On the Size of Pairing-based Non-interactive Arguments[⋆]

Jens Groth[⋆⋆]

University College London, UK
j.groth@ucl.ac.uk

Abstract. Non-interactive arguments enable a prover to convince a verifier that a statement is true. Recently there has been a lot of progress both in theory and practice on constructing highly efficient non-interactive arguments with small size and low verification complexity, so-called succinct non-interactive arguments (SNARGs) and succinct non-interactive arguments of knowledge (SNARKs).
Many constructions of SNARGs rely on pairing-based cryptography. In these constructions a proof consists of a number of group elements and the verification consists of checking a number of pairing product equations. The question we address in this article is how efficient pairing-based SNARGs can be.
Our first contribution is a pairing-based (preprocessing) SNARK for arithmetic circuit satisfiability, which is an NP-complete language. In our SNARK we work with asymmetric pairings for higher efficiency, a proof is only 3 group elements, and verification consists of checking a single pairing product equations using 3 pairings in total. Our SNARK is zero-knowledge and does not reveal anything about the witness the prover uses to make the proof.
As our second contribution we answer an open question of Bitansky, Chiesa, Ishai, Ostrovsky and Paneth (TCC 2013) by showing that 2-move linear interactive proofs cannot have a linear decision procedure. It follows from this that SNARGs where the prover and verifier use generic asymmetric bilinear group operations cannot consist of a single group element. This gives the first lower bound for pairing-based SNARGs. It remains an intriguing open problem whether this lower bound can be extended to rule out 2 group element SNARGs, which would prove optimality of our 3 element construction.

Keywords: SNARKs, non-interactive zero-knowledge arguments, linear interactive proofs, quadratic arithmetic programs, bilinear groups.

1 Introduction

Goldwasser, Micali and Rackoff [GMR89] introduced zero-knowledge proofs that enable a prover to convince a verifier that a statement is true without revealing anything else. They have three core properties:

Completeness: Given a statement and a witness, the prover can convince the verifier.
Soundness: A malicious prover cannot convince the verifier of a false statement.
Zero-knowledge: The proof does not reveal anything but the truth of the statement, in particular it does not reveal the prover's witness.

⋆ ©IACR 2016. This article is a minor revision of the version published by Springer-Verlag at http://dx.doi.org/10.1007/978-3-662-49896-5_11

⋆⋆ The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 307937 and the Engineering and Physical Sciences Research Council grant EP/J009520/1. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

Blum, Feldman and Micali [BFM88] extended the notion to *non-interactive* zero-knowledge (NIZK) proofs in the common reference string model. NIZK proofs are useful in the construction of non-interactive cryptographic schemes, e.g., digital signatures and CCA-secure public key encryption.

The amount of communication is an important performance parameter for zero-knowledge proofs. Kilian [Kil92] gave the first sublinear communication zero-knowledge argument that sends fewer bits than the size of the statement to be proved. Micali [Mic00] proposed sublinear size arguments by letting the prover in a communication efficient argument compute the verifier's challenges using a cryptographic function, and as remarked in Kilian [Kil95] this leads to sublinear size NIZK proofs when the interactive argument is public coin and zero-knowledge.

Groth, Ostrovsky and Sahai [GOS12,GOS06,Gro06,GS12] introduced pairing-based NIZK proofs, yielding the first linear size proofs based on standard assumptions. Groth [Gro10] combined these techniques with ideas from interactive zero-knowledge arguments [Gro09] to give the first constant size NIZK arguments. Lipmaa [Lip12] used an alternative construction based on progression-free sets to reduce the size of the common reference string.

Groth's constant size NIZK argument is based on constructing a set of polynomial equations and using pairings to efficiently verify these equations. Gennaro, Gentry, Parno and Raykova [GGPR13] found an insightful construction of polynomial equations based on Lagrange interpolation polynomials yielding a pairing-based NIZK argument with a common reference string size proportional to the size of the statement and witness. They gave two types of polynomial equations: quadratic span programs for proving boolean circuit satisfiability and quadratic arithmetic programs for proving arithmetic circuit satisfiability. Lipmaa [Lip13] suggested more efficient quadratic span programs using error correcting codes, and Danezis, Fournet, Groth and Kohlweiss [DFGK14] refined quadratic span programs to square span programs that give NIZK arguments consisting of 4 group elements for boolean circuit satisfiability.

Exciting work on implementation has followed the above theoretical advances [PHGR13,BCG$^+$13,BFR$^+$13,BCTV14b,KPP$^+$14,BBFR15,CTV15,WSR$^+$15,CFH$^+$15,SVdV16]. Most efficient implementations refine the quadratic arithmetic program approach of Gennaro et al. [GGPR13] and combine it with a compiler producing a suitable quadratic arithmetic program that is equivalent to the statement to be proven; libsnark [BCTV14b,BSCG$^+$14] also includes an NIZK argument based on [DFGK14].

One powerful motivation for building efficient non-interactive arguments is verifiable computation. A client can outsource a complicated computational task to a server in the cloud and get back the results. To convince the client that the computation is correct the server may include a non-interactive argument of correctness with the result. However, since the verifier does not have many computational resources this only makes sense if the argument is compact and computationally light to verify, i.e., it is a succinct non-interactive argument (SNARG) or a succinct non-interactive argument of knowledge (SNARK). While pairing-based SNARGs are efficient for the verifier, the computational overhead for the prover is still orders of magnitude too high to warrant use in outsourced computation [WB15,Wal15] and further efficiency improvements are needed. In their current state, SNARKs that are zero-knowledge already have uses when proving statements about private data though. Zero-knowledge SNARKs are for instance key ingredients in the virtual currency proposals Pinocchio coin [DFKP13] and Zerocash [BCG$^+$14].

In parallel with developments in pairing-based NIZK arguments there has been interesting work on understanding SNARKs. Gentry and Wichs [GW11] showed that SNARGs must necessarily rely on non-falsifiable assumptions, and Bitansky et al. [BCCT12] proved designated verifier SNARKs exist if and only if extractable collision-resistant hash functions exist. Of particular interest in terms of efficiency is a series of works studying how SNARKs compose [Val08,BCCT13,BCTV14a]. They show among other things that a preprocessing SNARK with a long common reference string can be used to build a fully succinct SNARK with a short common reference string.

Bitansky et al. [BCI$^+$13] give an abstract model of SNARKs that rely on linear encodings of field elements. Their information theoretic framework called linear interactive proofs (LIPs) capture proof systems where the prover is restricted to using linear operations in computing her messages. They give a generic conversion of a 2-move LIP to a publicly verifiable SNARK using pairing-based techniques or to a designated verifier using additively homomorphic encryption techniques.

1.1 Our contribution

Succinct NIZK. We construct a NIZK argument for arithmetic circuit satisfiability where a proof consists of only 3 group elements. In addition to being small, the proof is also easy to verify. The verifier just needs to compute a number of exponentiations proportional to the statement size and check a single pairing product equation, which only has 3 pairings. Our construction can be instantiated with any type of pairings including Type III pairings, which are the most efficient pairings.

The argument has perfect completeness and perfect zero-knowledge. For soundness we take an aggressive stance and rely on a security proof in the generic bilinear group model [Sho97,Nec94] in order to get optimal performance. This stance is partly justified by Gentry and Wichs [GW11] that rule out SNARGs based on standard falsifiable assumptions. However, following Abe, Groth, Ohkubo and Tibouchi [AGOT14] we do provide a hedge against cryptanalysis by proving our construction secure in the symmetric pairing setting. For optimal efficiency it makes sense to use our NIZK argument in the asymmetric setting, however, by providing a security proof in the symmetric setting we get additional security: even if cryptanalytic advances yield a hitherto unknown efficiently computable isomorphism between the source groups this does not necessarily lead to a break of our scheme. We therefore have a unified NIZK argument that can be instantiated with any type of pairing, yielding both optimal efficiency and optimal generic bilinear group resilience.

We give a performance comparison for boolean circuit satisfiability in Table 1 and for arithmetic circuit satisfiability in Table 2 of the size of the common reference string (CRS), the size of the proof, the prover's computation, the verifier's computation, and the number of pairing product equations used to verify a proof. We perform better than the state of the art on all efficiency parameters.

In both comparisons the number of wires exceeds the number of gates, $m \geq n$, since each gate has an output wire. We expect for typical cases that the statement size ℓ will be small compared to m and n. In both tables, we have excluded the size of representing

	CRS size	Proof size	Prover comp.	Verifier comp.	PPE
[DFGK14]	$2m+n-2\ell\ \mathbb{G}_1\ ,\ m+n-\ell\ \mathbb{G}_2$	$3\ \mathbb{G}_1\ ,\ 1\ \mathbb{G}_2$	$m+n-\ell\ E_1$	$\ell\ M_1\ ,\ 6\ P$	3
This work	$3m+n\ \mathbb{G}_1\ ,\ m\ \mathbb{G}_2$	$2\ \mathbb{G}_1\ ,\ 1\ \mathbb{G}_2$	$n\ E_1$	$\ell\ M_1\ ,\ 3\ P$	1

Table 1. Comparison for boolean circuit satisfiability with ℓ-bit statement, m wires and n fan-in 2 logic gates. Notation: $\mathbb{G}$ means group elements, M means multiplications, E means exponentiations and P means pairings with subscripts indicating the relevant group. It is possible to get a CRS size of $m+2n$ elements in $\mathbb{G}_1$ and n elements in $\mathbb{G}_2$ but we have chosen to include some precomputed values in the CRS to reduce the prover's computation, see Sect. 3.2.

