(X)} \end{array} \right\}$$

Alternatively, $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$ if there exists a (degree $\leq n-2$) polynomial $h(X)$, s.t.

$$\left(\sum_{j=0}^m A_j u_j(X)\right) \left(\sum_{j=0}^m A_j v_j(X)\right) - \sum_{j=0}^m A_j w_j(X) = h(X)\ell(X) \enspace .$$

Clearly, $\mathbf{R}_{\mathcal{Q}} = \mathbf{R}_{\mathcal{Q}_p}$, given that $\mathcal{Q}_p$ is constructed from $\mathcal{Q}$ as above.

3 Definitions: SNARKs and Subversion Zero Knowledge

Next, we define subversion zk-SNARKs and all their properties. To achieve subversion zero knowledge (Sub-ZK), we augment a zk-SNARK by requiring the existence of an *efficient* CRS verification algorithm. As outlined in Sect. 1, we also subdivide the CRS generation algorithm into three efficient algorithms.

Our definition of (statistical unbounded) Sub-ZK for SNARKs is motivated by the definition of [3]. We also define statistical composable Sub-ZK. As in [25], the definition of unbounded zero knowledge guarantees security for any (polynomial) number of queries to the prover or simulator, while the definition of composable zero knowledge guarantees security only in the case of a single query. Following [25] we prove that a statistical composable Sub-ZK argument system is also statistical unbounded Sub-ZK.

3.1 Syntax

Let $\mathcal{R}$ be a relation generator, such that $\mathcal{R}(1^\kappa)$ returns a polynomial-time decidable binary relation $\mathbf{R} = \{(\mathsf{x}, \mathsf{w})\}$. Here, x is the statement and w is the witness. We assume that κ is explicitly deductible from the description of $\mathbf{R}$. The relation generator also outputs auxiliary information $\mathsf{z}_{\mathbf{R}}$ that will be given to the honest parties and the adversary. As in [27, Sect. 2.3], $\mathsf{z}_{\mathbf{R}}$ will be equal to $\mathsf{gk} \leftarrow \mathsf{genbp}(1^\kappa, n)$ for a well-defined n. Because of this, we will also give $\mathsf{z}_{\mathbf{R}}$ as an input to the honest parties; if needed, one can include an additional auxiliary input as an input to the adversary. We recall that the choice of p and thus of the groups $\mathbb{G}_z$ depends on n. Let $\mathcal{L}_{\mathbf{R}} = \{\mathsf{x} : \exists \mathsf{w}, (\mathsf{x}, \mathsf{w}) \in \mathbf{R}\}$ be an $\mathbf{NP}$-language.

A (subversion-resistant) *non-interactive zero-knowledge argument system Ψ* for $\mathcal{R}$ consists of seven PPT algorithms:

CRS trapdoor generator: K_{tc} is a probabilistic algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, outputs a *CRS trapdoor* tc. Otherwise, it outputs a special symbol $\perp$.

Simulation trapdoor generator: K_{ts} is a *deterministic* algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$ where $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$ and $\mathsf{tc} \in \mathrm{range}(\mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}})) \setminus \{\perp\}$, outputs the *simulation trapdoor* ts. Otherwise, it outputs $\perp$.

CRS generator: $\mathsf{K}_{\mathsf{crs}}$ is a *deterministic* algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$, where $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$ and $\mathsf{tc} \in \mathrm{range}(\mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}})) \setminus \{\perp\}$, outputs crs. Otherwise, it outputs $\perp$. For the sake of efficiency and readability, we divide crs into $\mathsf{crs}_{\mathsf{P}}$ (the part needed by the prover), $\mathsf{crs}_{\mathsf{V}}$ (the part needed by the verifier), and $\mathsf{crs}_{\mathsf{CV}}$ (the part needed only by CV and not by P or V).

CRS verifier: CV is a probabilistic algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs})$, returns either 0 (the CRS is incorrectly formed) or 1 (the CRS is correctly formed),

Prover: P is a probabilistic algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w})$ for $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$ and $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, outputs an argument π. Otherwise, it outputs $\perp$.

Verifier: V is a probabilistic algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x}, \pi)$, returns either 0 (reject) or 1 (accept).

Simulator: S is a probabilistic algorithm that, given $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$ where $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$, outputs an argument π.

We also define the (non-subverted) CRS generation algorithm $\mathsf{K}(\mathbf{R}, \mathsf{z}_{\mathbf{R}})$ that first sets $\mathsf{tc} \leftarrow \mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}})$ and then outputs $(\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow (\mathsf{K}_{\mathsf{crs}} \,\|\, \mathsf{K}_{\mathsf{ts}})(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$.

One can remove S from the definition of Ψ, and instead require that for each PPT verifier V^* there exists a corresponding PPT simulator S. We follow [27] and other SNARK literature by letting Ψ to fix S; this guarantees that there exists a single simulator that simulates the CRS for all subverters. In the case of non subversion-resistant QANIZKs, existence of a single simulator is required [30]. See Def. 5 and Rem. 1 for a longer discussion.

SNARKs. A non-interactive argument system is *succinct* if the proof size is polynomial in κ and the verifier runs in time polynomial in $\kappa + |\mathsf{x}|$. A succinct non-interactive argument of knowledge is usually called *SNARK*. A zero knowledge SNARK is abbreviated to *zk-SNARK*.

Discussions. A (non subversion-resistant) non-interactive zero-knowledge argument system is defined as a tuple $(\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathsf{S})$, see, e.g., [27]. We will now briefly motivate the differences compared to the established syntax of non-interactive zero-knowledge argument systems. Sect. 3.2 will give formal security definitions where the above syntactic definition will become an important part.

The division of K into 3 subalgorithms K_{tc}, K_{ts}, and $\mathsf{K}_{\mathsf{crs}}$ is usually not needed, but the K algorithm of many known non-interactive zero-knowledge argument systems (and all SNARKs that we know) satisfies such a division. K_{tc} just generates all randomness (tc), needed to compute crs and ts, and then $\mathsf{K}_{\mathsf{crs}}$ and K_{ts} compute from tc deterministically crs and ts. We note that such division can be formalized by requiring the crs to be witness-sampleable [30] that also seems to be a reasonable requirement in the case one is interested in subversion-resistance.

Really, an important subgoal of Sub-ZK is to guarantee that the subverted CRS is consistent with at least some choice of tc. This means that for each tc there must exist corresponding crs (accepted by CV) and ts (that can be used by the simulator to simulate subverted crs corresponding to tc).

The existence of *efficient* CV will be crucial to obtain Sub-ZK. To guarantee Sub-ZK, it is intuitively clear that the honest prover should *at least* check the correctness of the CRS. Efficiency-wise, since the prover in known SNARK constructions (including say [22,27] and the current paper) takes superlinear time, this is fine unless the CRS verification will be even slower. For the sake of clarity, however, we do not assume that CV is a part of the P *algorithm*; instead we assume that an honest prover first runs CV and given that it accepts, runs P. This is since, in practice one could execute many zero-knowledge arguments by using the same CRS; it is natural to require that the prover executes CV only once. On the other hand, since we are not aiming to get subversion (knowledge-)soundness, the honest verifier does not have to execute CRS verification.

Finally, $\mathsf{crs_P}$ (resp., $\mathsf{crs_V}$) is the part of the CRS given to an honest prover (resp., an honest verifier), and $\mathsf{crs_{CV}}$ is the part of CRS not needed by the prover or the verifier except to run CV. The distinction between $\mathsf{crs_P}$, $\mathsf{crs_V}$, and $\mathsf{crs_{CV}}$ is not important from the security point of view, but in many cases $\mathsf{crs_V}$ is significantly shorter than $\mathsf{crs_P}$. Keeping $\mathsf{crs_{CV}}$ separate helps one to evaluate better the additional efficiency penalty introduced by subversion security.

3.2 Security Definitions

A Sub-ZK SNARK has to satisfy various security definitions. The most important ones are *subversion-completeness* (an honest prover convinces an honest verifier, and an honestly generated CRS passes the CRS verification test), *computational knowledge-soundness* (if a prover convinces an honest verifier, then he knows the corresponding witness), and *statistical Sub-ZK* (given a possibly subverted CRS, an argument created by the honest prover reveals no side information). In the case of Sub-ZK, we will consider the case of an efficient subverter and a computationally unbounded distinguisher; in Sect. 9 we will discuss the case when also the subverter is unbounded. Next, we will give definitions of those properties that guarantee both composability and subversion resistance.

To keep the new security definitions as close to the accepted security definitions of zk-SNARKs as possible, we will start with non subversion-resistant security definitions from [27] (that will be stated below for the sake of completeness), albeit by using our notation, and modify them by adding elements of subversion as in Bellare *et al.* [3]. To ease the reading, we will *emphasize* the differences between non-subversion and subversion definitions. We use the division of CRS generation into three different algorithms also in the non subversion-resistant case. As motivated in Sect. 3.1, we also give $\mathsf{z_R}$ as an input to all honest parties. Finally, the notions of unbounded (ZK is guaranteed against an adversary who can arbitrarily query an oracle that outputs either proofs or simulations) and composable (ZK is guaranteed against an adversary who has to distinguish a single argument from a simulation) zero knowledge follow the definitions given in [25] and are in fact equal in our case.

As [27], we define all security notions against a non-uniform adversary. However, since our security reductions are uniform, it is a simple matter to consider only uniform adversaries, as it was done by Bellare *et al.* [3] (see also [24]).

Definition 1 (Perfect Completeness [27]). *A non-interactive argument Ψ is* perfectly complete for $\mathcal{R}$*, if for all* κ*, all* $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$*,* $\mathsf{tc} \in \mathrm{range}(\mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_\mathbf{R})) \setminus \{\perp\}$*, and* $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$*,*

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) : \mathsf{V}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{V}, \mathsf{x}, \mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w})) = 1\right] = 1 \enspace .$$

Definition 2 (Perfect Subversion-Completeness). *A non-interactive argument Ψ is* perfectly subversion-complete for $\mathcal{R}$*, if for all* κ*, all* $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$*,* $\mathsf{tc} \in \mathrm{range}(\mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_\mathbf{R})) \setminus \{\perp\}$*, and* $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$*,*

$$\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) : \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \\ \mathsf{V}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{V}, \mathsf{x}, \mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w})) = 1\end{array}\right] = 1 \enspace .$$

Definition 3 (Computational Knowledge-Soundness [27]). *Ψ is computationally (adaptively)* knowledge-sound for $\mathcal{R}$*, if for every non-uniform PPT* A*, there exists a non-uniform PPT extractor* X_A*, s.t. for all* κ*,*

$$\Pr\left[\begin{array}{l}(\mathbf{R}, \mathsf{z}_\mathbf{R}) \leftarrow \mathcal{R}(1^\kappa), (\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \mathsf{z}_\mathbf{R}), \\ r \leftarrow_r \mathsf{RND}(\mathsf{A}), ((\mathsf{x}, \pi) \,\|\, \mathsf{w}) \leftarrow (\mathsf{A} \,\|\, \mathsf{X}_\mathsf{A})(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}; r) : \\ (\mathsf{x}, \mathsf{w}) \not\in \mathbf{R} \wedge \mathsf{V}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{V}, \mathsf{x}, \pi) = 1\end{array}\right] \approx_\kappa 0 \enspace .$$

Here, $\mathsf{z}_\mathbf{R}$ can be seen as a common auxiliary input to A and X_A that is generated by using a benign [11] relation generator; we recall that we just think of $\mathsf{z}_\mathbf{R}$ as being the description of a secure bilinear group. A knowledge-sound argument system is called an *argument of knowledge.*

Next, we define statistically unbounded ZK. Unbounded (non-Sub) ZK was not defined in [27], presumably because it is a corollary of composable non-Sub ZK as shown in [25]. Hence, we will first give a modified version of the definition of non-Sub ZK from [25]; in [25], K will only output a crs and a CRS simulator $\mathsf{S}_{\mathsf{crs}}$ will return $(\mathsf{crs} \,\|\, \mathsf{ts})$. As in [27], we find it more convenient to let $\mathsf{S}_{\mathsf{crs}}$ (whom we call K) to generate also the honest crs.

Definition 4 (Statistically Unbounded ZK [25]). *Ψ is* statistically unbounded Sub-ZK for $\mathcal{R}$*, if for all* κ*, all* $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$*, and all computationally unbounded* A*,* $\varepsilon_0^{unb} \approx_\kappa \varepsilon_1^{unb}$*, where*

$$\varepsilon_b^{unb} = \Pr[(\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \mathsf{z}_\mathbf{R}) : \mathsf{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1] \enspace .$$

Here, the oracle $\mathsf{O}_0(\mathsf{x}, \mathsf{w})$ *returns* $\perp$ *(reject) if* $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$*, and otherwise it returns* $\mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w})$*. Similarly,* $\mathsf{O}_1(\mathsf{x}, \mathsf{w})$ *returns* $\perp$ *(reject) if* $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$*, and otherwise it returns* $\mathsf{S}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$*. Ψ is* perfectly unbounded Sub-ZK for $\mathcal{R}$ *if one requires that* $\varepsilon_0^{unb} = \varepsilon_1^{unb}$*.*

The following definition of unbounded Sub-ZK differs from this as follows. Since we allow the CRS to be subverted, the CRS is generated by a subverter who also returns z_Σ. The adversary's access to z_Σ models the possibility that the subverter and the adversary collaborate. The extractor X_Σ extracts tc from Σ, and then tc is used to generate the simulation trapdoor ts that is then given as an auxiliary input to the adversary and to the oracle O_1. In [25], tc was not given to the adversary because K was not required to return tc and thus was not guaranteed to exist; adding this input to the adversary only increases the power of the adversary. In our construction, it does not matter since a computationally unbounded A could compute ts herself (see Alg. 5). If we allow both the subverter and the extractor to be computationally unbounded, we would not need to rely on a knowledge assumption; the Sub-ZK proof in Sect. 7 would also simplify.

