Square Span Programs with Applications to Succinct NIZK Arguments

George Danezis[1], Cédric Fournet[2], Jens Groth[1] *, and Markulf Kohlweiss[2]

[1] University College London, UK
[2] Microsoft Research

Abstract. We propose a new characterization of NP using square span programs (SSPs). We first characterize NP as affine map constraints on small vectors. We then relate this characterization to SSPs, which are similar but simpler than Quadratic Span Programs (QSPs) and Quadratic Arithmetic Programs (QAPs) since they use a single series of polynomials rather than 2 or 3.
We use SSPs to construct succinct non-interactive zero-knowledge arguments of knowledge. For performance, our proof system is defined over Type III bilinear groups; proofs consist of just 4 group elements, verified in just 6 pairings. Concretely, using the Pinocchio libraries, we estimate that proofs will consist of 160 bytes verified in less than 6 ms.

Keywords: Square span program, quadratic span program, SNARKs, non-interactive zero-knowledge arguments of knowledge.

1 Introduction

Gennaro, Gentry, Parno and Raykova [GGPR13] proposed a new, influential characterization of the complexity class **NP** using Quadratic Span Programs (QSPs), a natural extension of span programs defined by Karchmer and Wigderson [KW93]. Their main motivation was the construction of Succinct Non-interactive Arguments of Knowledge (SNARKs). Their work has lead to fast progress towards practical verifiable computations, whereby a resource-constrained client offloads the computation of an expensive function to a computationally endowed server or cloud, but still intends to verify the correctness of any returned results. For instance, using Quadratic Arithmetic Programs (QAPs), a generalization of QSPs for arithmetic circuits, Pinocchio [PHGR13] provides evidence that verified remote computation can be faster than local computation. At the same time, zero-knowledge variants of their construction enable the server to keep intermediate and additional values used in the computation private. Such constructions are at the forefront of privacy-friendly variants of Bitcoin, such as Pinocchio Coin [DFKP13] and Zerocash [BSCG+14].

* 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.

We introduce Square Span Programs (SSPs), a radical simplification of quadratic span programs, and we use them to build simpler and more efficient SNARKs and Non-Interactive Zero-Knowledge arguments (NIZKs) for the verified computation of binary circuits and the verification of SAT solving, two closely related problems. Thus, SSPs can be used to build NIZK arguments to support privacy properties while guaranteeing high integrity, at a minimal cost for the verifier.

Square span programs are based on the insight that every 2-input binary gate $g(a, b) = c$ can be specified using (1) an affine combination $\ell = \alpha a + \beta b + \gamma c + \delta$ of the gate's input and output wires that take exactly two values, $\ell = 0$ or $\ell = 2$, when the wires meet the gate's logical specification; and (2), equivalently, as a single 'square' constraint $(\ell - 1)^2 = 1$. Composing such constraints, a satisfying assignment for any binary circuit (or any SAT problem) can be specified first as a set of affine map constraints, then as a constraint on the span of a set of polynomials, defining the square span program for this circuit.

Due to their conceptual simplicity, SSPs offer several advantages over previous constructions for binary circuits. Their reduced number of constraints lead to smaller programs, and to lower sizes and degrees for the polynomials required to represent them, which in turn reduce the computation complexity required in proving or verifying NIZK arguments. Notably, their simpler 'square' form requires only a single polynomial to be evaluated for verification (instead of two for earlier QSPs, and three for Pinocchio [PHGR13]) leading to a simpler and more compact setup, smaller keys, and fewer operations required for proof and verification.

The resulting, SSP-based SNARKs may be the most compact constructions to date. For performance, our proof system is defined over Type III bilinear groups; to this end, we revisit and restate known assumptions for Type III bilinear groups. The communicated proofs consist of just 4 group elements (3 in the left group, and one in the right group); they can be verified in just 6 pairings, plus one multiplication for each (non-zero) bit of input, irrespective of the size of the circuit. Concretely, using the same groups as in the implementation of Pinocchio, we arrive at 160-byte proofs that we estimate can be verified in less than 6 ms, for circuits with millions of gates. For instance, our SNARKs would be entirely adequate to verify the solutions of large SAT problems offloaded to specialized servers and tools, such as those available in the annual SAT competition[3], without the need to communicate (or even reveal) their complete solutions.

2 Square span programs

In this section we will provide new characterizations of languages in NP. First, we show that circuit satisfiability can be recast as a set of constraints on affine maps over the integers. Next, we show in Section 2.2 that this leads to the NP-completeness of square span programs as defined below. The reader may find the example in Section 2.3 useful to illustrate the transformation from circuit

[3] http://satcompetition.org/

satisfiability to square span programs. We compare square span programs to quadratic span programs in Section 2.4.

Definition 1 (Square span program). *A square span program Q over the field $\mathbb{F}$ consists of $m+1$ polynomials $v_0(x), v_1(x), \ldots, v_m(x)$ and a target polynomial $t(x)$ such that $\deg(v_i(x)) \leq \deg(t(x))$ for all $i = 0, \ldots, m$.*

We say that the square span program Q has size m and degree $d = \deg(t(x))$.