	CRS size	Proof size	Prover comp.	Verifier comp.	PPE
[PHGR13]	$7m+n-2\ell\ \mathbb{G}$	$8\ \mathbb{G}$	$7m+n-2\ell\ E$	$\ell\ E\ ,\ 11\ P$	5
This work	$m+2n\ \mathbb{G}$	$3\ \mathbb{G}$	$m+3n-\ell\ E$	$\ell\ E\ ,\ 3\ P$	1
[BCTV14a]	$6m+n+\ell\ \mathbb{G}_1\ ,\ m\ \mathbb{G}_2$	$7\ \mathbb{G}_1\ ,\ 1\ \mathbb{G}_2$	$6m+n-\ell\ E_1\ ,\ m\ E_2$	$\ell\ E_1\ ,\ 12\ P$	5
This work	$m+2n\ \mathbb{G}_1\ ,\ n\ \mathbb{G}_2$	$2\ \mathbb{G}_1\ ,\ 1\ \mathbb{G}_2$	$m+3n-\ell\ E_1\ ,\ n\ E_2$	$\ell\ E_1\ ,\ 3\ P$	1

Table 2. Comparison for arithmetic circuit satisfiability with ℓ-element statement, m wires, n multiplication gates. Notation: $\mathbb{G}$ means group elements, E means exponentiations and P means pairings. We compare symmetric pairings in the first two rows and asymmetric pairings in the last two rows.

the relation for which we give proofs. In the boolean circuit satisfiability case, we are considering arbitrary fan-in 2 logic gates. In the arithmetic circuit satisfiability case we work with fan-in 2 multiplication gates where each input factor can be a weigthed sum of other wires. We assume each multiplication gate input depends on a constant number of wires; otherwise the cost of evaluating the relation itself may exceed the cost of the subsequent proof generation.

We note that [PHGR13] uses symmetric bilinear groups where $\mathbb{G}_1 = \mathbb{G}_2$ and we are therefore comparing with a symmetric bilinear group instantiation of our scheme, which saves n elements in the common reference string. However, in the implementation of their system, called Pinocchio, asymmetric pairings are used for better efficiency. The switch to asymmetric pairings only requires minor modifications, see e.g. [BCTV14a] for a specification of such a SNARK, which has been implemented in the libsnark library.

Size matters. While the reduction in proof size to 3 group elements and the reduction in verification time is nice in itself, we would like to highlight that it is particularly important when composing SNARKs. [BCCT13,BCTV14a] show that preprocessing SNARKs with a long CRS can be composed to yield fully succinct SNARKs with a short CRS.[1] The transformations split the statement into smaller pieces, prove each piece is correct by itself, and recursively construct proofs of knowledge of other proofs that jointly show the pieces are correct and fit together. In the recursive construction of proofs, it is extra beneficial when the proofs are small and easy to verify since the resulting statements "there exists a proof satisfying the verification equation..." become small themselves. So we gain both from the prover's lower computation and from the

[1] We remark that soundness against generic adversaries is not preserved under composition (an issue that also appears in [Val08]), since the composition needs a concrete instantiation of the bilinear groups when writing out the statements corresponding to verification of another SNARK. What we are saying is that if our SNARK is knowledge sound in the *standard model* then we can use recursion to get fully succinct SNARKs.

fact that the statements in the recursive composition are smaller since we have a more efficient verification procedure for our SNARK. Chiesa and Virza [CV16] report a factor 4-5 speedup from using our SNARKs in the implementation of [BCTV14a].

Technique. All pairing-based SNARKs in the literature follow a common paradigm where the prover computes a number of group elements using generic group operations and the verifier checks the proof using a number of pairing product equations. Bitansky et al. [BCI$^+$13] formalize this paradigm through the definition of linear interactive proofs (LIPs). A linear interactive proof works over a finite field and the prover's and verifier's messages consist of vectors of field elements. It furthermore requires that the prover computes her messages using only linear operations. Once we have an approriate 2-move LIP, it can be compiled into a SNARK by executing the equations "in the exponent" using pairing-based cryptography. One source of our efficiency gain is that we design a LIP system for arithmetic circuits where the prover only sends 3 field elements. In comparison, the quadratic arithmetic programs by [GGPR13,PHGR13] correspond to LIPs where the prover sends 4 field elements.

A second source of efficiency gain compared to previous work is a more aggressive compilation of the LIP. Bitansky et al. [BCI$^+$13] propose a transformation in the symmetric bilinear group setting, where each field element gets compiled into two group elements. They then use a knowledge of exponent assumption to argue that the prover knows the relevant field elements. A less conservative choice would be to compile each field element into a single group element. Compiling with a single group element per field element improves efficiency but we only prove security generic group model [Sho97,BBG05] since we can no longer use the knowledge of exponent assumption. It is also possible to make a choice between these two extremes, Parno et al. [PHGR13] for instance have a LIP with 4 field elements, which gets compiled into 7 group elements. To summarize, in this paper we have opted for maximal efficiency and compile each field element in the LIP into a single group element and argue security in the generic group model.

We prefer to work with asymmetric bilinear groups for their higher efficiency than symmetric bilinear groups. This means that there is more to the story than the number of field elements the prover sends in the LIP and the choice of how aggressive a compilation we use. When working with asymmetric bilinear groups, a field element can appear as an exponent in the first source group, the second source group, or both. Our LIP is carefully designed such that each field element gets compiled into a single source group element in order to minimize the proof size to 3 group elements in total.

Lower bounds. Working towards ever more efficient non-interactive arguments, it is natural to ask what the minimal proof size is. We will show that pairing-based SNARGs with a single group element proof cannot exist. This result relates to an open question raised by Bitansky et al. [BCI$^+$13], whether there are LIPs with a linear decision procedure for the verifier. Such a linear decision procedure would be quite useful; it could for instance enable the construction of SNARGs based on ElGamal encryption.

We answer this open problem negatively by proving that LIPs with a linear decision procedure do not exist. A consequence of this is that any pairing-based SNARG must pair group elements from the proof together to make the decision procedure quadratic instead of linear. Working over asymmetric bilinear groups we must therefore have elements in both source groups in order to do such a pairing. This rules out the existence of 1 group element SNARGs, regardless of whether it is zero-knowledge or not, and shows our NIZK argument has close to optimal proof size. It remains an intriguing open problem to completely close the gap by either constructing a SNARG with exactly one element from each source group $\mathbb{G}_1$ and $\mathbb{G}_2$, or alternatively rule out the existence of such a SNARG.

2 Preliminaries

Given two functions $f, g : \mathbb{N} \to [0,1]$ we write $f(\lambda) \approx g(\lambda)$ when $|f(\lambda) - g(\lambda)| = \lambda^{-\omega(1)}$. We say that f is *negligible* when $f(\lambda) \approx 0$ and that f is *overwhelming* when $f(\lambda) \approx 1$. We will use λ to denote a security parameter, with the intuition that as λ grows we expect stronger security.

We write $y = A(x; r)$ when algorithm A on input x and randomness r, outputs y. We write $y \leftarrow A(x)$ for the process of picking randomness r at random and setting $y = A(x; r)$. We also write $y \leftarrow S$ for sampling y uniformly at random from the set S. We will assume it is possible to sample uniformly at random from sets such as $\mathbb{Z}_p$.

Following Abe and Fehr [AF07] we write $(y; z) \leftarrow (\mathcal{A} \parallel \mathcal{X}_{\mathcal{A}})(x)$ when $\mathcal{A}$ on input x outputs y, and $\mathcal{X}_{\mathcal{A}}$ on the same input (including random coins) outputs z.

2.1 Bilinear groups

We will work over bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ with the following properties:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p
- The pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map
- g is a generator for $\mathbb{G}_1$, h is a generator for $\mathbb{G}_2$, and $e(g, h)$ is a generator for $\mathbb{G}_T$
- There are efficient algorithms for computing group operations, evaluating the bilinear map, deciding membership of the groups, deciding equality of group elements and sampling generators of the groups. We refer to these as the generic group operations.

There are many ways to set up bilinear groups both as symmetric bilinear groups where $\mathbb{G}_1 = \mathbb{G}_2$ and as asymmetric bilinear groups where $\mathbb{G}_1 \neq \mathbb{G}_2$. Galbraith, Paterson and Smart [GPS08] classify bilinear groups as Type I where $\mathbb{G}_1 = \mathbb{G}_2$, Type II where there is an efficiently computable non-trivial homomorphism $\Psi : \mathbb{G}_2 \to \mathbb{G}_1$, and Type III where no such efficiently computable homomorphism exists in either direction between $\mathbb{G}_1$ and $\mathbb{G}_2$. Type III bilinear groups are the most efficient type of bilinear groups and hence the most relevant for practical applications. We give lower bound for pairing-based SNARGs in Type III bilinear groups. Our constructions on the other hand can be instantiated in all 3 types of bilinear groups.

It will be useful to use a notation that represents group elements by their discrete logarithms. We stress the discrete logarithms are hard to compute, this notation is just convenient for representational purposes. We write $[a]_1$ for g^a, $[b]_2$ for h^b, and $[c]_T$ for $e(g, h)^c$. With this notation $g = [1]_1$, $h = [1]_2$ and $e(g, h) = [1]_T$, while the neutral elements are $[0]_1, [0]_2$ and $[0]_T$. Working with the discrete logarithm representation of group it is natural to use additive notation in all groups, so for instance $[a]_T + [b]_T =$

$[a+b]_T$. A vector of group elements will be represented as $[\boldsymbol{a}]_i$. Our notation allows us to define natural operations using standard linear algebra notation, so $[\boldsymbol{a}]_i + [\boldsymbol{b}]_i = [\boldsymbol{a}+\boldsymbol{b}]_i$ assuming $\boldsymbol{a}$ and $\boldsymbol{b}$ have the same dimension, and also assuming appropriate dimension we define $A[\boldsymbol{b}]_i = [A\boldsymbol{b}]_i$. Given two vectors of n group elements $[\boldsymbol{a}]_1$ and $[\boldsymbol{b}]_2$, we define their dot product as $[\boldsymbol{a}]_1 \cdot [\boldsymbol{b}]_2 = [\boldsymbol{a} \cdot \boldsymbol{b}]_T$, which can be efficiently computed using the pairing e.

We say an algorithm is generic if it only uses generic group operations to create and manipulate group elements. Shoup [Sho97] formalized the generic group model by considering random injective encodings $[\cdot]_i$ instead of real group elements. Generic group operations are then handled through an oracle the algorithm has access to, e.g., it can for instance on (add, $[a]_i$, $[b]_i$) return $[a+b]_i$. Due to the randomness of the encoding, the generic algorithm can only do meaningful operations through the generic group oracle. One implication of this is that if it has input $[\boldsymbol{a}]_1$ and return elements $[\boldsymbol{b}]$, we can by checking the addition queries it has made in $\mathbb{G}_1$ efficiently deduce a matrix M such that $\boldsymbol{b} = M\boldsymbol{a}$. The same holds in $\mathbb{G}_2$, while in $\mathbb{G}_T$ there may also be elements computed from the pairing operation, but we can still write any output element as an explicit quadratic polynomial in the inputs.