A weaker version of Sub-ZK definition would require that the extractor only outputs ts that is used then for simulation. We prefer the current definition since it is stronger and allows us to prove CRS trapdoor extractability (see Def. 8). Since we consider statistical ZK, achieving our stronger definition will concur almost no extra cost if we also allow X_Σ to be computationally unbounded.

Definition 5 (Statistically Unbounded Sub-ZK). *Ψ is* statistically unbounded Sub-ZK for $\mathcal{R}$, *if for any non-uniform PPT subverter Σ there exists a non-uniform PPT X_Σ, such that for all κ, all $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, and all computationally unbounded A, $\varepsilon_0^{unb} \approx_\kappa \varepsilon_1^{unb}$, where*

$$\varepsilon_b^{unb} = \Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\Sigma), (\mathsf{crs}, \mathsf{z}_\Sigma \parallel \mathsf{tc}) \leftarrow (\Sigma \parallel \mathsf{X}_\Sigma)(\mathbf{R}, \mathsf{z}_\mathbf{R}; r), \\ \mathsf{ts} \leftarrow \mathsf{K}_{\mathsf{ts}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) : \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \\ \mathsf{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{z}_\Sigma) = 1 \end{array}\right] .$$

Here, the oracle $\mathsf{O}_0(\mathsf{x}, \mathsf{w})$ returns $\perp$ (reject) if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}$, and otherwise it returns $\mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w})$. *Similarly, $\mathsf{O}_1(\mathsf{x}, \mathsf{w})$ returns $\perp$ (reject) if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R}$, and otherwise it returns* $\mathsf{S}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$. *$\Psi$ is* perfectly unbounded Sub-ZK for $\mathcal{R}$ *if one requires that $\varepsilon_0^{unb} = \varepsilon_1^{unb}$.*

Remark 1 (Comparison to Sub-ZK Definition of [3]). In [3], it is required that for any (non-uniform) PPT subverter Σ there exists a (non-uniform) PPT simulator $(\mathsf{X}_\Sigma, \mathsf{S})$, such that for all κ, $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, all $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, and all (non-uniform) PPT A, $\varepsilon_0^{bfs} \approx_\kappa \varepsilon_1^{bfs}$, where

$$\varepsilon_b^{bfs} = \Pr\left[\begin{array}{l} \textbf{if } b = 0 \textbf{ then } r \leftarrow_r \mathsf{RND}(\Sigma), \mathsf{crs} \leftarrow \Sigma(\mathbf{R}, \mathsf{z}_\mathbf{R}; r) \\ \textbf{else } (\mathsf{crs}, r) \leftarrow \mathsf{X}_\Sigma(\mathbf{R}, \mathsf{z}_\mathbf{R}) \textbf{ endif} : \mathsf{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, r) \end{array}\right] .$$

for O_b defined as before.

First, [3] defines computational Sub-ZK while we define statistical Sub-ZK. This by itself changes several aspects of the definition. E.g., we could just let A to compute ts and z_Σ from crs instead of giving them as extra inputs to A.

Second, compared to [3], we give to A extra information, ts and z_Σ. Having access to ts means that one can implement SNARKs for different relations using the same CRS [25]. Really, given ts, A will be able to both form and simulate arguments for any of the considered relations. Having access to z_{Σ} models, as already mentioned, the possibility that Σ and A are collaborating. (Bellare *et al.* only allow the subverter to communicate r, the secret coins.) In this sense, our definition is stronger compared to [3].

Third, Bellare *et al.* required that for each Σ there exists a simulator $(\mathsf{X}_{\Sigma}, \mathsf{S})$. We consider it to be more natural to think of X_{Σ} as an extractor and allow only X_{Σ} to depend on Σ. In the SNARK literature, extractors usually depend on the adversary (here, Σ) while there is a single simulator that works for all adversaries. In particular, this is a formal requirement in the case of QANIZKs, [30]. Also Bellare *et al.* used X_{Σ} as an extractor in their construction. In this sense, our definition is stronger compared to [3].

Fourth, we give to X_{Σ} the same coins r as to Σ, while Bellare *et al.* allow X_{Σ} to generate its own coins, only requiring that the distribution of (crs, r) is computationally indistinguishable from the output of Σ. Also in this sense, our definition seems to be stronger.

Finally, we use explicitly the syntax of subversion-resistant SNARKs from Sect. 3.1, assuming the existence of algorithms K_{ts} and CV. While it might seem to be restrictive, as argued in Sect. 3.1, existence of both K_{ts} and CV seems to be necessary for subversion-resistance. $\square$

In the case of composable ZK, the adversary can only see one purported (i.e., real or simulated) argument π, instead of being able to make many queries to a purported prover oracle as in the case of unbounded ZK. If composable ZK is defined carefully, it will be at least as strong as unbounded ZK while potentially allowing for simpler ZK proofs, [25]. We will show that the same holds in the case of Sub-ZK.

Definition 6 (Statistically Composable ZK [27]). *Ψ is* statistically composable Sub-ZK for *$\mathcal{R}$, if for any non-uniform PPT subverter Σ there exists a non-uniform PPT X_{Σ}, such that for all κ, $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^{\kappa}))$, all $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, and all computationally unbounded A, $\varepsilon_0^{comp} \approx_{\kappa} \varepsilon_1^{comp}$, where*

$$\varepsilon_b^{comp} = \Pr\left[\begin{array}{l} (\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}), \\ \textbf{if } b = 0 \textbf{ then } \pi \leftarrow \mathsf{P}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w}) \\ \textbf{else } \pi \leftarrow \mathsf{S}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \mathsf{x}) \textbf{ endif} : \mathsf{A}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \pi) = 1 \end{array}\right] .$$

Ψ is perfectly composable Sub-ZK for *$\mathcal{R}$ if one requires that $\varepsilon_0^{comp} = \varepsilon_1^{comp}$.*

Next, we define statistical composable subversion zero knowledge (Sub-ZK). This definition is related to but crucially different from the definition of (computational) composable ZK from [25]. Most importantly, [25] defines two properties, the first being *reference string indistinguishability*, meaning that the CRS generated by *honest* K and the CRS simulated by a simulator $\mathsf{S}_{\mathsf{crs}}$ should be indistinguishable. We will use the CRS generated by the subverter in both the real and the simulated case. The second property in [25] is simulation indistinguishability. Our definition of composable Sub-ZK is similar to the definition of simulation indistinguishability in [25]. However, instead of simulating the CRS, we use crs generated by the subverter, assume that an extractor extracts tc from crs, compute ts from tc by using K_{ts}, and finally verify that CV accepts crs. We also quantify over all valid $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$ instead of letting the adversary to choose it; since we deal with statistical Sub-ZK this results in an equivalent definition.

Moreover, we differ from the definition of composable non-Sub ZK in [27] as follows. The CRS is generated by Σ who also returns z_Σ. A's access to z_Σ models the possibility that Σ and A collaborate; although of course A could just recompute it. X_Σ extracts tc from Σ, and then tc is used to generate the simulation trapdoor ts that is then given as an auxiliary input to A.

Definition 7 (Statistically Composable Sub-ZK). *Ψ is* statistically composable Sub-ZK for *$\mathcal{R}$, if for any non-uniform PPT subverter Σ there exists a non-uniform PPT X_Σ, such that for all κ, all $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, all $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, and all computationally unbounded A, $\varepsilon_0^{comp} \approx_\kappa \varepsilon_1^{comp}$, where*

$$\varepsilon_b^{comp} = \Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\Sigma), (\mathsf{crs}, \mathsf{z}_\Sigma \parallel \mathsf{tc}) \leftarrow (\Sigma \parallel \mathsf{X}_\Sigma)(\mathbf{R}, \mathsf{z}_\mathbf{R}; r), \\ \mathsf{ts} \leftarrow \mathsf{K}_{\mathsf{ts}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}), \\ \textbf{if } b = 0 \textbf{ then } \pi \leftarrow \mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w}) \\ \textbf{else } \pi \leftarrow \mathsf{S}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{x}) \textbf{ endif} : \\ \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \mathsf{A}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \pi, \mathsf{z}_\Sigma) = 1 \end{array}\right] .$$

Ψ is perfectly composable Sub-ZK for *$\mathcal{R}$ if one requires that $\varepsilon_0^{comp} = \varepsilon_1^{comp}$.*

Now, we will prove the following result that makes it possible to operate in the rest of the paper with the simpler composable Sub-ZK definition. It is motivated by a similar result of Groth (ASIACRYPT 2006, [25]) that considers computational non subversion-resistant zero knowledge. As seen from below, we can establish the same result for statistical zero knowledge, but then we have to restrict the number of oracle calls to a polynomial number.

Theorem 1. *(i) Statistical composable Sub-ZK implies statistical unbounded Sub-ZK, assuming that A is given access to polynomially many oracle calls. (ii) Perfect composable Sub-ZK implies perfect unbounded Sub-ZK, even if given access to an unbounded number of oracle calls.*

Proof. (I) STATISTICAL SUB-ZK. Assume that the adversary can make up to $q(\kappa)$ oracle queries for some fixed polynomial q. We define a sequence of $q(\kappa)+1$ oracles $\mathsf{O}'_0(\mathsf{x}, \mathsf{w}), \dots, \mathsf{O}'_{q(\kappa)}(\mathsf{x}, \mathsf{w})$. Given the jth adversarial query $(\mathsf{x}_j, \mathsf{w}_j)$, the oracle $\mathsf{O}'_k(\cdot, \cdot)$ responds with $\mathsf{O}_1(\mathsf{x}_j, \mathsf{w}_j)$ for $j \in [1 \mathinner{..} k]$ and $\mathsf{O}_0(\mathsf{x}_j, \mathsf{w}_j)$ for $j \in [k+1 \mathinner{..} q(\kappa)]$. Hence, $\mathsf{O}'_0 = \mathsf{O}_0$ and $\mathsf{O}'_{q(\kappa)} = \mathsf{O}_1$.

Due to the statistical composable Sub-ZK property, we get that for $i \in [0 \mathinner{..} q(\kappa) - 1]$, $\varepsilon_i \approx_\kappa \varepsilon_{i+1}$, where

$$\varepsilon_i = \Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\Sigma), (\mathsf{crs}, \mathsf{z}_\Sigma \parallel \mathsf{tc}) \leftarrow (\Sigma \parallel \mathsf{X}_\Sigma)(\mathbf{R}, \mathsf{z}_\mathbf{R}; r), \\ \mathsf{ts} \leftarrow \mathsf{K}_{\mathsf{ts}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) : \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \\ \mathsf{A}^{\mathsf{O}'_i(\cdot,\cdot)}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{z}_\Sigma) = 1 \end{array}\right]$$

since the oracles can be efficiently implemented given crs and ts, and inserting π as the answer to the ith query. Since $\varepsilon_0 = \varepsilon_0^{unb}$, q is a polynomial, and $\varepsilon_0^{unb} = \varepsilon_0 \approx_\kappa \cdots \approx_\kappa \varepsilon_{q(\kappa)} = \varepsilon_1^{unb}$, we get that $\varepsilon_0^{unb} \approx_\kappa \varepsilon_1^{unb}$ and hence the claim holds.

(II) PERFECT SUB-ZK. As above, but assume q is the actual (possibly unbounded) number of queries. We get $\varepsilon_0^{unb} = \varepsilon_0 = \cdots = \varepsilon_{q(\kappa)} = \varepsilon_1^{unb}$, and hence $\varepsilon_0^{unb} = \varepsilon_1^{unb}$, and the claim holds. □

In [25], composable ZK was a stronger requirement than unbounded ZK since in the case of composable ZK, (i) the simulated CRS was required to be indistinguishable from the real CRS, and (ii) the adversary got access to ts. In the case of our Sub-ZK definitions, there is no such difference and it is easy to see that composable Sub-ZK and unbounded Sub-ZK are in fact equal notions.

In the proof that the new SNARK is Sub-ZK (see Thm. 5), we require that if $\mathsf{\Sigma}$ generates a crs accepted by CV then there exists a non-uniform PPT extractor $\mathsf{X}_\mathsf{\Sigma}$ that generates a CRS trapdoor tc that corresponds to crs; that is, K_crs on input tc outputs crs. We formalize this property — that intuitively gives us a stronger version of witness-sampleability — as follows.

Definition 8 (Statistical CRS Trapdoor Extractability). *Ψ has* statistical CRS trapdoor extractability for $\mathcal{R}$, *if for any non-uniform PPT subverter $\mathsf{\Sigma}$ there exists a non-uniform PPT $\mathsf{X}_\mathsf{\Sigma}$, such that for all κ and all $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$,*

$$\Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\mathsf{\Sigma}), (\mathsf{crs}, \mathsf{z}_\mathsf{\Sigma} \,\|\, \mathsf{tc}) \leftarrow (\mathsf{\Sigma} \,\|\, \mathsf{X}_\mathsf{\Sigma})(\mathbf{R}, \mathsf{z}_\mathbf{R}; r) : \\ \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \mathsf{K}_\mathsf{crs}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) \neq \mathsf{crs} \end{array}\right] \approx_\kappa 0 \ .$$

Ψ has perfect CRS trapdoor extractability for $\mathcal{R}$ *if the same property holds but with $\approx_\kappa$ changed to $=$.*

4 GBGM And Sub-GBGM

Preliminaries: Generic Bilinear Group Model. In Sect. 6, we will prove that the new zk-SNARK is knowledge-sound in the subversion generic bilinear group model (Sub-GBGM). In the current subsection, we will introduce the GBGM [36,40,35,13], by following the exposition in [35]. After that, we will introduce the Sub-GBGM.

We start by picking a random asymmetric bilinear group $\mathsf{gk} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathsf{genbp}(1^\kappa, n)$. Consider a black box $\mathbf{B}$ that can store values from additive groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ in internal state variables $\mathsf{cell}_1, \mathsf{cell}_2, \ldots$, where for simplicity we allow the storage space to be infinite (this only increases the power of a generic adversary). The initial state consists of some values $(\mathsf{cell}_1, \mathsf{cell}_2, \ldots, \mathsf{cell}_{|inp|})$, which are set according to some probability distribution. Each state variable cell_i has an accompanying type $\mathsf{type}_i \in \{1, 2, T, \perp\}$. We assume initially $\mathsf{type}_i = \perp$ for $i > |inp|$. The black box allows computation operations on internal state variables and queries about the internal state. No other interaction with $\mathbf{B}$ is possible.

Let Π be an allowed set of computation operations. A computation operation consists of selecting a (say, t-ary) operation $f \in \Pi$ together with $t+1$ indices $i_1, i_2, \ldots, i_{t+1}$. Assuming inputs have the correct type, $\mathbf{B}$ computes $f(\mathsf{cell}_{i_1}, \ldots, \mathsf{cell}_{i_t})$ and stores the result in $\mathsf{cell}_{i_{t+1}}$. For a set Σ of relations, a query consists of selecting a (say, t-ary) relation $\varrho \in \Sigma$ together with t indices $i_1, i_2, \ldots, i_t$. Assuming inputs have the correct type, $\mathbf{B}$ replies to the query with $\varrho(\mathsf{cell}_{i_1}, \ldots, \mathsf{cell}_{i_t})$. In the GBGM, we define $\Pi = \{+, \hat{e}\}$ and $\Sigma = \{=\}$, where

1. On input $(+, i_1, i_2, i_3)$: if $\mathsf{type}_{i_1} = \mathsf{type}_{i_2} \neq \perp$ then set $\mathsf{cell}_{i_3} \leftarrow \mathsf{cell}_{i_1} + \mathsf{cell}_{i_2}$ and $\mathsf{type}_{i_3} \leftarrow \mathsf{type}_{i_1}$.
2. On input $(\hat{e}, i_1, i_2, i_3)$: if $\mathsf{type}_{i_1} = 1$ and $\mathsf{type}_{i_2} = 2$ then set $\mathsf{cell}_{i_3} \leftarrow \hat{e}(\mathsf{cell}_{i_1}, \mathsf{cell}_{i_2})$ and $\mathsf{type}_{i_3} \leftarrow T$.
3. On input $(=, i_1, i_2)$: if $\mathsf{type}_{i_1} = \mathsf{type}_{i_2} \neq \perp$ and $\mathsf{cell}_{i_1} = \mathsf{cell}_{i_2}$ then return 1. Otherwise return 0.

Since we are proving lower bounds, we will give a generic adversary A additional power. We assume that all relation queries are for free. We also assume that A is successful if after τ operation queries, he makes an equality query $(=, i_1, i_2)$, $i_1 \neq i_2$, that returns 1; at this point A quits. Thus, if $\mathsf{type}_i \neq \perp$, then $\mathsf{cell}_i = F_i(\mathsf{cell}_1, \ldots, \mathsf{cell}_{|inp|})$ for a polynomial F_i known to A.

Sub-GBGM. By following Bellare *et al.* [3], we enhance the power of generic bilinear group model [36,40,35,13]. Since the power of the generic adversary will increase, security proofs in the resulting *Sub-GBGM* will be somewhat more realistic than in the GBGM model.

As noted by Bellare *et al.* [3], it is known how to hash into elliptic curves and thus create group elements without knowing their discrete logarithms. However, it is not known how to create four elements $[1]_z$, $[a]_z$, $[b]_z$, and $[ab]_z$ without knowing either a or b. The corresponding assumption — that may also be true in the case of symmetric pairings — was named *DH-KE(A)* in [3].

However, asymmetric pairings are much more efficient than symmetric pairings. If we work in the type III pairing setting [20] where there is no efficient isomorphism either from $\mathbb{G}_1$ to $\mathbb{G}_2$ or from $\mathbb{G}_2$ to $\mathbb{G}_1$, then clearly an adversary cannot, given $[a]_z$ for $z \in \{1, 2\}$ and an unknown a, compute $[a]_{3-z}$. In the same vein, it seems reasonable to make a stronger assumption (that we call *BDH-KE*, a simplification of the asymmetric PKE assumption of [17]) that if an adversary creates $[a]_1$ and $[a]_2$ then she knows a. Really, since there is no polynomial-time isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ (or back), it seems to be natural to assume that one does not have to worry about an adversary knowing some trapdoor that would break the BDH-KE assumption. Since BDH-KE is not a falsifiable assumption, this does not obviously mean that it must hold for each type III pairing. Instead, the BDH-KE assumption can be interpreted as a *stronger* definition of the type III pairing setting. We formalize the added adversarial power as follows.

We give the generic model adversary an additional power to effectively create new indeterminates Y_i in groups $\mathbb{G}_1$ and $\mathbb{G}_2$ (e.g., by hashing into elliptic curves), without knowing their values. We note since $[Y]_1 \bullet [1]_2 = [Y]_T$ and $[1]_1 \bullet [Y]_2 = [Y]_T$, the adversary that has generated an indeterminate Y in $\mathbb{G}_z$ can also operate with Y in $\mathbb{G}_T$. Formally, this means that Π will contain one more operation create, with the following semantics:

4. On input (create, i, t): if $\mathsf{type}_i = \perp$ and $t \in \{1, 2, T\}$ then set $\mathsf{cell}_i \leftarrow \mathbb{Z}_p$ and $\mathsf{type}_i \leftarrow t$.

The semantics of create dictates that the actual value of the indeterminate Y_i is uniformly random in $\mathbb{Z}_p$, that is, the adversary cannot create indeterminates for which she does not know the discrete logarithm and that yet are not random. This assumption is needed for the lower bound on the generic adversary's time to be provable in Thm. 3.

In the type III setting [20], this semantics does *not* allow the adversary to create the same indeterminate Y_i in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$; she can only create a representation of a known to her integer in both groups. We formalize this by making the following Bilinear Diffie-Hellman Knowledge of Exponents (*BDH-KE*) assumption: if the adversary, given random generators $\mathfrak{g}_1 = [1]_1 \in \mathbb{G}_1$ and $\mathfrak{g}_2 = [1]_2 \in \mathbb{G}_2$, can generate elements $[\alpha_1]_1 \in \mathbb{G}_1$ and $[\alpha_2]_2 \in \mathbb{G}_2$, such that $[1]_1 \bullet [\alpha_2]_2 = [\alpha_1]_1 \bullet [1]_2$, then the adversary knows the value $\alpha_1 = \alpha_2$. To simplify the further use of the BDH-KE assumption in security reductions, we give the adversary access to $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$. As before, $\mathsf{z}_\mathbf{R}$ just contains gk, that is, the description of the bilinear group together with $[1]_1$ and $[1]_2$.

Definition 9 (BDH-KE). *We say that* genbp *is* BDH-KE secure *for* $\mathcal{R}$ *if for any* κ*,* $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$*, and non-uniform PPT adversary* A *there exists a non-uniform PPT extractor* X_A*, such that*

$$\Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\mathsf{A}), ([\alpha_1]_1, [\alpha_2]_2 \,\|\, a) \leftarrow (\mathsf{A} \,\|\, \mathsf{X}_\mathsf{A})(\mathbf{R}, \mathsf{z}_\mathbf{R}; r) : \\ [\alpha_1]_1 \bullet [1]_2 = [1]_1 \bullet [\alpha_2]_2 \wedge a \neq \alpha_1 \end{array}\right] \approx_\kappa 0 \enspace .$$

The BDH-KE assumption is a simple special case of the PKE assumption as used in the case of asymmetric pairings say in [17]. In the PKE assumption of [17], adversary is given as an input the tuple $\{([\chi^i]_1, [\chi^i]_2)\}_{i=0}^n$ for some $n \geq 0$, and it is assumed that if an adversary outputs $([\alpha]_1, [\alpha]_2)$ then she knows $(a_0, a_1, \ldots, a_n)$, such that $\alpha = \sum_{i=0}^n a_i \chi^i$. In our case, $n = 0$. BDH-KE can also be seen as an asymmetric-pairing version of the original KE assumption [16].

We think that for the following reasons, the BDH-KE assumption is more natural than the DH-KE assumption by Bellare *et al.* [3] which states that if the adversary can create elements $[\alpha_1]_z$, $[\alpha_2]_z$ and $[\alpha_1\alpha_2]_z$ of the group $\mathbb{G}_z$ then she knows either α_1 or α_2.

First, the BDH-KE assumption is well suited to type-III pairings that are by far the most efficient pairings. The DH-KE assumption is tailored to type-I pairings. In the case of type-III pairings, DH-KE assumption can still be used, but it results in inefficient protocols. For example in [3], in security proofs the authors employs an adversary that extracts either α_1 or α_2. Since it is not known a priori which value will be extracted, several elements in the argument system have to be doubled, for the case α_1 is extracted and for the case α_2 is extracted.

Second, most of the efficient SNARKs are constructed to be sound and zero-knowledge in the (most efficient) type-III setting. While the SNARK of

Groth [27] is known to be sound in the case of both symmetric and asymmetric pairings, in the case of symmetric pairings it will be much less efficient. To take the advantage of already known efficient SNARKs, it is only natural to keep the type-III setting. In the current paper, we are able to have the best of both worlds. As in the case of [27], we construct a SNARK that uses type-III pairings. On the one hand, we prove it to be Sub-ZK solely under the BDH-KE assumption. On the other hand, we prove that it is (adaptively) knowledge-sound in the Sub-GBGM, independently of whether one uses type-I, type-II, or type-III pairings. This provides a partial hedge against cryptanalysis: even if one were to later find an efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$, this would only break the Sub-ZK of the new SNARK but leave the soundness property intact.

5 New Sub-ZK Secure SNARK

Consider a QAP instance $\mathcal{Q}_p = (\mathbb{Z}_p, m_0, \{u_j, v_j, w_j\}_{j=0}^m)$. The goal of the prover of a QAP argument of knowledge [22,27] is to show that for public $(A_1, \dots, A_{m_0})$ and $A_0 = 1$, the prover knows $(A_{m_0+1}, \dots, A_m)$ and a degree $\leq n-2$ polynomial $h(X)$, such that

$$h(X) = \frac{a(X)b(X) - c(X)}{\ell(X)} \ , \qquad (2)$$

where $a(X) = \sum_{j=0}^m A_j u_j(X)$, $b(X) = \sum_{j=0}^m A_j v_j(X)$, $c(X) = \sum_{j=0}^m A_j w_j(X)$.

5.1 Construction

Next, we describe a Sub-ZK SNARK for $\mathcal{R}$ that is closely based on the (non subversion-resistant) SNARK by Groth from EUROCRYPT 2016 [27]. See Fig. 1 and Fig. 2. As always, we assume implicitly that each algorithm checks that their inputs belong to correct groups and that $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$.

This SNARK uses crucially several random variables, $\chi, \alpha, \beta, \gamma, \delta$. As in [27], α and β (and the inclusion of $\alpha\beta$ in the verification equation) will guarantee that $\mathfrak{a}$, $\mathfrak{b}$, and $\mathfrak{c}$ are computed by using the same coefficients A_i. The role of γ and δ is to make the three products in the verification equation "independent" of each other. Due to the lack of space, we omit a more precise intuition behind Groth's SNARK and refer an interested reader to [27].

As emphasized before, the new Sub-ZK SNARK is closely based on Groth's zk-SNARK. In fact, the differences between the construction of the two SNARKs can be summarized very briefly:

(i) We add to the CRS $2n + 3$ new elements (see the variable $\mathsf{crs}_{\mathsf{CV}}$ in Fig. 1) that are needed for CV to work efficiently.

(ii) We divide the CRS generation algorithm into three algorithms, K_{tc}, K_{ts}, and $\mathsf{K}_{\mathsf{crs}}$. Groth's CRS generation algorithm returns $\mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}))$ (minus the mentioned $\mathsf{crs}_{\mathsf{CV}}$ part) as the CRS and $\mathsf{K}_{\mathsf{ts}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}))$ as the simulation trapdoor.

(iii) We describe an efficient CRS verification algorithm CV (see Fig. 1).