2.2 Non-interactive zero-knowledge arguments of knowledge

Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable binary relation R. For pairs $(\phi, w) \in R$ we call ϕ the statement and w the witness. We define $\mathcal{R}_\lambda$ to be the set of possible relations R the relation generator may output given 1^λ. We will in the following for notational simplicity assume λ can be deduced from the description of R. The relation generator may also output some side information, an auxiliary input z, which will be given to the adversary. An efficient prover publicly verifiable non-interactive argument for $\mathcal{R}$ is a quadruple of probabilistic polynomial algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ such that

$(\sigma, \tau) \leftarrow \mathsf{Setup}(R)$: The setup produces a common reference string σ and a simulation trapdoor τ for the relation R.

$\pi \leftarrow \mathsf{Prove}(R, \sigma, \phi, w)$: The prover algorithm takes as input a common reference string σ and $(\phi, w) \in R$ and returns an argument π.

$0/1 \leftarrow \mathsf{Vfy}(R, \sigma, \phi, \pi)$: The verification algorithm takes as input a common reference string σ, a statement ϕ and an argument π and returns 0 (reject) or 1 (accept).

$\pi \leftarrow \mathsf{Sim}(R, \tau, \phi)$: The simulator takes as input a simulation trapdoor and statement ϕ and returns an argument π.

Definition 1. *We say* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy})$ *is a non-interactive argument for* $\mathcal{R}$ *if it has perfect completeness and computational soundness as defined below.*

Definition 2. *We say* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ *is a perfect non-interactive zero-knowledge argument of knowledge for* $\mathcal{R}$ *if it has perfect completeness, perfect zero-knowledge and computational knowledge soundness as defined below.*

Perfect completeness. Completeness says that, given any true statement, an honest prover should be able to convince an honest verifier. For all $\lambda \in \mathbb{N}$, $R \in \mathcal{R}_\lambda$, $(\phi, w) \in R$

$$\Pr\left[(\sigma, \tau) \leftarrow \mathsf{Setup}(R); \pi \leftarrow \mathsf{Prove}(R, \sigma, \phi, w) : \mathsf{Vfy}(R, \sigma, \phi, \pi) = 1\right] = 1.$$

Perfect zero-knowledge. An argument is zero-knowledge if it does not leak any information besides the truth of the statement. We say $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ is perfect zero-knowledge if for all $\lambda \in \mathbb{N}$, $(R, z) \leftarrow \mathcal{R}(1^\lambda)$, $(\phi, w) \in R$ and all adversaries $\mathcal{A}$

$$\Pr\Big[(\sigma,\tau) \leftarrow \mathsf{Setup}(R); \pi \leftarrow \mathsf{Prove}(R,\sigma,\phi,w) : \mathcal{A}(R,z,\sigma,\tau,\pi) = 1\Big]$$
$$= \Pr\Big[(\sigma,\tau) \leftarrow \mathsf{Setup}(R); \pi \leftarrow \mathsf{Sim}(R,\tau,\phi) : \mathcal{A}(R,z,\sigma,\tau,\pi) = 1\Big].$$

Computational soundness. We say $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ is sound if it is not possible to prove a false statement, i.e., convince the verifier if no witness exists. Let L_R be the language consisting of statements for which there exist matching witnesses in R. Formally, we require that for all non-uniform polynomial time adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c}(R,z) \leftarrow \mathcal{R}(1^\lambda); (\sigma,\tau) \leftarrow \mathsf{Setup}(R); (\phi,\pi) \leftarrow \mathcal{A}(R,z,\sigma) : \\ \phi \notin L_R \text{ and } \mathsf{Vfy}(R,\sigma,\phi,\pi) = 1\end{array}\right] \approx 0.$$

Computational knowledge soundness. Strengthening the notion of soundness, we call $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ an argument of knowledge if there is an extractor that can compute a witness whenever the adversary produces a valid argument. The extractor gets full access to the adversary's state, including any random coins. Formally, we require that for all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a non-uniform polynomial time extractor $\mathcal{X}_\mathcal{A}$ such that

$$\Pr\left[\begin{array}{c}(R,z) \leftarrow \mathcal{R}(1^\lambda); (\sigma,\tau) \leftarrow \mathsf{Setup}(R); ((\phi,\pi);w) \leftarrow (\mathcal{A} \parallel \mathcal{X}_\mathcal{A})(R,z,\sigma) : \\ (\phi,w) \notin R \text{ and } \mathsf{Vfy}(R,\sigma,\phi,\pi) = 1\end{array}\right] \approx 0.$$

Public verifiability and designated verifier proofs. We can naturally generalize the definition of a non-interactive argument by splitting σ into two parts σ_P and σ_V used by the prover and verifier respectively. We say the non-interactive argument is publicly verifiable when σ_V can be deduced from σ_P. Otherwise we refer to it as a designated verifier argument. For designated verifier arguments it is possible to relax soundness and knowledge soundness such that the adversary only sees σ_P but not σ_V.

SNARGs and SNARKs. A non-interactive argument where the verifier runs in polynomial time in $\lambda + |\phi|$ and the proof size is polynomial in λ is called a preprocessing succinct non-interactive argument (SNARG) if it sound, and a preprocessing succinct argument of knowledge (SNARK) if it is knowledge sound. If we also restrict the common reference string to be polynomial in λ we say the non-interactive argument is a fully succinct SNARG or SNARK. Bitansky et al. [BCCT13] show that preprocessing SNARKs can be composed to yield fully succinct SNARKs. The focus of this paper is on preprocessing SNARKs, where the common reference string may be long.

Benign relation generators. Bitansky et al. [BCPR14] show that indistinguishability obfuscation implies that for every candidate SNARK there are auxiliary output distributions that enable the adversary to create a valid proof without it being possible to extract the witness. Assuming also public coin differing input obfuscation and other cryptographic assumptions, Boyle and Pass [BP15] strengthen this impossibility to show that there is an auxiliary output distribution that defeats witness extraction for all candidate SNARKs. These counter examples, however, rely on specific auxiliary input distributions. We will therefore in the following assume the relationship generator is *benign* in the sense that the relation and the auxiliary input are distributed in such a way that the SNARKs we construct can be knowledge sound.

2.3 Quadratic arithmetic programs

Consider an arithmetic circuit consisting of addition and multiplication gates over a finite field $\mathbb{F}$. We may designate some of the input/output wires as specifying a statement and use the rest of the wires in the circuit to define a witness. This gives us a binary relation R consisting of statement wires and witness wires that satisfy the arithmetic circuit, i.e., make it consistent with the designated input/output wires.

Generalizing arithmetic circuits, we may be interested in relations described by equations over a set of variables. Some of the variables correspond to the statement; the remaining variables correspond to the witness. The relation consists of statements and witnesses that satisfy all the equations. The equations will be over $a_0 = 1$ and variables $a_1, \dots, a_m \in \mathbb{F}$ and be of the form

$$\sum a_i u_{i,q} \cdot \sum a_i v_{i,q} = \sum a_i w_{i,q},$$

where $u_{i,q}, v_{i,q}, w_{i,q}$ are constants in $\mathbb{F}$ specifying the qth equation.

We observe that addition and multiplication gates are special cases of such equations so such systems of arithmetic constraints do indeed generalize arithmetic circuits. A multiplication gate can for instance be described as $a_i \cdot a_j = a_k$ (using $u_i = 1, v_j = 1$ and $w_k = 1$ and setting the remaining constants for this gate to 0). Addition gates are handled for free in the sums defining the equations, i.e., if $a_i + a_j = a_k$ and a_k is multiplied by a_ℓ, we may simply write $(a_i + a_j) \cdot a_\ell$ and skip the calculation of a_k.

Following Gennaro, Gentry, Parno and Raykova [GGPR13] we can reformulate the set of arithmetic constraints as a quadratic arithmetic program assuming $\mathbb{F}$ is large enough. Given n equations we pick arbitrary distinct $r_1, \dots, r_n \in \mathbb{F}$ and define $t(x) = \prod_{q=1}^{n}(x - r_q)$. Furthermore, let $u_i(x), v_i(x), w_i(x)$ be degree $n-1$ polynomials such that

$$u_i(r_q) = u_{i,q} \qquad v_i(r_q) = v_{i,q} \qquad w_i(r_q) = w_{i,q} \qquad \text{for} \quad i = 0, \dots, m, q = 1, \dots, n.$$

We now have that $a_0 = 1$ and the variables $a_1, \dots, a_m \in \mathbb{F}$ satisfy the n equations if and only if in each point $r_1, \dots, r_q$

$$\sum_{i=0}^{m} a_i u_i(r_q) \cdot \sum_{i=0}^{m} a_i v_i(r_q) = \sum_{i=0}^{m} a_i w_i(r_q).$$

Since $t(X)$ is the lowest degree monomial with $t(r_q) = 0$ in each point, we can reformulate this condition as

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) \equiv \sum_{i=0}^{m} a_i w_i(X) \mod t(X).$$

Formally, we will be working with quadratic arithmetic programs R that have the following description

$$R = \left(\mathbb{F}, \text{aux}, \ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)\right),$$

where $\mathbb{F}$ describes a finite field, aux is some auxiliary information, $1 \leq \ell \leq m$, $u_i(X), v_i(X), w_i(X), t(X) \in \mathbb{F}[X]$ and $u_i(X), v_i(X), w_i(X)$ have strictly lower degree than n, the degree of $t(X)$. A quadratic arithmetic program with such a description defines the following binary relation, where we define $a_0 = 1$,

$$R = \left\{ (\phi, w) \left| \begin{array}{l} \phi = (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell \\ w = (a_{\ell+1}, \ldots, a_m) \in \mathbb{F}^{m-\ell} \\ \\ \sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) \equiv \sum_{i=0}^m a_i w_i(X) \mod t(X) \end{array} \right. \right\}.$$

We say $\mathcal{R}$ is a quadratic arithmetic program generator if it generates relations of the form given above with fields of size larger than $2^{\lambda-1}$.