$\mathsf{K_{tc}}(\mathbf{R}, \mathsf{z_R})$**:** Generate $\mathsf{tc} = (\chi, \alpha, \beta, \gamma, \delta) \leftarrow_r \mathbb{Z}_p^3 \times (\mathbb{Z}_p^*)^2$.
$\mathsf{K_{ts}}(\mathbf{R}, \mathsf{z_R}, \mathsf{tc})$**:** Set $\mathsf{ts} \leftarrow (\chi, \alpha, \beta, \delta)$.
$\mathsf{K_{crs}}(\mathbf{R}, \mathsf{z_R}, \mathsf{tc})$**:** Compute $(\ell_i(\chi))_{i=1}^n$ by using Alg. 1. Set $u_j(\chi) \leftarrow \sum_{i=1}^n U_{ij}\ell_i(\chi)$, $v_j(\chi) \leftarrow \sum_{i=1}^n V_{ij}\ell_i(\chi)$, $w_j(\chi) \leftarrow \sum_{i=1}^n W_{ij}\ell_i(\chi)$ for all $j \in \{0, \dots, m\}$. Let

$$\mathsf{crs_P} \leftarrow \left(\begin{matrix} \left[\alpha, \beta, \delta, \left(\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\delta}\right)_{j=m_0+1}^m\right]_1 , \\ \left[(\chi^i \ell(\chi)/\delta)_{i=0}^{n-2}, (u_j(\chi), v_j(\chi))_{j=0}^m\right]_1, \left[\beta, \delta, (v_j(\chi))_{j=0}^m\right]_2 \end{matrix} \right) ,$$

$$\mathsf{crs_V} \leftarrow \left(\left[\left(\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\gamma}\right)_{j=0}^{m_0}\right]_1, [\gamma, \delta]_2, [\alpha\beta]_T \right) ,$$

$$\mathsf{crs_{CV}} \leftarrow \left(\left[\gamma, (\chi^i)_{i=1}^{n-1}, (\ell_i(\chi))_{i=1}^n\right]_1, \left[\alpha, \chi, \chi^{n-1}\right]_2 \right) .$$

Return $\mathsf{crs} \leftarrow (\mathsf{crs_{CV}}, \mathsf{crs_P}, \mathsf{crs_V})$.
$\mathsf{K}(\mathbf{R}, \mathsf{z_R})$**:** Let $\mathsf{tc} \leftarrow \mathsf{K_{tc}}(\mathbf{R}, \mathsf{z_R})$. Return $(\mathsf{crs} \parallel \mathsf{ts}) \leftarrow (\mathsf{K_{crs}} \parallel \mathsf{K_{ts}})(\mathbf{R}, \mathsf{z_R}, \mathsf{tc})$.
$\mathsf{CV}(\mathbf{R}, \mathsf{z_R}, \mathsf{crs})$**:**

1. For $\iota \in \{\gamma, \delta\}$: check that $[\iota]_1 \neq [0]_1$
2. For $\iota \in \{\alpha, \beta, \gamma, \delta\}$: check that $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [\iota]_2$,
3. For $i = 1$ to $n - 1$: check that $\left[\chi^i\right]_1 \bullet [1]_2 = \left[\chi^{i-1}\right]_1 \bullet [\chi]_2$,
4. Check that $([\ell_i(\chi)]_1)_{i=1}^n$ is correctly computed by using Alg. 2,
5. For $j = 0$ to m:
   (a) Check that $[u_j(\chi)]_1 = \sum_{i=1}^n U_{ij} [\ell_i(\chi)]_1$,
   (b) Check that $[v_j(\chi)]_1 = \sum_{i=1}^n V_{ij} [\ell_i(\chi)]_1$,
   (c) Set $[w_j(\chi)]_1 \leftarrow \sum_{i=1}^n W_{ij} [\ell_i(\chi)]_1$,
   (d) Check that $[v_j(\chi)]_1 \bullet [1]_2 = [1]_1 \bullet [v_j(\chi)]_2$,
6. For $j = 0$ to m_0: check that $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\gamma]_1 \bullet [\gamma]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$,
7. For $j = m_0 + 1$ to m: check that $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta]_1 \bullet [\delta]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$,
8. Check that $\left[\chi^{n-1}\right]_1 \bullet [1]_2 = [1]_1 \bullet \left[\chi^{n-1}\right]_2$,
9. For $i = 0$ to $n-2$: check that $\left[\chi^i \ell(\chi)/\delta\right]_1 \bullet [\delta]_2 = \left[\chi^{i+1}\right]_1 \bullet \left[\chi^{n-1}\right]_2 - \left[\chi^i\right]_1 \bullet [1]_2$,
10. Check that $[\alpha]_1 \bullet [\beta]_2 = [\alpha\beta]_T$.

Fig. 1. The CRS generation and verification of the Sub-ZK SNARK for $\mathcal{R}$

It is straightforward to see that Groth's original zk-SNARK does not achieve Sub-ZK. Really, since neither $[\ell_i(\chi)]_1$ nor $\left[\chi^i\right]_1$ are given to the prover, he cannot check the correctness of $[u_j(\chi)]_1$ and $[v_j(\chi)]_1$. This means that a subverter can change those values to some bogus values and due to that, the proof computed by an honest prover and a simulated proof (that relies on the knowledge of trapdoor elements α and β and does not use the CRS elements $[u_j(\chi)]_1$ and $[v_j(\chi)]_1$; see Fig. 3) will have different distributions.

We prove the completeness of the new SNARK in the rest of this section. We postpone the full proof of knowledge-soundness to Sect. 6 and of zero knowledge to Sect. 7. We analyze the efficiency of this SNARK in Sect. 8.

```
$\mathsf{P}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x} = (A_1, \ldots, A_{m_0}), \mathsf{w} = (A_{m_0+1}, \ldots, A_m))$:
    /* After executing CV & assuming $A_0 = 1$, the prover does:    */
    1. Let $a^\dagger(X) \leftarrow \sum_{j=0}^m A_j u_j(X)$, $b^\dagger(X) \leftarrow \sum_{j=0}^m A_j v_j(X)$,
    2. Let $c^\dagger(X) \leftarrow \sum_{j=0}^m A_j w_j(X)$,
    3. Set $h(X) = \sum_{i=0}^{n-2} h_i X^i \leftarrow (a^\dagger(X) b^\dagger(X) - c^\dagger(X))/\ell(X)$,
    4. Set $[h(\chi)\ell(\chi)/\delta]_1 \leftarrow \sum_{i=0}^{n-2} h_i \left[\chi^i \ell(\chi)/\delta\right]_1$,
    5. Set $r_a \leftarrow_r \mathbb{Z}_p$; Set $\mathfrak{a} \leftarrow \sum_{j=0}^m A_j [u_j(\chi)]_1 + [\alpha]_1 + r_a [\delta]_1$,
    6. Set $r_b \leftarrow_r \mathbb{Z}_p$; Set $\mathfrak{b} \leftarrow \sum_{j=0}^m A_j [v_j(\chi)]_2 + [\beta]_2 + r_b [\delta]_2$,
    7. Set $\mathfrak{c} \leftarrow r_b \mathfrak{a} + r_a \left(\sum_{j=0}^m A_j [v_j(\chi)]_1 + [\beta]_1\right) +$
           $\sum_{j=m_0+1}^m A_j [(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta]_1 + [h(\chi)\ell(\chi)/\delta]_1$,
    8. Return $\pi \leftarrow (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$.
$\mathsf{V}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x} = (A_1, \ldots, A_{m_0}), \pi = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c}))$: assuming $A_0 = 1$, check that
```

$$\mathfrak{a} \bullet \mathfrak{b} = \mathfrak{c} \bullet [\delta]_2 + \left(\sum_{j=0}^{m_0} A_j \left[\frac{u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)}{\gamma}\right]_1\right) \bullet [\gamma]_2 + [\alpha\beta]_T \quad .$$

Fig. 2. The prover and the verifier of the Sub-ZK SNARK for $\mathcal{R}$ (unchanged from Groth's zk-SNARK)

Algorithm 2: Checking that $[(\ell_i(\chi))_{i=1}^n]_1$ is correctly computed

```
  **Input**: $([\chi^{n-1}, (\ell_i(\chi))_{i=1}^n]_1, [1, \chi]_2, [1]_T)$
  // $i = 1$
1 $[\zeta]_T \leftarrow ([\chi^{n-1}]_1 \bullet [\chi]_2 - [1]_T)/n$; $[\omega']_2 \leftarrow [1]_2$;
2 Check that $[\ell_1(\chi)]_1 \bullet ([\chi]_2 - [\omega']_2) = [\zeta]_T$;
3 for $i = 2$ to $n$ do
4     $[\zeta]_T \leftarrow \omega [\zeta]_T$; $[\omega']_2 \leftarrow \omega [\omega']_2$;
5     Check that $[\ell_j(\chi)]_1 \bullet ([\chi]_2 - [\omega']_2) = [\zeta]_T$;
```

5.2 Subversion Completeness

In the proof of subversion-completeness, we need the following result that hopefully has independent interest.

Lemma 2. *Given $([\chi^{n-1}, (\ell_i(\chi))_{i=1}^n]_1, [1, \chi]_2, [1]_T)$ as an input, Alg. 2 checks that $[\ell_i(\chi)]_1$ has been correctly computed for all $i \in [1 \,..\, n]$. It can be implemented by using $n+1$ pairings, $n-1$ exponentiations in $\mathbb{G}_2$, and $n-1$ exponentiations in $\mathbb{G}_T$.*

Proof. Recalling that $\ell_i(X) = (X^n - 1)\omega^{i-1}/(n(X - \omega^{i-1}))$ are as given by Eq. (1), the proof is straightforward. Really, by induction on i, at any concrete value of i, $\zeta = (\chi^n - 1)\omega^{i-1}/n$ and $\omega' = \omega^{i-1}$. Thus $[\ell_i(\chi)]_1 \bullet ([\chi]_2 - [\omega']_2) = [(\chi - \omega^{i-1}) \cdot \ell_i(\chi)]_T = [\zeta]_T$ iff $[\ell_i(\chi)]_1$ was correctly computed. □

For $z \in \{1, 2\}$, let crs_z be the subset of the (honest or subverted) crs that consists of all elements of $\mathbb{G}_z$.

Theorem 2. *The new SNARK of Sect. 5.1 is perfectly subversion-complete.*

Proof. CV ACCEPTS AN HONESTLY GENERATED CRS: this can be established by a simple but tedious calculation. Basically, CV accepts due to the properties of the bilinear map, and due to the definitions of $\ell(X)$ and $\ell_i(X)$. We only prove the correctness of two of the least obvious steps.

Step 6 of CV holds since $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\gamma]_1 \bullet [\gamma]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$ iff $[(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\gamma \cdot \gamma]_T = [u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)]_T$ which is a tautology.

Similarly, Lem. 2 has established that Alg. 2 works correctly. Thus, Step 9 of CV holds since $\left[\chi^i\ell(\chi)/\delta\right]_1 \bullet [\delta]_2 = \left[\chi^{i+1}\right]_1 \bullet \left[\chi^{n-1}\right]_2 - \left[\chi^i\right]_1 \bullet [1]_2$ iff $\left[\chi^i\ell(\chi)/\delta \cdot \delta\right]_T = \left[\chi^{i+1} \cdot \chi^{n-1} - \chi^i\right]_T$ iff $\left[\chi^i\ell(\chi)\right]_T = \left[\chi^i(\chi^n - 1)\right]_T$, which is a tautology since $\ell(\chi) = \chi^n - 1$.

V ACCEPTS AN HONESTLY GENERATED ARGUMENT: In the honest case, see Fig. 2, $\mathfrak{a} = [A(\boldsymbol{x})]_1$, $\mathfrak{b} = [B(\boldsymbol{x})]_2$, and $\mathfrak{c} = [C(\boldsymbol{x})]_1$, where $A(\boldsymbol{x}) = \sum_{j=0}^{m} A_j u_j(\chi) + \alpha + r_a\delta$, $B(\boldsymbol{x}) = \sum_{j=0}^{m} A_j v_j(\chi) + \beta + r_b\delta$, and $C(\boldsymbol{x}) = r_b A(\boldsymbol{x}) + r_a(B(\boldsymbol{x}) - r_b\delta) + \sum_{j=m_0+1}^{m}(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta + h(\chi)\ell(\chi)/\delta$. Clearly,

$$\begin{aligned} A(\boldsymbol{x})B(\boldsymbol{x}) &- r_b A(\boldsymbol{x})\delta - r_a(B(\boldsymbol{x}) - r_b\delta)\delta \\ &= \left(\textstyle\sum_{j=0}^{m} A_j u_j(\chi) + \alpha\right) \cdot \left(\textstyle\sum_{j=0}^{m} A_j v_j(\chi) + \beta\right) \\ &= \left(\textstyle\sum_{j=0}^{m} A_j u_j(\chi)\right) \cdot \left(\textstyle\sum_{j=0}^{m} A_j v_j(\chi)\right) + \alpha \textstyle\sum_{j=0}^{m} A_j v_j(\chi) + \\ &\quad \beta \textstyle\sum_{j=0}^{m} A_j u_j(\chi) + \alpha\beta \ . \end{aligned}$$

Let $V(\boldsymbol{x}) = A(\boldsymbol{x})B(\boldsymbol{x}) - C(\boldsymbol{x})\delta - \sum_{j=0}^{m_0} A_j(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)) - \alpha\beta$. Hence, $V(\boldsymbol{x}) = a^\dagger(\chi)b^\dagger(\chi) - c^\dagger(\chi) - h(\chi)\ell(\chi)$. Due to the definition of $h(\boldsymbol{X})$, $V(\boldsymbol{x}) = 0$. Thus, $[V(\boldsymbol{x})]_T = [0]_T$, which is what the verification equation ascertains. $\square$

6 Proof of Knowledge-Soundness

Since we are proving knowledge-soundness (and not subversion knowledge-soundness), the following security proof is similar to the corresponding proof in [27]. The main difference is in the need to take into account added elements to the CRS and to incorporate new indeterminates Y_i created by the adversary. This means that the proof will be slightly (but not much) more complicated than the proof in [27].

Next, we will prove adaptive knowledge-soundness even in the case when one uses symmetric pairings, as it was done in [27]. The use of symmetric pairings means that the generic adversary gets additional power compared to Sub-GBGM: in particular, we do not assume that BDH-KE holds, and moreover, in the asymmetric variant of the following theorem, the generic adversary will be allowed to create new indeterminates Y_i in both $\mathbb{G}_1$ and $\mathbb{G}_2$, without knowing their discrete logarithms. Since this only increases the power of the generic adversary, it provides some hedge against future cryptanalytic attacks that say make it possible to compute an efficient isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$, or at least show that BDH-KE does not hold in the concrete groups.

Theorem 3 (Knowledge-soundness). *Consider the new argument system of Sect. 5.1. It is adaptively knowledge-sound in the Sub-GBGM even in the case of symmetric pairings. More precisely, any generic adversary attacking the knowledge-soundness of the new argument system in the symmetric setting requires $\Omega(\sqrt{p/n})$ computation.*

Proof. Assume the symmetric setting, $\mathbb{G}_1 = \mathbb{G}_2$. Let $\boldsymbol{X}$ be the vector of indeterminates created by $\mathsf{K_{tc}}$ and $\boldsymbol{Y} = (Y_1, \dots, Y_q)^\top$ (for a non-negative integer q) be the vector of indeterminates created by the generic adversary.

The three elements output by a generic adversary are equal to $\mathfrak{a} = [A(\boldsymbol{x}, \boldsymbol{y})]_1$, $\mathfrak{b} = [B(\boldsymbol{x}, \boldsymbol{y})]_1$, and $\mathfrak{c} = [C(\boldsymbol{x}, \boldsymbol{y})]_1$, where, for $T \in \{A, B, C\}$,

$$\begin{aligned} T(\boldsymbol{X}, \boldsymbol{Y}) =& T_\alpha X_\alpha + T_\beta X_\beta + T_\gamma X_\gamma + T_\delta X_\delta + T_c(X) + \\ & \textstyle\sum_{j=0}^{m_0} T_j \cdot \frac{u_j(X) X_\beta + v_j(X) X_\alpha + w_j(X)}{X_\gamma} + \\ & \textstyle\sum_{j=m_0+1}^{m} T_j \cdot \frac{u_j(X) X_\beta + v_j(X) X_\alpha + w_j(X)}{X_\delta} + \frac{T_h(X)\ell(X)}{X_\delta} + \sum_{i=1}^{q} T_{yi} Y_i \enspace . \end{aligned}$$

Here, $T_c(X)$ is a degree $\leq n - 1$ and $T_h(X)$ is a degree $\leq n - 2$ polynomial. Thus, $T(\boldsymbol{X}, \boldsymbol{Y}) \cdot X_\gamma X_\delta$ is a degree $(n - 2) + n - 1 + 2 = 2n - 1$ polynomial.

Since the only difference compared to the knowledge-soundness proof in [27] is in the addition of the terms $\sum_{i=1}^{q} T_{yi} Y_i$ to those three polynomials, it suffices that we show that $T_{yi} = 0$ for $T \in \{A, B, C\}$ and $i \in [1 \mathinner{..} q]$. After that, the knowledge-soundness of the new SNARK follows from the knowledge-soundness of Groth's SNARK.

Motivated by the verification equation in Fig. 2, define

$$\begin{aligned} V(\boldsymbol{X}, \boldsymbol{Y}) :=& A(\boldsymbol{X}, \boldsymbol{Y}) B(\boldsymbol{X}, \boldsymbol{Y}) - C(\boldsymbol{X}, \boldsymbol{Y}) X_\delta - \\ & \textstyle\sum_{j=0}^{m_0} A_j \left(u_j(X) X_\beta + v_j(X) X_\alpha + w_j(X)\right) - X_\alpha X_\beta \end{aligned} ,$$

where the Laurent polynomials $A(\boldsymbol{X}, \boldsymbol{Y})$, $B(\boldsymbol{X}, \boldsymbol{Y})$, and $C(\boldsymbol{X}, \boldsymbol{Y})$ are as given before. The verification equation states that $[V(\boldsymbol{x}, \boldsymbol{y})]_T = [0]_T$ and hence in the case of a generic adversary, $V(\boldsymbol{X}, \boldsymbol{Y}) \cdot X_\gamma^2 X_\delta^2 = 0$ as a polynomial or equivalently, each of the coefficients of $V(\boldsymbol{X}, \boldsymbol{Y})$ is 0.

First, the coefficient of X_α^2 in $V(\boldsymbol{X}, \boldsymbol{Y})$ is $A_\alpha B_\alpha$. Thus, from $V(\boldsymbol{X}, \boldsymbol{Y}) = 0$ it follows that $A_\alpha B_\alpha = 0$. Since $A(\boldsymbol{X}, \boldsymbol{Y})$ and $B(\boldsymbol{X}, \boldsymbol{Y})$ play dual roles in the symmetric case, we can assume, w.l.o.g., that $B_\alpha = 0$ (the same assumption was made in [27]).

Because of $B_\alpha = 0$, the following claims hold:

- from the coefficient of $X_\alpha X_\beta$, $A_\alpha B_\beta + A_\beta B_\alpha - 1 = 0$. Thus, $A_\alpha B_\beta = 1$ (and in particularly, neither of them is equal to 0).
- from the coefficient of $X_\alpha Y_j$, $j \in [1 \mathinner{..} q]$, $A_\alpha B_{yj} + A_{yj} B_\alpha = 0$. Thus, $B_{yj} = 0$,
- from the coefficient of $X_\beta Y_j$, $j \in [1 \mathinner{..} q]$, $A_\beta B_{yj} + A_{yj} B_\beta = 0$. Thus, $A_{yj} = 0$,
- from the coefficient of $X_\delta Y_j$, $j \in [1 \mathinner{..} q]$, $C_{yj} = B_{yj} r_a + A_{yj} r_b$. Thus, $C_{yj} = 0$,

Since the coefficients A_{yj}, B_{yj}, and C_{yj} were the only coefficients that are new compared to the knowledge-soundness proof of [27], the rest of the current proof follows from the knowledge-soundness proof of [27].

Let us now compute a lower bound to the efficiency of a generic adversary (this was not done in [27], our bound clearly also holds in the case of Groth's SNARK). Assume that after some τ steps, the adversary has made a successful equality query $(=, i_1, i_2)$, i.e., $\mathsf{cell}_{i_1} = \mathsf{cell}_{i_2}$ for $i_1 \neq i_2$. Hence, she has found a collision $D_1(\boldsymbol{x}, \boldsymbol{y}) = D_2(\boldsymbol{x}, \boldsymbol{y})$ such that $D_1(\boldsymbol{X}, \boldsymbol{Y}) \neq D_2(\boldsymbol{X}, \boldsymbol{Y})$. Redefine $D_j(\boldsymbol{X}, \boldsymbol{Y}) := D_j(\boldsymbol{X}, \boldsymbol{Y}) \cdot X_\gamma X_\delta$ (if $\mathsf{type}_{i_1} \in \{1, 2\}$) and $D_j(\boldsymbol{X}, \boldsymbol{Y}) := D_j(\boldsymbol{X}, \boldsymbol{Y}) \cdot X_\gamma^2 X_\delta^2$ (if $\mathsf{type}_{i_1} = T$) for $j \in \{1, 2\}$, this guarantees that $D_j(\boldsymbol{X}, \boldsymbol{Y})$ is a polynomial. Thus,

$$D_1(\boldsymbol{x}, \boldsymbol{y}) - D_2(\boldsymbol{x}, \boldsymbol{y}) \equiv 0 \pmod{p} \ . \tag{3}$$

Note that

- If $\mathsf{type}_{i_1} = 1$, then $\deg D_j(\boldsymbol{X}, \boldsymbol{Y}) \leq 2n - 1 =: d_1$,
- If $\mathsf{type}_{i_1} = 2$, then $\deg D_j(\boldsymbol{X}, \boldsymbol{Y}) \leq 2n - 1 =: d_2$, and thus
- If $\mathsf{type}_{i_1} = T$, then $\deg D_j(\boldsymbol{X}, \boldsymbol{Y}) \leq 2 \cdot (2n - 1) = 4n - 2 =: d_T$.

Clearly, $\boldsymbol{x} = (\chi, \alpha, \beta, \gamma, \delta)$ is chosen uniformly random from $\mathbb{Z}_p^3 \times (\mathbb{Z}_p^*)^2$. Due to the assumption that the canonical values of Y_i are uniformly random in $\mathbb{Z}_p$, $\boldsymbol{y} = (\upsilon_1, \upsilon_2, \upsilon_3)$ is a uniformly random value in $\mathbb{Z}_p^3$. Hence, due to the Schwartz-Zippel lemma and since $D_1(\boldsymbol{X}, \boldsymbol{Y}) \neq D_2(\boldsymbol{X}, \boldsymbol{Y})$ as a polynomial, Eq. (3) holds with probability at most $\deg D_j(\boldsymbol{X}, \boldsymbol{Y})/(p-1) \leq d_{\mathsf{type}_{i_1}}/(p-1)$. Clearly, an adversary working in time τ can generate up to τ new group elements. Then the probability that there exists a collision between any two of those group elements is upper bounded by $\binom{\tau}{2} \cdot \deg D_j(\boldsymbol{X}, \boldsymbol{Y})/(p-1) \leq \binom{\tau}{2} \cdot d_{\mathsf{type}_{i_1}}/(p-1) \leq \tau^2/2 \cdot d_{\mathsf{type}_{i_1}}/(p-1)$. Thus, a successful adversary on average requires time at least τ, where $\tau^2 \geq 2(p-1)/d_{\mathsf{type}_{i_1}} \geq 2(p-1)/d_T = 2(p-1)/(4n-2)$, to produce a collision. Simplifying, we get $\tau = \Omega(\sqrt{p/n})$. □

7 Proof of Perfect Composable Subversion ZK

Before proving Thm. 4, we first prove the following helpful lemma that has a simple but tedious proof.

Lemma 3. *Let* $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$*, and let* crs *be any CRS such that* $\mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1$*. Then, with probability* 1*,* $\mathsf{crs} = \mathsf{K_{crs}}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc})$ *for a* tc *bijectively corresponding to* $[\chi, \alpha, \beta, \gamma, \delta]_1$*.*

Proof. In what follows, we will consider each line in the construction of CV in Fig. 1 separately, and write down the corollary from that line. We note that the CRS verification equations in Fig. 1 were written down as if the CRS were already correctly formed; e.g., there we have a check that $[\beta]_1 \bullet [1]_2 = [1]_1 \bullet [\beta]_2$ which may fail (and then obviously there exists no such β). However, before these equations are checked, it is of course not known that β in the left hand side (LHS) and in the right hand side (RHS) are equal. Therefore, in this proof only, in each

Algorithm 3: $\Sigma^{\iota}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$

```
1 $(\mathsf{crs}, \mathsf{z}_\Sigma) \leftarrow \Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$;
2 return $([\iota]_1, [\iota]_2)$;
```

Algorithm 4: $\mathsf{X}_\Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$

```
1 for $\lambda \in \{\chi, \alpha, \beta, \gamma, \delta\}$ do
2     $\lambda \leftarrow \mathsf{X}^{\lambda}_{\Sigma}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$;
3 $\mathsf{tc} \leftarrow (\chi, \alpha, \beta, \gamma, \delta)$;
4 return $\mathsf{tc}$;
```

Algorithm 5: $\mathsf{S}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$

```
1 $\sigma, \tau \leftarrow_r \mathbb{Z}_p$;
2 $(\mathfrak{a}, \mathfrak{b}) \leftarrow ([\sigma]_1, [\tau]_2)$;
3 $\mathfrak{c} \leftarrow [(\sigma\tau - \alpha\beta - \sum_{j=m_0+1}^{m}(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi)))/\delta]_2$;
4 return $\pi \leftarrow (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$;
```

Fig. 3. Algorithms used in extraction and simulation, where $\iota \in \{\chi, \alpha, \beta, \gamma, \delta\}$

step below, we use D_1 as the temporary name of the yet-unestablished LHS variable and D_2 as the temporary name of the yet-unestablished RHS variable.

We assume that ι in $[\iota]_1$ is already established for $\iota \in \{\chi, \alpha, \beta, \gamma, \delta\}$. (In particular, χ is established on Step 3 when $i = 1$.) We can do it since $[\iota]_1$ information-theoretically fixes ι.

1. For $\iota \in \{\gamma, \delta\}$: after that we know that $\iota \neq 0$, and hence there is a bijection between a valid $\mathsf{tc} \in \mathbb{Z}_p^3 \times (\mathbb{Z}_p^*)^2$ and the value hidden in $[\chi, \alpha, \beta, \gamma, \delta]_1$.
2. For $\iota \in \{\alpha, \beta, \gamma, \delta\}$: from $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [D_2]_2$ we get $[\iota]_T = [D_2]_T$. This implies $[\iota - D_2]_T = [0]_T$. Since $[1]_T$ is not the unity element, $[D_2]_2 = [\iota]_2$.
3. For $i = 1$ to $n - 1$: we can assume by induction that we have already established $\left[\chi^{i-1}\right]_1$. Since $[D_1]_T = \left[\chi^{i-1}\right]_1 \bullet [\chi]_2$, we get $[D_1]_T = \left[\chi^{i}\right]_T$.
4. For $i = 1$ to n: $\omega' = \omega^{i-1}$ and $\zeta = (\chi^n - 1)\omega^{i-1}/n$ in the algorithm in Sect. 8 are computed by the CRS verifier. The equation $[D_1]_1 \bullet ([\chi]_2 - [\omega']_2) = [\zeta]_T$ implies $[D_1]_1 = [\zeta/(\chi - \omega')]_1 = \left[(\chi^n - 1)\omega^{i-1}/(n(\chi - \omega^{i-1}))\right]_1 = [\ell_i(\chi)]_1$.
5. For $j = 0$ to m:
   (a) $[D_1]_1 = \sum_{i=1}^{n} U_{ij} [\ell_i(\chi)]_1$ implies $[D_1]_1 = [\sum_{i=1}^{n} U_{ij}\ell_i(\chi)]_1 = [u_j(\chi)]_1$.
   (b) $[D_1]_1 = \sum_{i=1}^{n} V_{ij} [\ell_i(\chi)]_1$ implies $[D_1]_1 = [\sum_{i=1}^{n} V_{ij}\ell_i(\chi)]_1 = [v_j(\chi)]_1$.
   (c) Clearly, $[w_k(\chi)]_1$ is computed correctly.
   (d) $[v_j(\chi)]_1 \bullet [1]_2 = [1]_1 \bullet [D_2]_2$ implies $[D_2]_2 = [v_j(\chi)]_2$.
6. For $j = 0$ to m_0: $[D_1]_1 \bullet [\gamma]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$ implies $[D_1]_1 = [(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\gamma]_1$.
7. For $j = m_0+1$ to m: $[D_1]_1 \bullet [\delta]_2 = [u_j(\chi)]_1 \bullet [\beta]_2 + [v_j(\chi)]_1 \bullet [\alpha]_2 + [w_j(\chi)]_1 \bullet [1]_2$ implies $[D_1]_1 = [(u_j(\chi)\beta + v_j(\chi)\alpha + w_j(\chi))/\delta]_1$.
8. $\left[\chi^{n-1}\right]_1 \bullet [1]_2 = [1]_1 \bullet [D_2]_2$ implies $[D_2]_2 = \left[\chi^{n-1}\right]_2$,
9. For $i = 0$ to $n - 2$: $[D_1]_1 \bullet [\delta]_2 = \left[\chi^{i+1}\right]_1 \bullet \left[\chi^{n-1}\right]_2 - \left[\chi^{i}\right]_1 \bullet [1]_2$ implies $[D_1]_1 = \left[\chi^{i}\ell(\chi)/\delta\right]_1$.
10. $[\alpha]_1 \bullet [\beta]_2 = [D_2]_T$ implies $[D_2]_T = [\alpha\beta]_T$.