Relations can arise in many different ways in practice. It may be that the relationship generator is deterministic or it may be that it is randomized. It may be that first the field $\mathbb{F}$ is generated and then the rest of the relation is built on top of the field. Or it may be that the polynomials are specified first and then a random field is chosen. To get maximal flexibility we have chosen our definitions to be agnostic with respect to the exact way the field and the relation is generated, the different options can all be modelled by appropriate choices of relation generators.

Looking ahead, we will in our pairing-based NIZK arguments let the auxiliary information aux specify a bilinear group. It may seem a bit surprising to make the choice of bilinear group part of the relation generator but this provides a better model of settings where the relation is built on top of an already existing bilinear group. Again, there is no loss of generality in this choice, one can think of a traditional setting where the relation is chosen first and then the bilinear group is chosen at random as the special case where the relation generator works in two steps, first choosing the relation and then picking a random bilinear group. Of course letting the relation generator pick the bilinear group is another good reason that we need to assume it is benign; an appropriate choice of bilinear group is essential for security.

2.4 Linear non-interactive proofs

Bitansky et al. [BCI+13] give a useful characterization of the information theoretic underpinning of recent SNARK constructions that they call 2-move algebraic input-oblivious linear interactive proofs. To clarify the connection to non-interactive arguments that we defined in Section 2.2, we will rename this notion as non-interactive linear proofs (NILP). NILPs are defined relative to a relation generator $\mathcal{R}$, where we assume the relations specify a finite field $\mathbb{F}$, and work as follows.

$(\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R)$**:** The setup is a probabilistic polynomial time algorithm that returns vectors $\boldsymbol{\sigma} \in \mathbb{F}^m$ and $\boldsymbol{\tau} \in \mathbb{F}^n$. We will for notational simplicity assume that $\boldsymbol{\sigma}$ always contains 1 as an entry such that there is no distinction between affine and linear functions of $\boldsymbol{\sigma}$.

$\boldsymbol{\pi} \leftarrow \mathsf{Prove}(R, \boldsymbol{\sigma}, \phi, w)$**:** The prover operates in two stages:
- First it runs $\Pi \leftarrow \mathsf{ProofMatrix}(R, \phi, w)$, where $\mathsf{ProofMatrix}$ is a probabilistic polynomial time algorithm that generates a matrix $\Pi \in \mathbb{F}^{k \times m}$.
- Then it computes the proof as $\boldsymbol{\pi} = \Pi \boldsymbol{\sigma}$.

$0/1 \leftarrow \mathsf{Vfy}(R, \boldsymbol{\sigma}, \phi, \boldsymbol{\pi})$**:** The verifier runs in two stages:

- First it runs a deterministic polynomial time algorithm $\boldsymbol{t} \leftarrow \mathsf{Test}(R, \phi)$ to get an arithmetic circuit $\boldsymbol{t} : \mathbb{F}^{m+k} \rightarrow \mathbb{F}^{\eta}$ corresponding to the evaluation of a vector of multi-variate polynomials of total degree d.
- It then accepts the proof if and only if $\boldsymbol{t}(\boldsymbol{\sigma}, \boldsymbol{\pi}) = \mathbf{0}$.

The degree d and the dimensions μ, m, n, k, η may be constants or polynomials in the security parameter λ.

Definition 3 (Linear non-interactive proof). *The tuple* (Setup, Prove, Vfy) *is a linear non-interactive proof for* $\mathcal{R}$ *if it has perfect completeness and statistical knowledge soundness against affine prover strategies as defined below.*

Statistical knowledge soundness against affine prover strategies. A NILP has knowledge soundness against affine prover strategies if a witness can be extracted from a successful proof matrix Π. More precisely, there is a polynomial time extractor $\mathcal{X}$ such that for all adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c}(R, z) \leftarrow \mathcal{R}(1^{\lambda}); (\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R); (\phi, \Pi) \leftarrow \mathcal{A}(R, z); w \leftarrow \mathcal{X}(R, \phi, \Pi) : \\ \Pi \in \mathbb{F}^{m \times k} \ \wedge \ \mathsf{Vfy}(R, \boldsymbol{\sigma}, \phi, \Pi\boldsymbol{\sigma}) = \mathbf{0} \ \wedge \ (\phi, w) \notin R\end{array}\right] \approx 0.$$

The notion of zero-knowledge from Section 2.2 also applies to NILPs and corresponds to honest-verifier zero-knowledge for a 2-move LIP. Another potential extension is to designated-verifier NILPs where the common reference string $\boldsymbol{\sigma}$ is split into two parts $\boldsymbol{\sigma}_P$ used by the prover and $\boldsymbol{\sigma}_V$ used by the verifier.

2.5 Non-interactive arguments from linear non-interactive proofs.

NILPs are useful because they can be compiled into publicly verifiable non-interactive arguments using pairings and designated verifier non-interactive arguments using a variant of Paillier encryption [BCI+13]. If we work in the pairing setting, the intuition is that a NILP with verifier degree $d = 2$ can be executed "in the discrete logarithms". The common reference string contains encodings of field elements in $\boldsymbol{\sigma}$. The prover computes the proof as linear combinations of group elements in the common reference string. The verifier checks the argument by verifying a number of pairing product equations (equations formed by multiplying together the results of pairings), which corresponds to checking quadratic equations in the encoded field elements. We will now formalize this methodology.

When working with Type III pairings, executing the NILP in the discrete logarithms requires that we specify for each element in which group the operations should take place. We will therefore define a split NILP, which is a NILP where the common reference string can be split into two parts $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2)$ and the prover's proof can be split into two parts $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$. Each part of the proof is computed from the corresponding part of the common reference string. Finally, when verifying the proof, we want the verifier's test to be a quadratic equation where each variable has degree 1. Writing it out, a split NILP is a NILP of the following form:

$(\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R)$**:** The setup algorithm generates vectors $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2) \in \mathbb{F}^{m_1} \times \mathbb{F}^{m_2}$ and $\boldsymbol{\tau} \in \mathbb{F}^n$. We will for notational simplicity assume $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ both contain 1 as an entry such that there is no distinction between affine and linear functions of $\boldsymbol{\sigma}$.

$\boldsymbol{\pi} \leftarrow \mathsf{Prove}(R, \boldsymbol{\sigma}, \phi, w)$**:** The prover operates in two stages:

- First it runs $\Pi \leftarrow \mathsf{ProofMatrix}(R, \phi, w)$, where we require $\mathsf{ProofMatrix}$ generates a matrix of the form $\Pi = \begin{pmatrix} \Pi_1 & 0 \\ 0 & \Pi_2 \end{pmatrix}$, where $\Pi_1 \in \mathbb{F}^{k_1 \times m_1}$ and $\Pi_2 \in \mathbb{F}^{k_2 \times m_2}$.
- Then it computes $\boldsymbol{\pi}_1 = \Pi_1 \boldsymbol{\sigma}_1$ and $\boldsymbol{\pi}_2 = \Pi_2 \boldsymbol{\sigma}_2$ and returns $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$.

$0/1 \leftarrow \mathsf{Vfy}(R, \boldsymbol{\sigma}, \phi, \boldsymbol{\pi})$**:** The verifier runs in two stages:

- First it runs $\boldsymbol{t} \leftarrow \mathsf{Test}(R, \phi)$ to get an arithmetic circuit $\boldsymbol{t} : \mathbb{F}^{m_1+k_1+m_2+k_2} \to \mathbb{F}^\eta$ corresponding to matrices $T_1, \ldots, T_\eta \in \mathbb{F}^{(m_1+k_1)\times(m_2+k_2)}$.
- It then accepts the proof if and only if for all matrices $T_1, \ldots, T_\eta$

$$\begin{pmatrix} \boldsymbol{\sigma}_1 \\ \boldsymbol{\pi}_2 \end{pmatrix} \cdot T_i \begin{pmatrix} \boldsymbol{\sigma}_2 \\ \boldsymbol{\pi}_2 \end{pmatrix} = 0.$$

Intuitively, after compiling the split NILP we want to argue soundness by saying a cheating prover that uses generic group operations cannot deviate from the NILP. However, when the prover sees the common reference string, she may learn useful information from it and choose her matrix Π in a way that depends on it. To counter this type of adversary, we will define a *disclosure-free* common reference string as one where the prover does not gain useful information that can help her choose a special matrix Π.

Definition 4 (Disclosure-free NILP). *We say a split NILP is disclosure-free if for all adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{c}(R, z) \leftarrow \mathcal{R}(1^\lambda); T \leftarrow \mathcal{A}(R, z); (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\tau}), (\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2, \boldsymbol{\tau}') \leftarrow \mathsf{Setup}(R) : \\ \boldsymbol{\sigma}_1 \cdot T\boldsymbol{\sigma}_2 = 0 \text{ if and only if } \boldsymbol{\sigma}'_1 \cdot T\boldsymbol{\sigma}'_2 = 0\end{array}\right] \approx 1.$$

The way to interpret the definition of a disclosure-free common reference string is that the outcome of any test the adversary can run on $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2$ can be predicted by running it on an independently generated $\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2$.

We are now ready to describe a compiler that uses a split NILP $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{Sim})$ with disclosure-free common reference strings to give us a pairing-based non-interactive argument $(\mathsf{Setup}', \mathsf{Prove}', \mathsf{Vfy}', \mathsf{Sim}')$.

$(\sigma, \tau) \leftarrow \mathsf{Setup}'(R)$**:** Run $(\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R)$. Return $\sigma = ([\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2)$ and $\tau = \boldsymbol{\tau}$.