By direct observation, it is clear that we have now established all elements of the CRS in Fig. 1. Hence, $\mathsf{crs} = \mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$. $\square$

Theorem 4. *The SNARK from Sect. 5.1 has perfect CRS trapdoor extractability under the BDH-KE assumption.*

Proof. Let Σ be a subverter, and let $r \leftarrow_r \mathsf{RND}(\Sigma)$. Let $\iota \in \{\chi, \alpha, \beta, \gamma, \delta\}$. Let $\Sigma^\iota(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$ be as in Alg. 3 in Fig. 3. Since $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs})$ accepts, crs must contain $([\iota]_1, [\iota]_2)$. Hence, by the BDH-KE assumption, there exists a non-uniform PPT extractor X_Σ^ι, such that if $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$ then $\iota \leftarrow \mathsf{X}_\Sigma^\iota(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$.

Finally, we construct the (non-uniform PPT) extractor $\mathsf{X}_\Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$ in Alg. 4 in Fig. 3. By the BDH-KE assumption, if $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$ then $\mathsf{tc} \leftarrow \mathsf{X}_\Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$ satisfies $\mathsf{tc} \in \mathrm{range}(\mathsf{K}_{\mathsf{tc}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}))$.

To prove CRS trapdoor extractability, assume that crs is returned by Σ and tc is returned by X_Σ. In addition, assume that $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$. By $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$ and Lem. 3, $\mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc}) = \mathsf{crs}$. The claim follows. $\square$

In Thm. 5, we also need the next simple lemma.

Lemma 4. *Let* $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, *and let* crs *be any CRS such that* $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$. *Consider any values of* $\mathfrak{a}$, $\mathfrak{b}$, *and* $(A_j)_{j=0}^{m_0}$. *Then there exists at most one value* $\mathfrak{c}$, *such that* $\mathsf{V}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x}, (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})) = 1$.

Proof. Assume that $\mathfrak{c}_0$ and $\mathfrak{c}_1$ are both accepted by the verifier. That means (see Fig. 2) that $\mathfrak{c}_k \bullet [\delta]_2 = [s]_T$ for $k \in \{0, 1\}$, where s is some k-independent value. By bilinearity, $(\mathfrak{c}_1 - \mathfrak{c}_0) \bullet [\delta]_2 = [0]_T$. Since $\delta \neq 0$ (this is guaranteed by CV accepting crs) and the pairing is non-degenerate, we have $\mathfrak{c}_0 = \mathfrak{c}_1$. $\square$

Theorem 5. *The SNARK from Sect. 5.1 is perfectly composable Sub-ZK under the BDH-KE assumption.*

Proof. We use the same simulator $\mathsf{S}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$ as defined in [27], see Alg. 5 in Fig. 3. Fix a subverter Σ, and let X_Σ be as defined in Thm. 4, see Alg. 4 in Fig. 3. Fix κ, $(\mathbf{R}, \mathsf{z}_{\mathbf{R}}) \leftarrow \mathcal{R}(1^\kappa)$, $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, and an adversary A. As in Def. 7, assume $r \leftarrow_r \mathsf{RND}(\Sigma)$, $(\mathsf{crs}, \mathsf{z}_\Sigma) \leftarrow \Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$ such that $\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}) = 1$, $\mathsf{tc} \leftarrow \mathsf{X}_\Sigma(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$, $\mathsf{ts} \leftarrow \mathsf{K}_{\mathsf{ts}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc})$. Assume that $\pi_0 \leftarrow \mathsf{P}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w})$ ($b = 0$) and $\pi_1 \leftarrow \mathsf{S}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$ ($b = 1$). It is sufficient to show that π_0 and π_1 have the same distribution.

Case $b = 0$**.** The honest prover creates $\mathfrak{a} \leftarrow \ldots + r_a [\delta]_1$ and $\mathfrak{b} \leftarrow \ldots + r_b [\delta]_2$ for uniformly random r_a and r_b. Since $\delta \neq 0$ (this is guaranteed by CV accepting crs) and the pairing is non-degenerate, $\mathfrak{a}$ and $\mathfrak{b}$ are uniformly random. We know by the perfect CRS trapdoor extractability (Thm. 4) that $\mathsf{K}_{\mathsf{crs}}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{tc}) = \mathsf{crs}$. Thus, $(\mathsf{crs}, \mathsf{ts})$ are created as in the first step of the definition of perfect subversion-completeness (Def. 2). Since the new SNARK satisfies perfect subversion-completeness (Thm. 2), we obtain $\mathsf{V}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x}, \pi_0) = 1$. By Lem. 4, the verification equation, crs, $\mathfrak{a}$, and $\mathfrak{b}$ uniquely determine the acceptable $\mathfrak{c}$.

Case $b = 1$**.** Here, by the definition of the simulator, $\mathfrak{a}$ and $\mathfrak{b}$ are uniformly random. Moreover, π_1 is explicitly created so that $\mathsf{V}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{V}, \mathsf{x}, \pi_1) = 1$. Since CV accepts then by Lem. 4, the verification equation, crs, $\mathfrak{a}$, and $\mathfrak{b}$ uniquely determine the acceptable $\mathfrak{c}$.

Since CV accepts crs, in both cases $\mathfrak{a}$ and $\mathfrak{b}$ are uniformly random and $\mathfrak{c}$ is uniquely determined by them, crs, and the verification equation, the real and the simulated arguments have identical distributions. □

The following result follows directly from Thm. 1 and Thm. 5.

Theorem 6. *The SNARK from Sect. 5.1 is perfectly unbounded Sub-ZK under the BDH-KE assumption.*

8 Efficiency

CRS Length. The CRS contains gk (that includes $([1]_1, [1]_2)$) and a number of additional elements. Not counting gk, the number of CRS elements in different groups is given by the following table. Hence, the total size of the CRS is $4m + 3n + 13$ group elements.

	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	Total
crs_P	$3m + n - m_0 + 4$	$m + 3$	0	$4m + n - m_0 + 7$
crs_V	$m_0 + 1$	2	1	$m_0 + 4$
crs_CV	$2n$	3	0	$2n + 3$
Total	$3m + 3n + 5$	$m + 7$	1	$4m + 3n + 13$

One element (namely, $[\delta]_2$) belongs both to crs_P and crs_V and thus the numbers in the "total" row are not equal to the sum of the numbers in previous rows.

In Groth's zk-SNARK [27] the CRS consists of $m + 2n$ elements of $\mathbb{G}_1$ and n elements of $\mathbb{G}_2$. On top of it, we added $2n + 3$ group elements to make the CRS verification possible and also some elements to speed up the prover's computation and the verifier's computation; the latter elements can alternatively be computed from the rest of the CRS.

CRS Generation: Computational Complexity. Assume that gk has already been computed. One can compute crs by first computing all CRS elements within brackets, and then compute their bracketed versions. One can evaluate $u_j(\chi)$, $v_j(\chi)$, and $w_j(\chi)$ for each $j \in [0 \mathinner{..} m]$ in time $\Theta(n)$ by using precomputed values $\ell_i(\chi)$ for $i \in [1 \mathinner{..} n]$ and the fact that the matrices U, V, W contain $\Theta(n)$ non-zero elements. The rest of the CRS can be computed efficiently by using straightforward algorithms.

By using Alg. 1, the whole CRS generation algorithm is dominated by $3m + 3n + 5$ exponentiations in $\mathbb{G}_1$, $m + 7$ exponentiations in $\mathbb{G}_2$, and 1 exponentiation in $\mathbb{G}_T$ (one per CRS element) and $\Theta(n)$ multiplications/divisions in $\mathbb{Z}_p$.

CV*'s Computational Complexity.* We assume that it is difficult to subvert gk; this makes sense assuming that the SNARK uses well-known bilinear groups (say, the Barreto-Naehrig curves). Consider the CRS verification algorithm in Fig. 1. It is clear that all other steps but Step 4 are efficient (computable in $\Theta(n)$ cryptographic operations); this follows from the fact that U, V, and W are sparse. Computation in those steps is dominated by $6m + 5n + 12$ pairings. On top of it, one has to execute $s(U) + s(V) + s(W)$ exponentiations in $\mathbb{G}_1$, where $s(M)$ is the number of "large" (i.e., large enough so that exponentiating with them would be expensive) entries in the matrix M. Often, $s(M)$ are very small.

By using Alg. 2, one can check that $[\ell_i(\chi)]_1$ has been correctly computed for all $i \in [1 \mathinner{..} n]$ in $n+1$ pairings, $n-1$ exponentiations in $\mathbb{G}_2$ and n exponentiations in $\mathbb{G}_T$. Hence, the whole CRS verification algorithm is dominated by $6m+6n+13$ pairings, $s(U) + s(V) + s(W)$ exponentiations in $\mathbb{G}_1$, $n - 1$ exponentiations in $\mathbb{G}_2$, and n exponentiations in $\mathbb{G}_T$.

In addition, one can speed up CV by using batching [4]. Namely, clearly if $\sum_{i=1}^{s} t_i([a_i]_1 \bullet [b_i]_2) = [c]_T$ for uniformly random t_i, then w.h.p., $[a_i]_1 \bullet [b_i]_2 = [c]_T$ for each individual $i \in [1 \mathinner{..} s]$. The speed up follows from the use of bilinear properties and from the fact that exponentiation is faster than pairing. Moreover, one can further slightly optimize this by assuming $t_s = 1$ [34,19].

Full batched version of CV is described in App. B. As we will show there, a batched CV will be dominated by $5(m+n)+s(U)+s(V)$ (mostly, short-exponent) exponentiations in $\mathbb{G}_1$ and $m + s(W)$ (mostly, short-exponent) exponentiations in $\mathbb{G}_2$. Since an exponentiation with short exponent is significantly less costly than a pairing, this will decrease the execution time of CV significantly. (See App. B.2 for concrete numbers.)

We note that after taking batching into account, CV will become a probabilistic algorithm, and will accept incorrect CRSs with negligible probability. This means that one has to modify some of the previous security results. For example, Thms. 4 and 5 will be modified as follows.

Theorem 7. *After batching* CV*, the SNARK from Sect. 5.1 has statistical CRS trapdoor extractability under the BDH-KE assumption.*

Theorem 8. *After batching* CV*, the SNARK from Sect. 5.1 is statistically composable Sub-ZK under the BDH-KE assumption.*

Prover's Computational Complexity. As in [27], the prover's computational complexity is dominated by the need to compute $h(X)$ (3 interpolations, 1 polynomial multiplication, and 1 polynomial division; in total $\Theta(n \log n)$ non-cryptographic operations in $\mathbb{Z}_p$), followed by $(n-1)+(s(A)+1)+1+(s(A)+1)+s(A_1, \dots, A_{m_0}) \le n + 3s(A) + 2$ exponentiations in $\mathbb{G}_1$ and $s(A) + 1$ exponentiations in $\mathbb{G}_2$, where $s(A)$ is the number of large elements in A (i.e., large enough so that exponentiating with them would be expensive). This means that the prover's computation is dominated by $\Theta(n \log n)$ non-cryptographic operations and $\Theta(n)$ cryptographic operations.

Verifier's Computational Complexity. The verifier has to execute a single pairing equation that is dominated by 3 pairings and m_0 exponentiations in $\mathbb{G}_1$. The exponentiations can be done offline since they do not depend on the argument π but only on the common input $(A_1, \ldots, A_{m_0})$. Hence, the verifier's computation is dominated by $\Theta(m_0)$ cryptographic operations but her online computation is only dominated by 3 pairings.

Argument Length. The argument consists of 2 elements from $\mathbb{G}_1$ and 1 element from $\mathbb{G}_2$.

9 The Case of Unbounded Subverter

Since we consider statistical Sub-ZK, it is natural to ask what will happen if also the subverter is computationally unbounded. It comes out that in this case, several definitions and proofs would actually simplify. For this reason, we decided to first present the case of efficient subverter.

Assume now that the subverter Σ is computationally unbounded. Then we need a computationally unbounded extractor X_Σ (otherwise, it will not be able to even execute Σ). For the sake of completeness, we will give the corresponding version of Def. 5 with changes being *emphasized*.

Definition 10 (Statistically Unbounded USub-ZK). *Ψ is* statistically unbounded Sub-ZK with unbounded subverter (USub-ZK) for $\mathcal{R}$, *if for any computationally unbounded subverter Σ there exists a computationally unbounded X_Σ, such that for all κ, all $(\mathbf{R}, \mathsf{z}_\mathbf{R}) \in \mathrm{range}(\mathcal{R}(1^\kappa))$, and all computationally unbounded A, $\varepsilon_0^{unb} \approx_\kappa \varepsilon_1^{unb}$, where*

$$\varepsilon_b^{unb} = \Pr\left[\begin{array}{l} r \leftarrow_r \mathsf{RND}(\Sigma), (\mathsf{crs}, \mathsf{z}_\Sigma \parallel \mathsf{tc}) \leftarrow (\Sigma \parallel \mathsf{X}_\Sigma)(\mathbf{R}, \mathsf{z}_\mathbf{R}; r), \\ \mathsf{ts} \leftarrow \mathsf{K}_\mathsf{ts}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{tc}) : \mathsf{CV}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}) = 1 \wedge \\ \mathsf{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{z}_\Sigma) = 1 \end{array}\right] .$$

Here, the oracle $\mathsf{O}_0(\mathsf{x}, \mathsf{w})$ returns $\perp$ (reject) if $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$, and otherwise it returns $\mathsf{P}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}_\mathsf{P}, \mathsf{x}, \mathsf{w})$. Similarly, $\mathsf{O}_1(\mathsf{x}, \mathsf{w})$ returns $\perp$ (reject) if $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$, and otherwise it returns $\mathsf{S}(\mathbf{R}, \mathsf{z}_\mathbf{R}, \mathsf{crs}, \mathsf{ts}, \mathsf{x})$. Ψ is perfectly unbounded USub-ZK for $\mathcal{R}$ *if one requires that $\varepsilon_0^{unb} = \varepsilon_1^{unb}$.*

An unbounded Σ can obviously break the BDH-KE assumption by creating elements of $\mathbb{G}_1$ and $\mathbb{G}_2$ as he wants; however, because she is unbounded, we can of course still argue that Σ will know their discrete logarithms, or more formally, that X_Σ will be able to extract them. Since the extraction is now unconditional (i.e., it does not depend on any assumptions), it means that X_Σ can extract the discrete logarithm of any element of $\mathbb{G}_z$.

In particular, there is no need anymore to include $[\gamma]_1$ to the CRS (this makes the CRS shorter by 1 element), or handle it in CV (that is, one could remove the check that $[\gamma]_1 \bullet [1]_2 = [1]_1 \bullet [\gamma]_2$) or in security proofs (e.g., in Lem. 3, one would not have to use the equation $[\iota]_1 \bullet [1]_2 = [1]_1 \bullet [\iota]_2$ to establish $[\gamma]_1$, or define $\Sigma^{\gamma}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}; r)$ inside Thm. 4).

Moreover, if the CRS has the (easily satisfied) bijectivity property required in Sect. 7 (see Lem. 3), it means that the requirement that X_{Σ} returns tc (instead of just returning ts) is not restrictive anymore.

Acknowledgment. We would like to thank Janno Siim for his help in the optimization of the batched CV algorithm. This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 653497 (project PANORAMIX), and by institutional research funding IUT2-1 of the Estonian Ministry of Education and Research.

References

1. Barbulescu, R., Duquesne, S.: Updating Key Size Estimations for Pairings. Technical Report 2017/334, IACR (2017) `http://eprint.iacr.org/2017/334`, revision from April 26, 2017.
2. Barreto, P.S.L.M., Naehrig, M.: Pairing-Friendly Elliptic Curves of Prime Order. In: SAC 2005. LNCS, vol. 3897, pp. 319–331
3. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion. In: ASIACRYPT 2016 (2). LNCS, vol. 10032, pp. 777–804
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., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized Anonymous Payments from Bitcoin. In: IEEE SP 2014, pp. 459–474
6. 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
7. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs. In: IEEE SP 2015, pp. 287–304
8. 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
9. 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
10. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data. In: STOC 2013, pp. 241–250
11. Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the Existence of Extractable One-Way Functions. In: STOC 2014, pp. 505–514
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., Goh, E.J.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456
14. Bowe, S., Gabizon, A., Green, M.: A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Available from `https://z.cash/downloads/mpc_whitepaper.pdf`, last checked in March 2017 (2016)

15. 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
16. Damgård, I.: Towards Practical Public Key Systems Secure against Chosen Ciphertext Attacks. In: CRYPTO 1991. LNCS, vol. 576, pp. 445–456
17. 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
18. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.L.: An Algebraic Framework for Diffie-Hellman Assumptions. In: CRYPTO (2) 2013. LNCS, vol. 8043, pp. 129–147
19. Fauzi, P., Lipmaa, H., Zając, M.: A Shuffle Argument Secure in the Generic Model. In: ASIACRYPT 2016 (2). LNCS, vol. 10032, pp. 841–872
20. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for Cryptographers. Discrete Applied Mathematics **156**(16) (2008) pp. 3113–3121
21. Gennaro, R., Gentry, C., Parno, B.: Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: CRYPTO 2010. LNCS, vol. 6223, pp. 465–482
22. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and NIZKs without PCPs. In: EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645
23. Gentry, C., Wichs, D.: Separating Succinct Non-Interactive Arguments from All Falsifiable Assumptions. In: STOC 2011, pp. 99–108
24. Goldreich, O.: A Uniform-Complexity Treatment of Encryption and Zero-Knowledge. J. Cryptology **6**(1) (1993) pp. 21–53
25. Groth, J.: Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In: ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459
26. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340
27. Groth, J.: On the Size of Pairing-based Non-interactive Arguments. In: EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326
28. Groth, J., Sahai, A.: Efficient Non-interactive Proof Systems for Bilinear Groups. In: EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432
29. Hess, F., Smart, N.P., Vercauteren, F.: The Eta Pairing Revisited. IEEE Trans. on Inf. Theory **52**(10) (2006) pp. 4595–4602
30. Jutla, C.S., Roy, A.: Shorter Quasi-Adaptive NIZK Proofs for Linear Subspaces. In: ASIACRYPT 2013 (1). LNCS, vol. 8269, pp. 1–20
31. Kim, T., Barbulescu, R.: Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case. In: CRYPTO 2016. LNCS, vol. 9814, pp. 543–571
32. Lipmaa, H.: Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In: TCC 2012. LNCS, vol. 7194, pp. 169–189
33. 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
34. Lipmaa, H.: Prover-Efficient Commit-And-Prove Zero-Knowledge SNARKs. In: AFRICACRYPT 2016. LNCS, vol. 9646, pp. 185–206
35. Maurer, U.M.: Abstract Models of Computation in Cryptography. In: Cryptography and Coding 2005, pp. 1–12
36. Nechaev, V.I.: Complexity of a Determinate Algorithm for the Discrete Logarithm. Mathematical Notes **55**(2) (1994) pp. 165–172 Translated from *Matematicheskie Zapiski*, 55(2):91–101, 1994.

37. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: IEEE SP 2013, pp. 238–252
38. 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
39. Schwartz, J.T.: Fast Probabilistic Algorithms for Verification of Polynomial Identities. Journal of the ACM **27**(4) (1980) pp. 701–717
40. Shoup, V.: Lower Bounds for Discrete Logarithms and Related Problems. In: EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266
41. Zippel, R.: Probabilistic Algorithms for Sparse Polynomials. In: EUROSM 1979. LNCS, vol. 72, pp. 216–226

Supplementary Material

A Comparison to MPC Approach

In [7], Ben-Sasson *et al.* proposed a very different (at least, when compared to the paper of Bellare *et al.* [3]) approach to achieve security in the case of active CRS subversion. Their approach involves joint creation of the CRS by multiple parties, by using a specialized and heavily optimized MPC protocol. (See [14] for additional optimizations.) During their protocol, each party first generates a non-zero multiplicative share of each element of the trapdoor tc, broadcasts it, and gives a zero-knowledge proof of knowledge of the share. After that, the parties will essentially execute a joint MPC CRS creation/verification protocol (in particular, this approach also needs the CRS to be verifiable and thus the CRS will have additional elements compared to a non subversion-resistant SNARK). Intuitively, at each step of the MPC protocol (the actual protocol is more complicated), the parties jointly generate a single new element of the CRS by combining their multiplicative shares and giving zero-knowledge proofs that the combining part was done correctly.

At the intuitive level, the protocol of Ben-Sasson *et al.* [7] achieves something very similar to subversion-resistance (the precise term in [7] is *auditable transcript*): the CRS of their SNARK is verifiable and it also achieves, due to the use of proofs of knowledge, the CRS trapdoor extractability property. Hence, a scaled-down single-party version of their SNARK can be seen as getting *some version* of Sub-ZK. However, [7] does not provide security definitions corresponding to Sub-ZK nor consider that case.

It also only considers "duplex-pairing groups", meaning that CRS elements have to be doubled, i.e., to be present in both $\mathbb{G}_1$ or $\mathbb{G}_2$; by following our approach, this is only needed for exponentiated CRS trapdoor elements (even this is only needed in the case of efficient subverter). Their initial proofs of knowledge can be seen as an alternative approach to the use of knowledge assumptions in [3] and in the current paper; however, without having interactive zero-knowledge

Algorithm 6: Checking that $[(\ell_i(\chi))_{i=1}^n]_1$ is correctly computed: batched version

```
   Input: $([\chi^{n-1}, (\ell_i(\chi))_{i=1}^n]_1, [1, \chi]_2, [1]_T)$
1  $\omega_0 \leftarrow 1/\omega$;
2  $[a]_1 \leftarrow [0]_1$; $[b]_1 \leftarrow [0]_1$; $c \leftarrow 0$;
3  for $i = 1$ to $n$ do
4      $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$; $\omega_i \leftarrow \omega\omega_{i-1}$;
5      $[a]_1 \leftarrow [a]_1 + t_i [\ell_i(\chi)]_1$; $[b]_1 \leftarrow [b]_1 + t_i\omega_i [\ell_i(\chi)]_1$; $c \leftarrow c + t_i\omega_i$;
6  Check that $[a]_1 \bullet [\chi]_2 - [b]_1 \bullet [1]_2 = c/n \cdot ([\chi^{n-1}]_1 \bullet [\chi]_2 - [1]_T)$;
```

arguments or random-oracle model, the only possibility to apply proofs of knowledge seems to be to use knowledge assumptions.

This curious similarity between [7] and the current paper seems to ascertain that the formalism given in the current paper is correct. We will leave it up to a further work to study how to combine the MPC approach of [7] and the subversion-resistance approach of [3] and the current paper. For example, knowledge assumptions can be used to replace proofs of knowledge in [7] (and hence speed up their protocol). Most importantly, the security definitions of the current paper and the new Sub-ZK SNARK intuitively capture (although this still has to be studied formally) what one can achieve in the practically relevant scenario where one wants to achieve subversion-resistance but does not have a reason to trust *any* of the CRS creators.

We will provide a more specific comparison (based on actual implementations) between the speed of the CV in the new SNARK and the speed of transcript verification in App. B.2.

B Batching and Implementation

B.1 Batching

We used batching techniques from [4] to make the CV algorithm more efficient, see Fig. 4 for implementation details. Especially, we use a corollary of the Schwartz-Zippel lemma stating that if $\sum_{i=1}^{s-1} t_i X_i + X_s = 0$, where $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i < s$, then $X_i = 0$ for each i, with probability $1 - 1/2^\kappa$. We also employ the following lemma to show that randomness generated in batching can be used multiple times. The lemma comes originally from [34,19], but it is closely related to the small exponents test from [4].

Lemma 5. *Assume* $1 < t < q$. *Assume* $\boldsymbol{t}$ *is a vector chosen uniformly random from* $[1 \mathinner{..} t]^{k-1} \times \{1\}$, $\boldsymbol{\chi}$ *is a vector of integers in* $\mathbb{Z}_q$, *and* f_i *are some polynomials of degree* $\mathsf{poly}(\kappa)$. *If* $\sum_{i=1}^k f_i(\boldsymbol{\chi}) t_i \cdot ([1]_1 \bullet [1]_2) = [0]_T$, *then with probability* $\geq 1 - \frac{1}{t}$, $f_i(\boldsymbol{\chi})([1]_1 \bullet [1]_2) = [0]_T$ *for each* $i \in [1 \mathinner{..} k]$.