$\pi \leftarrow \mathsf{Prove}'(R, \sigma, \phi, w)$**:** Generate $(\Pi_1, \Pi_2) \leftarrow \mathsf{ProofMatrix}(R, x, w)$ and return $\pi = ([\boldsymbol{\pi}_1]_1, [\boldsymbol{\pi}_2]_2)$ computed as

$$[\boldsymbol{\pi}_1]_1 = \Pi_1 [\boldsymbol{\sigma}_1]_1 \qquad\qquad [\boldsymbol{\pi}_2]_2 = \Pi_2 [\boldsymbol{\sigma}_2]_2.$$

$0/1 \leftarrow \mathsf{Vfy}'(R, \sigma, \phi, \pi)$**:** Generate $(T_1, \ldots, T_\eta) \leftarrow \mathsf{Test}(R, \phi)$. Parse $\pi = ([\boldsymbol{\pi}_1]_1, [\boldsymbol{\pi}_2)]_2 \in \mathbb{G}_1^{k_1} \times \mathbb{G}_2^{k_2}$. Accept the proof if and only if for all $T_1, \ldots, T_\eta$

$$\begin{bmatrix} \boldsymbol{\sigma}_1 \\ \boldsymbol{\pi}_1 \end{bmatrix}_1 \cdot T_i \begin{bmatrix} \boldsymbol{\sigma}_2 \\ \boldsymbol{\pi}_2 \end{bmatrix}_2 = [0]_T.$$

$\pi \leftarrow \mathsf{Sim}'(R, \tau, \phi)$**:** Simulate $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) \leftarrow \mathsf{Sim}(R, \boldsymbol{\tau}, \phi)$ and return $\pi = ([\boldsymbol{\pi}_1]_1, [\boldsymbol{\pi}_2]_2)$.

Lemma 1. *The protocol given above is a non-interactive argument with perfect completeness and statistical knowledge soundness against generic adversaries that only do a polynomial number of generic group operations. It is perfect zero-knowledge if the underlying split NILP is perfect zero-knowledge.*

Proof. Perfect completeness follows from the perfect completeness of the NILP and the fact it is a split NILP, which allows the adversary to compute the two parts of the proof $[\boldsymbol{\pi}_1]_1, [\boldsymbol{\pi}_2]_2$ using generic group operations in the relevant groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

Perfect zero-knowledge follows from the perfect zero-knowledge property of the NILP.

It remains to argue statistical soundness against generic adversaries. A generic adversary can use the generic group operations to multiply elements in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, test membership of the groups, evaluate the pairing, and test whether elements are equal.

We first argue that the disclosure-freeness implies that the adversary has negligible probability of learning non-trivial information about the common reference string. Whenever an adversary tests whether an element computed using generic group operations is 0, it can be written out as a pairing product equality test of the form $[\boldsymbol{\sigma}_1]_1 \cdot T[\boldsymbol{\sigma}_2]_2 = [0]_T$, where the matrix T can be deduced from the generic group operation queries the adversary has made. Instead of making these queries, we could instead run a modified adversary that picks an alternative common reference string $(\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2, \boldsymbol{\tau}')$ and answers the queries herself by testing whether $\boldsymbol{\sigma}'_1 \cdot T\boldsymbol{\sigma}'_2 = 0$. By the disclosure-freeness, the answers made this way are with overwhelming probability identical to what the adversary would see on the real common reference string, so we can from now on assume the generic adversary does not make any zero tests on elements involving the common reference string.

An adversary that does not make any zero-tests on the common reference string and only uses generic group operations, is equivalent to an adversary that picks matrices Π_1, Π_2 independently of $[\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2$ and then computes the proofs as $[\pi_1]_1 = \Pi_1[\boldsymbol{\sigma}_1]$ and $[\pi_2]_2 = \Pi_2[\boldsymbol{\sigma}_2]_2$. Taking discrete logarithms, this corresponds exactly to running a split NILP knowledge soundness adversary to get matrices Π_1, Π_2 and proofs $\boldsymbol{\pi}_1 = \Pi_1\boldsymbol{\sigma}_1, \boldsymbol{\pi}_2 = \Pi_1\boldsymbol{\sigma}_2$.

Taking discrete logarithms of the verification equations, we see that if the adversary is successful in finding ϕ and a valid proof $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2$, this corresponds to finding ϕ and Π_1, Π_2 such that for the test matrices $T_1, \dots, T_\eta \leftarrow \mathsf{Test}(R, \phi)$

$$\begin{pmatrix} \boldsymbol{\sigma}_1 \\ \Pi_1\boldsymbol{\sigma}_1 \end{pmatrix} \cdot T_i \begin{pmatrix} \boldsymbol{\sigma}_2 \\ \Pi_2\boldsymbol{\sigma}_2 \end{pmatrix} = 0.$$

By the statistical soundness of the split NILP this has negligible probability of happening unless knowledge of Π_1, Π_2 enables the extraction of a witness w such that $(\phi, w) \in R$. □

The proof for Lemma 1 also holds if we use a split NILP that only has soundness against split affine adversaries that are restricted to outputting a matrix $\Pi = \begin{pmatrix} \Pi_1 & 0 \\ 0 & \Pi_2 \end{pmatrix}$. However, the split NILP we construct later will actually be secure against any choice of

Π. The advantage of this is a hedge against cryptanalysis, even if the adversary finds an efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ and even if it maps g to h, we will still have security in the generic group model. Another advantage is that the construction also works for symmetric bilinear groups with very minor changes.

3 Constructions of non-interactive arguments

We will construct a pairing-based NIZK argument for quadratic arithmetic programs where proofs consist of only 3 group elements. We give the construction in two steps, first we construct a NILP for quadratic arithmetic programs, and then we observe it is also a split NILP and convert it to pairing-based NIZK argument using the compilation technique we presented earlier.

3.1 Non-interactive linear proofs for quadratic arithmetic programs

We will now construct a NILP for quadratic arithmetic program generators that outputs relations of the form

$$R = (\mathbb{F}, \mathrm{aux}, \ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)) .$$

The relation defines a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{F}^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{F}^{m-\ell}$ such that with $a_0 = 1$

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X),$$

for some degree $n-2$ quotient polynomial $h(X)$, where n is the degree of $t(X)$.

$(\boldsymbol{\sigma}, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R)$**:** Pick $\alpha, \beta, \gamma, \delta, x \leftarrow \mathbb{F}^*$. Set $\boldsymbol{\tau} = (\alpha, \beta, \gamma, \delta, x)$ and

$$\boldsymbol{\sigma} = \left(\alpha, \beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1}, \left\{ \tfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} \right\}_{i=0}^{\ell}, \left\{ \tfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} \right\}_{i=\ell+1}^{m}, \left\{ \tfrac{x^i t(x)}{\delta} \right\}_{i=0}^{n-2} \right).$$

$\boldsymbol{\pi} \leftarrow \mathsf{Prove}(R, \boldsymbol{\sigma}, a_1, \ldots, a_m)$**:** Pick $r, s \leftarrow \mathbb{F}$ and compute a $3 \times (m + 2n + 4)$ matrix Π such that $\boldsymbol{\pi} = \Pi \boldsymbol{\sigma} = (A, B, C)$ where

$$A = \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta \qquad B = \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta$$

$$C = \frac{\sum_{i=\ell+1}^{m} a_i \left(\beta u_i(x) + \alpha v_i(x) + w_i(x) \right) + h(x)t(x)}{\delta} + As + rB - rs\delta.$$

$0/1 \leftarrow \mathsf{Vfy}(R, \boldsymbol{\sigma}, a_1, \ldots, a_\ell)$**:** Compute a quadratic multi-variate polynomial $\boldsymbol{t}$ such that $\boldsymbol{t}(\boldsymbol{\sigma}, \boldsymbol{\pi}) = 0$ corresponds to the test

$$A \cdot B = \alpha \cdot \beta + \frac{\sum_{i=0}^{\ell} a_i \left(\beta u_i(x) + \alpha v_i(x) + w_i(x) \right)}{\gamma} \cdot \gamma + C \cdot \delta.$$

Accept the proof if the test passes.

$\boldsymbol{\pi} \leftarrow \mathsf{Sim}(R, \boldsymbol{\tau}, a_1, \ldots, a_\ell)$**:** Pick $A, B \leftarrow \mathbb{F}$ and compute $C = \frac{AB - \alpha\beta - \sum_{i=0}^{\ell} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}$. Return $\boldsymbol{\pi} = (A, B, C)$.

Before formally proving this is a NILP, let us give a little intuition behind the different components. The role of α and β is to ensure A, B and C are consistent with each other in the choice of $a_0, \ldots, a_m$. The product $\alpha \cdot \beta$ in the verification equation guarantees that A and B involve non-trivial α and β components. This means the product $A \cdot B$ involves a linear dependence on α and β, and we will later prove that this linear dependence can only be balanced out by C with a consistent choice of $a_0, \ldots, a_m$ in all three of A, B and C. The role of γ and δ is to make the two latter products of the verification equation independent from the first product, by dividing the left factors with γ and δ respectively. This prevents mixing and matching of elements intended for different products in the verification equation. Finally, we use r and s to randomize the proof to get zero-knowledge.

Theorem 1. *The construction above yields a NILP with perfect completeness, perfect zero-knowledge and statistical knowledge soundness against affine prover strategies.*

Proof. Perfect completeness is straightforward to verify. Perfect zero-knowledge follows from both real proofs and simulated proofs having uniformly random field elements A, B. These elements uniquely determine C through the verification equation, so real proofs and simulated proofs have identical probability distributions.

What remains is to demonstrate that for any affine prover strategy with non-negligible success probability we can extract a witness. When using an affine prover strategy we have

$$A = A_\alpha \alpha + A_\beta \beta + A_\gamma \gamma + A_\delta \delta + A(x) + \sum_{i=0}^{\ell} A_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} + \sum_{i=\ell+1}^{m} A_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + A_h(x) \frac{t(x)}{\delta},$$

for known field elements $A_\alpha, A_\beta, A_\gamma, A_\delta, A_i$ and polynomials $A(x), A_h(x)$ of degrees $n-1$ and $n-2$, respectively that correspond to the first row of the matrix Π. We can write out B and C in a similar fashion from the second and third rows of Π.

We now view the verification equation as an equality of multi-variate Laurent polynomials. By the Schwartz-Zippel lemma the prover has negligible success probability unless the verification equation holds when viewing A, B and C as formal polynomials in indeterminates $\alpha, \beta, \gamma, \delta, x$.

The terms with indeterminate α^2 are $A_\alpha B_\alpha \alpha^2 = 0$, which means $A_\alpha = 0$ or $B_\alpha = 0$. Since $AB = BA$ we can without loss of generality assume $B_\alpha = 0$. The terms with indeterminate $\alpha\beta$ give us $A_\alpha B_\beta + A_\beta B_\alpha = A_\alpha B_\beta = 1$. This means $AB = (AB_\beta)(A_\alpha B)$ so we can without loss of generality after rescaling assume $A_\alpha = B_\beta = 1$. The terms with indeterminate β^2 now give us $A_\beta B_\beta = A_\beta = 0$. We have now simplified A and B constructed by the adversary to be of the form

$$A = \alpha + A_\gamma \gamma + A_\delta \delta + A(x) + \cdots \qquad B = \beta + B_\gamma \gamma + B_\delta \delta + B(x) + \cdots .$$

Next, let us consider the terms involving $\frac{1}{\delta^2}$. We have

$$\left(\sum_{i=\ell+1}^{m} A_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)+A_h(x)t(x)\right)$$
$$\cdot\left(\sum_{i=\ell+1}^{m} B_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)+B_h(x)t(x)\right)=0,$$

showing either the left factor is 0 or the right factor is 0. By symmetry, let us without loss of generality assume $\sum_{i=\ell+1}^{m} A_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)+t(x)A_t(x)=0$. The terms in $\alpha\frac{\sum_{i=\ell+1}^{m} B_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+B_h(x)t(x)}{\delta}=0$ now show us that also $\sum_{i=\ell+1}^{m} B_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)+$ $B_h(x)t(x)=0$.

The terms involving $\frac{1}{\gamma^2}$ give us

$$\sum_{i=0}^{\ell} A_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)\cdot\sum_{i=0}^{\ell} B_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)=0,$$

showing either the left factor is 0 or the right factor is 0. By symmetry, let us without loss of generality assume $\sum_{i=0}^{\ell} A_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)=0$. The terms in $\alpha\frac{\sum_{i=0}^{m} B_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))}{\gamma}=0$ now show us $\sum_{i=0}^{\ell} B_i\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)=0$ as well.

The terms $A_\gamma\beta\gamma=0$ and $B_\gamma\alpha\gamma=0$ show us that $A_\gamma=0$ and $B_\gamma=0$. We now have

$$A=\alpha+A(x)+A_\delta\delta \qquad\qquad B=\beta+B(x)+B_\delta\delta.$$

The remaining terms in the verification equation that involve α give us $\alpha B(x)=\sum_{i=0}^{\ell} a_i\alpha v_i(x)+\sum_{i=\ell+1}^{m} C_i\alpha v_i(x)$. The terms involving β give us $\beta A(x)=\sum_{i=0}^{\ell} a_i\beta u_i(x)+\sum_{i=\ell+1}^{m} C_i\beta u_i(x)$. Defining $a_i=C_i$ for $i=\ell+1,\ldots,m$ we now have

$$A(x)=\sum_{i=0}^{m} a_i u_i(x) \qquad\qquad B(x)=\sum_{i=0}^{m} a_i v_i(x).$$

Finally, we look at the terms involving powers of x to get

$$\sum_{i=0}^{m} a_i u_i(x)\cdot\sum_{i=0}^{m} a_i v_i(x)=\sum_{i=0}^{m} a_i w_i(x)+C_h(x)t(x).$$

This shows that $(a_{\ell+1},\ldots,a_m)=(C_{\ell+1},\ldots,C_m)$ is a witness for the statement $(a_1,\ldots,a_\ell)$.

$\square$

2 FIELD ELEMENT NILPS. It is natural to ask whether the number of field elements the prover sends in the NILP can be reduced further. The square span programs of Danezis et al. [DFGK14] give rise to 2 field element NILPs for boolean circuit satisfiability. It is also possible to get a 2-element NILP for arithmetic circuit satisfiability by rewriting the circuit into one that only uses squaring gates, since each multiplication gate $a\cdot b=c$ can be rewritten as a $(a+b)^2-(a-b)^2=4c$. When an arithmetic circuit only has squaring

gates we get $u_i(x) = v_i(x)$ for all i. By choosing $r = s$ in the NILP, we now have that $B = A + \beta - \alpha$, so the prover only needs to send two elements A and C to make a convincing proof. Rewriting the arithmetic circuit to only use squaring gates may double the number of gates and also requires some additional wires for the subtraction of the squares, so the reduction of the size of the NILP comes at a significant computational cost though.

3.2 NIZK arguments for quadratic arithmetic programs

We will now give a pairing-based NIZK argument for quadratic arithmetic programs. We consider relation generators $\mathcal{R}$ that return relations of the form

$$R = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \ell, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X)),$$

with $|p| = \lambda$. The relation defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$ such that with $a_0 = 1$

$$\sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) = \sum_{i=0}^m a_i w_i(X) + h(X)t(X),$$

for some degree $n - 2$ quotient polynomial $h(X)$.

An important design feature of the NILP we gave above is that it is easily to make it a split NILP. The proof elements A, B and C are only used once in the verification equation and therefore it is easy to assign them to different sides of the bilinear test. By splitting the common reference string in two parts that enable the computation of each side of the proof we then get a split NILP. The resulting split NILP is also disclosure-free and can therefore be compiled into a NIZK argument in the generic group model as we did in Section 2.5. Since pairing-friendly elliptic curves usually have that the group element representations are smaller in $\mathbb{G}_1$ than in $\mathbb{G}_2$ [GPS08] we choose to assign A and C to the first source group and B to the second source group for maximal efficiency. This gives us the following NIZK argument.

$(\sigma, \tau) \leftarrow \mathsf{Setup}(R)$**:** Pick $\alpha, \beta, \gamma, \delta, x \leftarrow \mathbb{Z}_p^*$. Define $\tau = (\alpha, \beta, \gamma, \delta, x)$ and compute $\sigma = ([\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2)$, where

$$\boldsymbol{\sigma}_1 = \begin{pmatrix} \alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i=0}^{\ell} \\ \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i=\ell+1}^{m}, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2} \end{pmatrix} \qquad \boldsymbol{\sigma}_2 = \left(\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1}\right).$$

$\pi \leftarrow \mathsf{Prove}(R, \sigma, a_1, \ldots, a_m)$**:** Pick $r, s \leftarrow \mathbb{Z}_p$ and compute $\pi = ([A]_1, [C]_1, [B]_2)$, where

$$A = \alpha + \sum_{i=0}^m a_i u_i(x) + r\delta \qquad B = \beta + \sum_{i=0}^m a_i v_i(x) + s\delta$$

$$C = \frac{\sum_{i=\ell+1}^m a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + As + Br - rs\delta.$$

$0/1 \leftarrow \mathsf{Vfy}(R, \sigma, a_1, \ldots, a_\ell, \pi)$**:** Parse $\pi = ([A]_1, [C]_1, [B]_2) \in \mathbb{G}_1^2 \times \mathbb{G}_2$. Accept the proof if and only if

$$[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{\ell} a_i \left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2.$$

$\pi \leftarrow \mathsf{Sim}(R, \tau, a_1, \ldots, a_\ell)$**:** Pick $A, B \leftarrow \mathbb{Z}_p$ and compute a simulated proof $\pi = ([A]_1, [C]_1, [B]_2)$ with

$$C = \frac{AB - \alpha\beta - \sum_{i=0}^{\ell} a_i \left(\beta u_i(x) + \alpha v_i(x) + w_i(x)\right)}{\delta}.$$

Theorem 2. *The protocol given above is a non-interactive zero-knowledge argument with perfect completeness and perfect zero-knowledge. It has statistical knowledge soundness against adversaries that only use a polynomial number of generic bilinear group operations.*

Proof. It is easy to see that the non-interactive argument encodes a split NILP. The only thing that remains in order to apply Lemma 1 is to prove that the common reference string is disclosure-free. We observe that the common reference strings $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ contain multi-variate Laurent polynomials evaluated in elements in $\mathbb{Z}_p^*$. A test of the form $\boldsymbol{\sigma}_1 \cdot T\boldsymbol{\sigma}_2$ can evaluate to 0 because the corresponding formal multi-variate Laurent polynomial is zero, or because it is a non-zero Laurent polynomial that happens to evaluate to 0 in the concrete choice of input variables. It follows from a straightforward extension of the Schwartz-Zippel lemma that the latter case only occurs with negligible probability since the negative and poitive total degrees are polynomially bounded in λ. The remaining possibility is that the test corresponds to the zero polynomial formally speaking, but in that case any other common reference string $\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2$ would also have $\boldsymbol{\sigma}'_1 \cdot T\boldsymbol{\sigma}'_2 = 0$. □

Symmetric bilinear groups. The non-interactive argument system also works in symmetric bilinear groups where $\mathbb{G}_1 = \mathbb{G}_2$ and $g = h$. In this case, the common reference string contains the union of the elements in $[\boldsymbol{\sigma}_1]_1$ and $[\boldsymbol{\sigma}_2]_2$ and the proof and verification equation is computed the same way as described above.

Efficiency. The proof size is 2 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$. The common reference string contains a description of the relation R, n elements in $\mathbb{Z}_p$, $m + 2n + 3$ elements in $\mathbb{G}_1$, and $n + 3$ elements in $\mathbb{G}_2$.

The verifier does not need to know the entire common reference string, it suffices to know

$$\sigma_V = \left(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [1]_1, \left\{\left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right]_1\right\}_{i=0}^{\ell}, [1]_2, [\gamma]_2, [\delta]_2, [\alpha\beta]_T\right).$$

The verifier's reference string only contains $\ell + 2$ elements in $\mathbb{G}_1$, 3 elements in $\mathbb{G}_2$, and 1 element in $\mathbb{G}_T$.

The verification consists of checking that the proof consists of three appropriate group elements and checking a single pairing product equation. The verifier computes ℓ

exponentiations in $\mathbb{G}_1$, a small number of group multiplications, and 3 pairings (assuming $[\alpha\beta]_T = [\alpha]_1 \cdot [\beta]_2$ is precomputed in the verifier's reference string).