In the batched CV algorithm we execute Alg. 6 instead of Alg. 2. This algorithm needs 3 pairings, $2n$ exponentiations in $\mathbb{G}_1$, and 1 exponentiation in $\mathbb{G}_T$),

```
$\mathsf{CV}(\mathbf{R}, \mathsf{z}_{\mathbf{R}}, \mathsf{crs})$:  // batched CV
  1. For $\iota \in \{\gamma, \delta\}$: check that $[\iota]_1 \neq [0]_1$
     /* For $\iota \in \{\alpha, \beta, \gamma, \delta\}$:                                      */
  2. (a) Generate $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i = 1 \ldots 3$, then set $t_4 \leftarrow 1$.
     (b) Check that $[t_1\alpha + t_2\beta + t_3\gamma + \delta]_1 \bullet [1]_2 = [1]_1 \bullet [t_1\alpha + t_2\beta + t_3\gamma + \delta]_2$,
     /* For $i = 1$ to $n - 1$:                                                                  */
  3. (a) Generate $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i = 1$ to $n - 2$, then set $t_{n-1} \leftarrow 1$.
     (b) Check that $(\sum_{i=1}^{n-1} t_i \left[\chi^i\right]_1) \bullet [1]_2 = (\sum_{i=1}^{n-1} t_i \left[\chi^{i-1}\right]_1) \bullet [\chi]_2$
  4. Check that $\left[(\ell_j(\chi))_{j=1}^n\right]_1$ is correctly computed by using Alg. 6,
  5. For $i = 0$ to $m$:
     (a) Set $[u_i(\chi)]_1 \leftarrow \sum_{j=1}^n U_{ij} [\ell_j(\chi)]_1$,
     (b) Check that $[v_i(\chi)]_1 = \sum_{j=1}^n V_{ij} [\ell_j(\chi)]_1$,
     (c) Set $[w_i(\chi)]_1 \leftarrow \sum_{j=1}^n W_{ij} [\ell_j(\chi)]_1$,
     /* For $i = 0$ to $m$:                                                                      */
  6. (a) Generate $t_k \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $k = 0$ to $m - 1$, then set $t_m \leftarrow 1$.
     (b) Check that $(\sum_{k=0}^m t_k [v_k(\chi)]_1) \bullet [1]_2 = [1]_1 \bullet (\sum_{k=0}^m t_k [v_k(\chi)]_2)$,
     /* For $i = 0$ to $m_0$:                                                                    */
  7. (a) Generate $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i = 0$ to $m_0 - 1$, then set $t_{m_0} \leftarrow 1$.
     (b) Check that $(\sum_{i=0}^{m_0} t_i [(u_i(\chi)\beta + v_i(\chi)\alpha + w_i(\chi))/\gamma]_1) \bullet [\gamma]_2 =$
         $(\sum_{i=0}^{m_0} t_i [u_i(\chi)]_1) \bullet [\beta]_2 + (\sum_{i=0}^{m_0} t_i [v_i(\chi)]_1) \bullet [\alpha]_2 + (\sum_{i=0}^{m_0} t_i [w_i(\chi)]_1) \bullet [1]_2$,
     /* For $i = m_0 + 1$ to $m$:                                                                */
  8. (a) Generate $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i = m_0 + 1$ to $m - 1$, then set $t_m \leftarrow 1$.
     (b) Check that $(\sum_{i=m_0+1}^m t_i [(u_i(\chi)\beta + v_i(\chi)\alpha + w_i(\chi))/\delta]_1) \bullet [\delta]_2 =$
         $(\sum_{i=m_0+1}^m t_i [u_i(\chi)]_1) \bullet [\beta]_2 + (\sum_{i=m_0+1}^m t_i [v_i(\chi)]_1) \bullet [\alpha]_2 +$
         $(\sum_{i=m_0+1}^m t_i [w_i(\chi)]_1) \bullet [1]_2$.
  9. Check that $\left[\chi^{n-1}\right]_1 \bullet [1]_2 = [1]_1 \bullet \left[\chi^{n-1}\right]_2$.
     /* For $i = 0$ to $n - 2$:                                                                  */
 10. (a) Generate $t_i \leftarrow_r \{1, \ldots, 2^\kappa\}$ for $i = 0$ to $n - 3$, then set $t_{n-2} \leftarrow 1$.
     (b) Check that $(\sum_{i=0}^{n-2} t_i \left[\chi^i \ell(\chi)/\delta\right]_1) \bullet [\delta]_2 = (\sum_{i=0}^{n-2} t_i \left[\chi^{i+1}\right]_1) \bullet \left[\chi^{n-1}\right]_2 -$
         $(\sum_{i=0}^{n-2} t_i \left[\chi^i\right]_1) \bullet [1]_2$,
     (c) Check that $[\alpha]_1 \bullet [\beta]_2 = [\alpha\beta]_T$.
```

Fig. 4. Batched CV

and on top of it, 20 pairings, $5m + 5n + s(U) + s(V) - 12$ (mostly, short-exponent) exponentiations in $\mathbb{G}_1$, and $m + s(W) + 2$ (mostly, short-exponent) exponentiations in $\mathbb{G}_2$. This makes, in total, 23 pairings, $5m + 7n + s(U) + s(V) - 12$ (mostly, short-exponent) exponentiations in $\mathbb{G}_1$, $m + s(W) + 2$ (mostly, short-exponent) exponentiations in $\mathbb{G}_2$, and 1 exponentiation in $\mathbb{G}_T$.

B.2 Implementation

We compare the efficiency of the new subversion zk-SNARK with Groth's non-subversion zk-SNARK from both the theoretical (as reported in [27]) and the implementation (as implemented in the `libsnark` [9] library) point of view. Similarly to the pre-existing implementation of Groth's zk-SNARK in `libsnark`, we implemented the new SNARK in the C++ language by using low-level sub-

Table 1. Performance of the implementations of Groth's non-subversion zk-SNARK and the new Sub-ZK SNARK in `libsnark` for different values of n and m_0.

Protocol	n, m_0	K (s)	CV (s)	P (s)	Online V (s)
Groth's zk-SNARK	$n = 7\,500$, $m_0 = 100$	1.79	—	1.74	0.017
sub zk-SNARK	$n = 7\,500$, $m_0 = 100$	2.29	211.4	1.74	0.017
sub zk-SNARK batched	$n = 7\,500$, $m_0 = 100$	2.29	2.13	1.74	0.017
Groth's zk-SNARK	$n = 15\,000$, $m_0 = 100$	3.09	—	3.04	0.017
sub zk-SNARK	$n = 15\,000$, $m_0 = 100$	3.99	417.5	3.04	0.017
sub zk-SNARK batched	$n = 15\,000$, $m_0 = 100$	3.99	3.41	3.04	0.017
Groth's zk-SNARK	$n = 30\,000$, $m_0 = 100$	5.40	—	5.57	0.017
sub zk-SNARK	$n = 30\,000$, $m_0 = 100$	7.28	837.1	5.57	0.017
sub zk-SNARK batched	$n = 30\,000$, $m_0 = 100$	7.28	5.34	5.57	0.017
Groth'16 SNARK	$n = 30\,000$, $m_0 = 1\,000$	5.43	—	5.57	0.070
sub-SNARK	$n = 30\,000$, $m_0 = 1\,000$	7.30	845.7	5.57	0.070
sub-SNARK batched	$n = 30\,000$, $m_0 = 1\,000$	7.30	5.43	5.57	0.070
Groth's zk-SNARK	$n = 60\,000$, $m_0 = 1\,000$	9.83	—	10.5	0.070
sub zk-SNARK batched	$n = 60\,000$, $m_0 = 1\,000$	13.1	9.57	10.5	0.070
Groth'16 SNARK	$n = 120\,000$, $m_0 = 1\,000$	17.3	—	19.5	0.070
sub-SNARK batched	$n = 120\,000$, $m_0 = 1\,000$	22.9	17.2	19.5	0.070
Groth's zk-SNARK	$n = 250\,000$, $m_0 = 1\,000$	32.5	—	38.0	0.070
sub zk-SNARK batched	$n = 250\,000$, $m_0 = 1\,000$	43.0	32.3	38.0	0.070
Groth'16 SNARK	$n = 500\,000$, $m_0 = 1\,000$	61.1	—	74.0	0.070
sub-SNARK batched	$n = 500\,000$, $m_0 = 1\,000$	89.9	62.8	74.0	0.070

routines of `libsnark`. All the following results were measured in a 64-bit `Linux Ubuntu 16.10` virtual machine inside a 64-bit Windows 8.1 Enterprise with 4 GB RAM and a single core allocated for the virtual machine. The virtual machine was installed on a standard laptop (`HP EliteBook 840`), with the Intel `core i5-5200u 2.2 GHz` CPU and 8 GB RAM. We built the `libsnark` library by using the option `CURVE=BN128` that provides an instantiation based on a Barreto-Naehrig curve at 128 (or, 100 bits, according to [1]) bits of security.

Tbl. 1 compares the implementation of Groth's zk-SNARK in `libsnark` with our implementation of the new subversion-resistant zk-SNARK for several choices of m_0 and n. We report several measures including the running time of K, CV, P, and V. All times are expressed in seconds. Fig. 5 shows how much time the (batched) CV needs to verify the CRS.

In the case of the new Sub-ZK SNARK, we evaluated the performance of the CV algorithm by using both the non-batched (Fig. 1) and batched (Fig. 4) versions. In the execution of the batched CV, we first sample a vector $\boldsymbol{t}$ of random numbers from $[1\,..\,2^{80}]$. This vector has length m, since no verification equation needs more than m random values. As stated in Sect. B.1, we reuse randomness, i.e., in every verification equation we use random values from the same $\boldsymbol{t}$. We computed the running times as averages over 10 iterations. We emphasize that while the non-batched CV is very slow (this is why we are not giving its timings

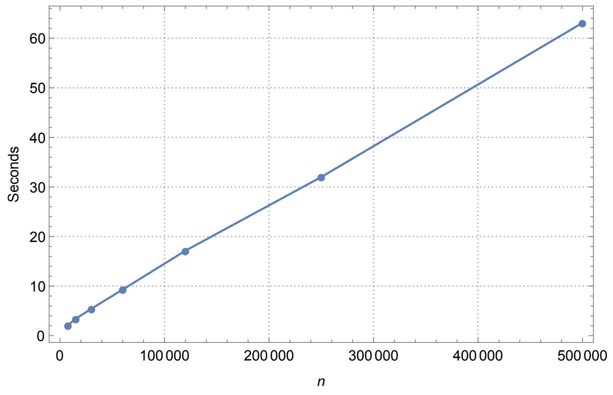

Fig. 5. Batched CV efficiency

for $n > 30\,000$), the batched CV is only about actually faster than P for $n = 30\,000$ and larger.

As it has been already mentioned, the randomness vector $\boldsymbol{t}$ takes inputs from $[1 \mathinner{..} 2^{80}]$, this assures that the batching causes security gap that is not bigger than 2^{-80}. This is a conservative approach. If instead the coordinates of $\boldsymbol{t}$ are chosen from $[1 \mathinner{..} 2^{40}]$ then the CV algorithm will take 1.30, 2.41, 3.75 and 3.82 seconds for (n, m_0) being equal to $(7500, 100)$, $(15000, 100)$, $(30000, 100)$, $(30000, 1000)$, respectively. These times are about 30% smaller than in the case $t \leftarrow_r [1 \mathinner{..} 2^{80}]$, and faster than the computational cost of the prover.

Tbl. 2 summarizes the number of CRS elements in Groth's zk-SNARK and our new subversion zk-SNARK based on values given in [27] and `libsnark` implementation. (Note the slight difference between CRS in the original Groth paper and the implemented version).

Table 2. A comparison on number of crs elements in Groth's non-subversion zk-SNARK and new subversion zk-SNARK.

	crs_P			crs_V			crs_CV			
SNARKs	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\sum$
Groth'16	$m + 2n - m_0 + 2$	$n + 1$	0	$m_0 + 1$	2	1	0	0	0	m+3n+6
$\sum$	$m + 3n - m_0 + 3$			$m_0 + 3$			0			
Groth'16 imp.	$3m + n - m_0 + 4$	$m + 1$	0	m_0	2	1	0	0	0	$4m + n + 8$
$\sum$	$4m + n - m_0 + 5$			$m_0 + 3$			0			
This work	$3m + n - m_0 + 4$	$m + 3$	0	$m_0 + 1$	2	1	$2n$	3	0	$4m + 3n + 14$
$\sum$	$4m + n - m_0 + 7$			$m_0 + 4$			$2n + 3$			
This work imp.	$3m + n - m_0 + 4$	$m + 3$	0	$m_0 + 1$	2	1	$2n + 2$	3	0	
$\sum$	$4m + n - m_0 + 7$			$m_0 + 4$			$2n + 5$			$4m + 3n + 16$

One could observe that the algorithm presented in Fig. 4 can be optimized even more. More precisely, the verification equations in steps 2, 6, and 9 can

Table 3. Comparison between CV algorithm and transcript verification from [7].

n	7 500	15 000	30 000	60 000	120 000	250 000	500 000
This paper, CV	2.13 s	3.41 s	5.34 s	9.57 s	17.20 s	32.3 s	62.8 s
[7] transcript ver.	7.73 s	15.46 s	30.9 s	61.8 s	123.6 s	247.2 s	494.4 s

be merged to a equation. Such an operation adds 1 exponentiation in $\mathbb{G}_1$ and 1 exponentiation in $\mathbb{G}_2$ and needs 4 pairings less. Similarly, the right sides of steps 7 and 8 have the same structure, which makes possible to decrease total number of pairings by 3 more. The verification equations $[\alpha]_1 \bullet [\beta]_2 = [\alpha\beta]_T$ in step 10.c of the batched CV can be done inside step 8.b of the algorithm, which will drop one more pairing. Thus, we need 8 pairings less and 1 extra exponentiation in $\mathbb{G}_1$ and 1 extra exponentiation in $\mathbb{G}_2$. The batched CV could be executed in total with 15 pairings, $5m + 7n + s(U) + s(V) - 11$ (mostly, short-exponent) exponentiations in $\mathbb{G}_1$, $m + s(W) + 3$ (mostly, short-exponent) exponentiations in $\mathbb{G}_2$, and 1 exponentiation in $\mathbb{G}_T$.

Comparison to the MPC approach. As it was mentioned before, Ben-Sasson *et al.* [7] provided a method to generate the CRS in an MPC manner. Their result can be also utilized for a single party performing all computations. In such a case *well-formedness* of CRS is guaranteed by the transcript verification. Unfortunately, in this case their protocol. Below we provide a short comparison between our CV protocol and transcript verification from [7].

Ben-Sasson *et al.* stated that for a circuit $\mathcal{C}$ of size $\mathsf{size}(\mathcal{C})$ transcript verification time takes $1.03 \cdot \mathsf{size}(\mathcal{C})$ milliseconds. (We skip here multiplier N referring to the number of parties, since in our case $N = 1$.) We did not measured the size of the circuit exactly, but we used a lower bound on it, i.e., we assumed that the size of the circuit is at least equal to n.

Since the computational complexity in [7] was expressed in seconds, we have to compare the computers which perform computations. Roughly speaking, while Ben-Sasson *et al.* used a device with an i7, 3.4 GHz CPU and 16 GB of RAM, we used an i5, 2.2 GHz CPU with 4 GB RAM. Nevertheless, as Tbl. 3 shows, CV is much faster than their transcript verification.