The prover has to compute the polynomial $h(X)$. The prover can compute the polynomial evaluations

$$\sum_{i=0}^{m} a_i u_i(r_q) = \sum_{i=0}^{m} a_i u_{i,q} \qquad \sum_{i=0}^{m} a_i v_i(r_q) = \sum_{i=0}^{m} a_i v_{i,q} \qquad \sum_{i=0}^{m} a_i w_i(r_q) = \sum_{i=0}^{m} a_i w_{i,q}$$

for $q = 1, \dots, n$. It depends on the relation how long time this computation takes; if it arises from an arithmetic circuit where each multiplication gate connects to a constant number of wires, the relation will be sparse and the computation will be linear in n. Since the polynomials have degree $n-1$ they are completely determined by these evaluation points. If $r_1, \dots, r_n$ are roots of unity for a suitable prime p she can compute $h(X)$ using standard Fast Fourier Transform techniques in $O(n \log n)$ operations in $\mathbb{Z}_p$. The prover can also compute the coefficients of $\sum_{i=0}^{m} a_i u_i(X)$ and $\sum_{i=0}^{m} a_i v_i(X)$ using FFT techniques. Having all the coefficients, the prover does $m + 3n - \ell + 3$ exponentiations in $\mathbb{G}_1$ and $n+1$ exponentiations in $\mathbb{G}_2$.

Asymptotically the exponentiations are the dominant cost as the security parameter grows. However, in practice the multiplications that go into the FFT computations may be more costly for moderate security parameters and large statements. In that case, it may be worth to use a larger common reference string that contains precomputed $[u_i(x)]_1, [v_i(x)]_1, [v_i(x)]_2$ elements for $i = 0, \dots, m$ such that A and B can be constructed directly instead of the prover having to compute the coefficients of $\sum_{i=0}^{m} a_i u_i(X)$ and $\sum_{i=0}^{m} a_i v_i(X)$ and then do the exponentiations. In the case of boolean circuits we have $a_i \in \{0, 1\}$ and the prover can with such precomputed elements just do m group multiplications for each when computing A and B. We have for this reason let the CRS be longer in Table 1 to get a low computational cost for the prover.[2]

4 Lower bounds for non-interactive arguments

It is an intriguing question how efficient non-interactive arguments can be. We will now give a lower bound showing that pairing-based non-interactive arguments must have at least 2 group elements in the proofs. More precisely, we look at pairing-based arguments where the common reference string contains a description of a bilinear group and a number of group elements, the proof consists of a number of group elements computed by the prover using generic group operations, and the verifier checks the proof using generic bilinear group operations. We will show that for such pairing-based argument systems, the proof needs to have elements from both $\mathbb{G}_1$ and $\mathbb{G}_2$ if the language includes hard decisional problems as defined below.

Consider sampleable problems for a relation R, where there are two sampling algorithms Yes and No. Yes samples statements and witnesses in the relation. No samples

[2] Since the modified common reference string that gives faster prover computation can be computed from the original common reference string, the security proof still applies and we get knowledge soundness against generic adversaries. We note that if the non-interactive argument has knowledge soundness in the *standard model* then the modified common reference string also gives knowledge soundness in the standard model *assuming we still give the original common reference string to the extractor.*

statements outside the language L_R defined by the relation. We are interested in relations where it is hard to tell whether a statement ϕ has been sampled by Yes or No.

Definition 5. *We say the relation generator $\mathcal{R}$ has hard decisional problems if there are two polynomial time algorithms* Yes *and* No *such that for* $(R, z) \leftarrow \mathcal{R}(1^\lambda)$ *we have* $\mathsf{Yes}(R) \rightarrow (\phi, w) \in R$ *and* $\mathsf{No}(R) \rightarrow \phi \notin L_R$ *with overwhelming probability, and for all non-uniform polynomial time distinguishers* $\mathcal{A}$

$$\Pr\left[(R, z) \leftarrow \mathcal{R}(1^\lambda); \phi_0 \leftarrow \mathsf{No}(R); (\phi_1, w_1) \leftarrow \mathsf{Yes}(R); b \leftarrow \{0,1\} : \mathcal{A}(R, z, \phi_b) = b\right] \approx \frac{1}{2}.$$

If one-way functions exist, we can construct pseudorandom generators. A pseudorandom generator can be used to generate a pseudorandom string, a Yes-instance, with the seed being the witness. To get a No-instance we sample a uniform random string, which with overwhelming probability is not pseudorandom. If the relation R is NP-complete, or just expressive enough to capture pseudorandom generators, then it has a hard decisional problem. In particular, when we are working with pairing-based arguments we must assume at the very least that the discrete logarithm problem is hard and then relation generators with hard decisional problems exist.

4.1 Linear interactive proofs cannot have linear decision procedures

We will now prove that NILPs cannot have a degree 1 verifier. This answers an open question raised by Bitansky et al. [BCI+13]. The result holds even if we consider designated verifier NILPs that get input $\boldsymbol{\sigma}_V$ not available to the prover, and instead of knowledge soundness we only consider the weaker notion of soundness that we now define.

Definition 6 (Statistical soundness against affine prover strategies). *We say a LIP is sound against affine prover strategies if for all adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{c}(R, z) \leftarrow \mathcal{R}(1^\lambda); (\boldsymbol{\sigma}_P, \boldsymbol{\sigma}_V, \boldsymbol{\tau}) \leftarrow \mathsf{Setup}(R); (\phi, \mathit{\Pi}) \leftarrow \mathcal{A}(R, z) \\ \boldsymbol{\pi} = \mathit{\Pi}\boldsymbol{\sigma}_P; \boldsymbol{t} \leftarrow \mathsf{Test}(R, \phi, \sigma_V) : \phi \notin L_R \ \wedge \ \boldsymbol{t}(\boldsymbol{\pi}) = \boldsymbol{0}\end{array}\right] \approx 0.$$

Theorem 3. *There are no NILPs with verifier degree 1 for relation generators with hard decisional problems.*

Proof. NILPs of degree 1 by definition have a decision procedure producing an arithmetic circuit $\boldsymbol{t} : \mathbb{F}^k \rightarrow \mathbb{F}^\eta$ evaluating polynomials of degree 1 and testing whether $\boldsymbol{t}(\boldsymbol{\pi}) = \boldsymbol{0}$. Since the polynomials have degree 1 it is possible to efficiently compute a matrix $A \in \mathbb{F}^{\eta \times k}$ and a vector $\boldsymbol{b} \in \mathbb{F}^\eta$ such that the test corresponds to checking $A\boldsymbol{\pi} = \boldsymbol{b}$.

Let us now construct an algorithm $\mathcal{A}$ that given R and ϕ has a good chance of determining whether $\phi \in L_R$ or $\phi \notin L_R$. It is crucial here that in our definition of NILPs the prover and soundness adversary choose the proof matrix $\mathit{\Pi}$ independently of the setup $\boldsymbol{\sigma}_P$ and $\boldsymbol{\sigma}_V$. The idea is to run an honest verifier repeatedly to create many common reference strings and verifications. When $\phi \in L_R$ the same proof matrix $\mathit{\Pi}$ will give valid proofs for all honest runs of the verifier. On the other hand, when $\phi \notin L_R$ soundness makes it unlikely that the same $\mathit{\Pi}$ can pass many tests.

We now give the details. First, $\mathcal{A}(R, \phi)$ runs the setup $N = mk \log |F|$ times to get $(\boldsymbol{\sigma}_{1,P}, \boldsymbol{\sigma}_{1,V}), \ldots, (\boldsymbol{\sigma}_{N,P}, \boldsymbol{\sigma}_{N,V})$. Then for each of them it creates the verifier's tests $A_i \in \mathbb{F}^{\eta \times k}$ and $\boldsymbol{b}_i \in \mathbb{F}^{\eta}$ for the statement ϕ. By completeness, if we had a witness for ϕ we could compute a proof matrix Π such that for all N tests $A_i \Pi \boldsymbol{\sigma}_{i,P} = \boldsymbol{b}_i$. The algorithm $\mathcal{A}$ does not know a witness for $\phi \in L_R$, but it can solve the set of linear equations to see whether such a Π exists. If it does, it outputs 1 to indicate $\phi \in L_R$ and otherwise it outputs 0 to indicate $\phi \notin L_R$.

Let us now analyze the success probability of the decision algorithm $\mathcal{A}$. On input $\phi \in L_R$ the completeness of the NILP means that such a Π exists and since the system of equations is linear it can be efficiently solved. The decision algorithm will therefore output 1 when $\phi \in L_R$. On input $\phi \notin L_R$ the soundness of the NILP means that any choice of Π has low probability of passing the verification. The chance that it passes all $N = mk \log |F|$ verifications is therefore upper bounded by $\text{negl}(\lambda)^{mk \log |F|}$. There are $|F|^{mk}$ possible choices of Π, so there is negligible probability that any Π exists that will pass all tests. The decision algorithm therefore outputs 0 with overwhelming probability when $\phi \notin L_R$. □

The proof of the Theorem 3 also holds for split NILPs and if we restrict the soundness adversary to produce split matrices Π_1, Π_2; this is just an extra restriction on the proof matrices the prover and adversary can produce. In Sect. 2.5 we constructed a pairing-based SNARK from disclosure-free split NILPs. The disclosure-freeness captured that a generic soundness adversary cannot learn interesting information about the common reference string in a pairing setting. All it can do is therefore to choose a statement $\phi \notin L_R$ and a proof matrix Π independently of $\boldsymbol{\sigma}$ and hope they pass the verification. Since a single element proof would correspond to a linear decision procedure in the Type III pairing setting, we get the following corollary:

Corollary 1. *A relation generator with pairing-based SNARKs in the Type III setting constructed from disclosure-free split NILPs as described in Sect. 2.5 must have at least two elements in the proofs for the languages to be non-trivial.*

4.2 Lower bound for the size of generic pairing-based non-interactive arguments

We will now generalize the statement that a pairing-based non-interactive argument over Type III groups must have elements in both $\mathbb{G}_1$ and $\mathbb{G}_2$ by not requiring the common reference string to be disclosure-free. The intuition behind the argument is still the same though: if we have a unilateral argument with only elements in $\mathbb{G}_1$ or only elements in $\mathbb{G}_2$, then the verification equations become linear and it becomes possible to violate soundness. For generality, we show this holds even if the common reference string and proof contain elements in $\mathbb{G}_T$. This results implies a lower bound of at least 2 group elements in a pairing-based non-interactive argument.

We will consider pairing-based argument systems $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy})$, where the common reference string and proofs consist of group elements computed using generic group operations and the verifier uses generic bilinear group operations to test the proof. We restrict the verifier to test the validity of the proof by creating a number of pairing product equations and accepting when they all holds. All known pairing-based SNARKs satisfy this restriction. The restriction rules out violations of the intention behind saying the argument is "pairing based". For instance, if the common reference string and the proof can be expressed as group elements raised to bits, $(G^0, G^1, \ldots, G^1, G^0)$, then we could imagine the prover would read of the bit-string in the common reference string, use this to create a non-pairing based SNARK, encode it as bits sent to the verifier, and the verifier would decode the bits in the proof and check it. Clearly this is just a very cumbersome way to encode a different type of SNARK and cannot be sadi to be pairing-based.

Let us be explicit about what we mean by a pairing-based non-interactive argument as described above and the consequences of using generic group operations.

$(\sigma, \tau) \leftarrow \mathsf{Setup}(R)$: The relation contains a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ and the common reference string contains group elements in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, i.e., $\sigma = ([\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2, [\boldsymbol{\sigma}_T]_T)$.
oup elements G, G^b, where b is a bit. The prover can easily recover the bit b by guessing it and verifying the guess with generic group operations.

$\pi \leftarrow \mathsf{Prove}(R, \sigma, \phi, w)$: The prover uses generic group operations to construct the proof. This means that she picks matrices Π_1, Π_2 and Π_T and computes the proof by setting[3]

$$\pi = (\Pi_1[\boldsymbol{\sigma}_1]_1, \Pi_2[\boldsymbol{\sigma}_2]_2, \Pi_T[\boldsymbol{\sigma}_T]_T).$$

Note that we do not assume the common reference string is disclosure-free, so it is possible the matrices Π_1, Π_2, Π_T are related to $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_T$.

$0/1 \leftarrow \mathsf{Vfy}(R, \sigma, \phi, \pi)$: The verifier works in two steps. First, it generates matrices and vectors $\{T_i, \boldsymbol{t}_i\}_{i=1}^{\eta}$. It chooses these matrices and vectors independently of the proof, but may use knowledge of the statement and the common reference string. Then it accepts if and only if all pairing product equations of the form

$$[\boldsymbol{\sigma}_1^\top, \boldsymbol{\pi}_1^\top]_1 \cdot T_i \begin{bmatrix} \boldsymbol{\sigma}_2 \\ \boldsymbol{\pi}_2 \end{bmatrix} = \boldsymbol{t} \cdot \begin{bmatrix} \boldsymbol{\sigma}_T \\ \boldsymbol{\pi}_T \end{bmatrix}_T$$

hold.

Theorem 4. *A pairing-based non-interactive argument with generic group algorithms as described above cannot exist for relation generators with hard decisional problems unless the proofs have elements both in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.

Proof. Let us for contradition assume we have a pairing-based non-interactive argument of the form described above where the proofs have no elements in $\mathbb{G}_1$. The case where the proof has no elements in $\mathbb{G}_2$ is similar. This means the proofs are of the form $[\Pi_2 \boldsymbol{\sigma}_2]_2, [\Pi_T \boldsymbol{\sigma}_T]_T$, where Π_2, Π_T are matrices chosen by the generic prover. The matrices and vectors for the test of a proof can then be rewritten as $(A_1, B_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \ldots, A_\eta, B_\eta, \boldsymbol{c}_\eta, \boldsymbol{d}_\eta)$, for which the verifier checks

$$[\boldsymbol{\sigma}_1]_1 \cdot A_i[\boldsymbol{\sigma}_2]_2 + [\boldsymbol{\sigma}_1]_1 \cdot B_i \Pi_2[\boldsymbol{\sigma}_2]_2 = \boldsymbol{c}_i \cdot [\boldsymbol{\sigma}_T]_T + \boldsymbol{d}_i \cdot \Pi_T[\boldsymbol{\sigma}_T]_T.$$

[3] The prover can also include pairings of elements in $\mathbb{G}_1$ and $\mathbb{G}_T$ in the proof but we can without loss of generality assume all possible pairwise pairings of elements in $[\boldsymbol{\sigma}_1]_1$ and $[\boldsymbol{\sigma}_2]_2$ are included in $[\boldsymbol{\sigma}_T]_T$.

We observe that the verification equations correspond to a system of linear equations in $\mathit{\Pi}_2$ and $\mathit{\Pi}_T$.

We will use such a pairing-based argument system to design an algorithm $\mathcal{A}(R, \phi)$ that gets a statement ϕ as input that is either generated as a Yes-instance or a No-instance and decides which case it is. The algorithm works in two stages: first it generates a lot of honest proofs for Yes-instances chosen by itself, and then it checks exloits the linearity of the verification equations to check if the current instance ϕ can have a proof similar to the other Yes-instances. If ϕ is a Yes-instance it can have such a structure, but if ϕ is a No-instance soundness says it cannot.

For the first stage, the algorithm samples many Yes-instances $(\phi_j, w_j) \leftarrow \mathsf{Yes}(R)$. It then generates a common reference string $[\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2, [\boldsymbol{\sigma}_T]_T$ and creates proof matrices $\mathit{\Pi}_{j,2}, \mathit{\Pi}_{j,T}$ and verification tests $(A_{j,1}, B_{j,1}, \boldsymbol{c}_{j,1}, \boldsymbol{d}_{j,1}, \ldots, A_{j,\eta}, B_{j,\eta}, \boldsymbol{c}_{j,\eta}, \boldsymbol{d}_{j,\eta})$ for all statements. Let V be the vector space generated by $(A_{j,1}, B_{j,1}\mathit{\Pi}_{j,2}, \boldsymbol{c}_{j,1}, \boldsymbol{d}_{j,1}^{\top}\mathit{\Pi}_T, \ldots, A_{j,\eta}, B_{j,\eta}\mathit{\Pi}_{j,2}, \boldsymbol{c}_{j,\eta}, \boldsymbol{d}_{j,\eta}^{\top}\mathit{\Pi}_{j,T}$

The algorithm keeps sampling until λ Yes-instances ϕ_j in a row give vectors already in V. The vector space has polynomial dimension, so this process terminates in polynomial time. Chernoff-bounds then tell us that with at least 50% probability the Yes-instance ϕ also gives rise to a vector in V. Of course, even if ϕ is a Yes-instance, the algorithm does not know the corresponding witness though.

The algorithm now proceeds to the second phase. Given ϕ it creates the test $(A_1, B_1, \boldsymbol{c}_1, \boldsymbol{d}_1, \ldots, A_\eta, B_\eta,$ Then it tries to solve for $\mathit{\Pi}_2, \mathit{\Pi}_T$ such that $(A_1, B_1\mathit{\Pi}_2, \boldsymbol{c}_1, \boldsymbol{d}_1^{\top}\mathit{\Pi}_T, \ldots, A_\eta, B_\eta\mathit{\Pi}_2, \boldsymbol{c}_\eta, \boldsymbol{d}_\eta^{\top}\mathit{\Pi}_T)$ belongs to the vector space V. This is a system of linear equations, so it can be efficiently solved. If the algorithm manages to solve for $\mathit{\Pi}_2, \mathit{\Pi}_T$ it returns 1, otherwise it returns 0.

Let us now analyze the success probability of the algorithm. If ϕ is sampled as a Yes-instance the algorithm has least 50% chance of finding $\mathit{\Pi}_2, \mathit{\Pi}_T$ giving rise to a vector in V. On the other hand, if ϕ is sampled as a No-instance, soundness implies that there is negligible probability of finding such a $\mathit{\Pi}_2, \mathit{\Pi}_T$. We note that soundness holds because the algorithm after generating the setup $[\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2, [\boldsymbol{\sigma}_T]_T$ honestly runs the generic prover and verifier several times, but never uses special knowledge about the underlying discrete logarithms $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_T$ except what an honest prover and verifier might learn with generic algorithms. $\square$

Corollary 2. *A pairing-based non-interactive argument with generic group algorithms as described must have at least two group elements in the proof.*

Acknowledgments

We are grateful to Alessandro Chiesa and Madars Virza for extensive comments on an earlier version of this paper and for their implementation and analysis of the SNARK in the libsnark library [CV16]. We also thank Eran Tromer and Michael Walfish for interesting discussions about the performance of SNARK implementations and the anonymous reviewers for their comments.

References

[AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 118–136, 2007.

[AGOT14] Masayuki Abe, Jens Groth, Miyako Ohkubo, and Mehdi Tibouchi. Unified, minimal and selectively randomizable structure-preserving signatures. In *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 688–712, 2014.

[BBFR15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *IEEE Symposium on Security and Privacy*, pages 271–286, 2015.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. Cryptology ePrint Archive, Report 2005/015, 2005.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108, 2013.

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

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333, 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, pages 505–514, 2014.

[BCTV14a] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294, 2014.

[BCTV14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX*, pages 781–796, 2014.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112, 1988.

[BFR$^+$13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *SOSP*, pages 341–357, 2013.

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In *ASIACRYPT*, volume 9453 of *Lecture Notes in Computer Science*, pages 236–261, 2015.

[BSCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Shaul Kfir, Eran Tromer, and Madars Virza. libsnark, 2014. Available at https://github.com/scipr-lab/libsnark.

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

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *EUROCRYPT*, volume 9057 of *Lecture Notes in Computer Science*, pages 371–403, 2015.

[CV16] Alessandro Chiesa and Madars Virza. Personal communication, 2016.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In *ASIACRYPT*, volume 8873 of *Lecture Notes in Computer Science*, pages 532–550, 2014.

[DFKP13] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *PETShopCCS*, 2013.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, 2013.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proofs. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 97–111, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM*, 59(3):11:1–11:35, 2012.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, volume 4248 of *Lecture Notes in Computer Science*, pages 444–459, 2006.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 192–208, 2009.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, 2010.

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM Journal on Computing*, 41(5):1193–1232, 2012.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC*, pages 723–732, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324, 1995.

[KPP$^+$14] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: faster verifiable set computations. In *USENIX*, pages 765–780, 2014.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189, 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *ASIACRYPT*, volume 8269 of *Lecture Notes in Computer Science*, pages 41–60, 2013.

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

[Nec94] Vasilii I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mat. Zametki*, 55(2):91–101, 1994.

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

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, 1997.

[SVdV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-friendly outsourcing by distributed verifiable computation. In *ACNS*, volume ???? of *Lecture Notes in Computer Science*, pages ???–???, 2016.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, 2008.

[Wal15] Michael Walfish. A wishlist for verifiable computation: An applied cs perspective, 2015.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

[WSR$^+$15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*, 2015.