We say that Q accepts an input $(a_1, \ldots, a_\ell) \in \mathbb{F}^\ell$ if and only if there exist $a_{\ell+1}, \ldots, a_m \in \mathbb{F}$ satisfying

$$t(x) \quad \textit{divides} \quad \left(v_0(x) + \sum_{i=1}^{m} a_i v_i(x)\right)^2 - 1.$$

We say that Q verifies a boolean function $f : \{0,1\}^\ell \to \{0,1\}$ if it accepts exactly those inputs $\boldsymbol{a} \in \mathbb{F}^\ell$ that satisfy $\boldsymbol{a} \in \{0,1\}^\ell$ and $f(\boldsymbol{a}) = 1$.

In the definition, we may see f as a binary circuit or, more abstractly, as a logical specification of a satisfiability problem. In our NIZK argument system in Section 3.3 we will split the ℓ inputs into ℓ_u public and ℓ_w private inputs. We remark that the public 'inputs' are considered from the viewpoint of the verifier: for an outsourced computation for instance, they may include both the inputs sent by the clients and the outputs returned by the server performing the computation together with its proof; for a SAT problem, they may provide a partial instantiation of the problem, or a part of its solution.

This treatment is strictly more general than classic Circuit-SAT which only cares about satisfiability and thus corresponds to the special case of $\ell_u = 0$, i.e., Q verifies a circuit C if it accepts exactly those w where $C(w) = 1$. Alternatively, if we want the same SSP Q to handle different circuits, it may be useful to let f be a universal circuit that takes as input an ℓ_u-bit description of a freely chosen circuit C and an ℓ_w-bit value w and returns 1 if and only if $C(w) = 1$.

2.1 The NP-completeness of affine map constraints

In this section we will show that circuit satisfiability can be recast as a set of constraints on the image of an affine map $\boldsymbol{a} \mapsto \boldsymbol{a}V + \boldsymbol{b}$.

Groth, Ostrovsky and Sahai [GOS12] used that a NAND-gate with input wires a, b and output wire c can be "linearized". Given values $a, b, c \in \{0,1\}$, with 0 meaning false and 1 meaning true, and writing $\bar{c}$ for $1 - c$, we have

$$c = \neg(a \wedge b) \quad \text{if and only if} \quad a + b - 2\bar{c} \in \{0,1\}.$$

All logic gates with fan-in 2 can be linearized. We will without loss of generality ignore gates corresponding to $c = a$, $c = \overline{a}$, $c = b$, $c = \overline{b}$, $c = 0$ and $c = 1$ since they are trivial and can be eliminated from a circuit. This leaves us with 10 types of logic gates. Table 1 displays their truth tables and their linearizations.

Let C be a circuit with m wires and n fan-in 2 gates. We can use linearization of the logic gates to rewrite the circuit as a set of constraints on the output of an affine map.

AND

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

$a + b - 2c \in \{0, 1\}$

OR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

$\bar{a} + \bar{b} - 2\bar{c} \in \{0, 1\}$

XOR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

$a + b + c \in \{0, 2\}$

NAND

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

$a + b - 2\bar{c} \in \{0, 1\}$

NOR

a	b	c
0	0	1
0	1	0
1	0	0
1	1	0

$\bar{a} + \bar{b} - 2c \in \{0, 1\}$

XNOR

a	b	c
0	0	1
0	1	0
1	0	0
1	1	1

$a + b + \bar{c} \in \{0, 2\}$

$\bar{a} \wedge b$

a	b	c
0	0	0
0	1	1
1	0	0
1	1	0

$\bar{a} + b - 2c \in \{0, 1\}$

$\overline{\bar{a} \wedge b}$

a	b	c
0	0	1
0	1	0
1	0	1
1	1	1

$\bar{a} + b - 2\bar{c} \in \{0, 1\}$

$a \wedge \bar{b}$

a	b	c
0	0	0
0	1	0
1	0	1
1	1	0

$a + \bar{b} - 2c \in \{0, 1\}$

$\overline{a \wedge \bar{b}}$

a	b	c
0	0	1
0	1	1
1	0	0
1	1	1

$a + \bar{b} - 2\bar{c} \in \{0, 1\}$

Table 1. Linearization of logic gates with inputs a, b and output c. We omit the 6 remaining gates, which depend on at most one input and are not used in circuits.

Theorem 1. *For any circuit C with m wires and n fan-in 2 gates for a total size of $d = m + n$, there exists a matrix $V \in \mathbb{Z}^{m \times d}$ and a vector $\boldsymbol{b} \in \mathbb{Z}^d$ such that C is satisfiable if and only if there is a vector $\boldsymbol{a} \in \mathbb{Z}^m$ satisfying $\boldsymbol{a}V + \boldsymbol{b} \in \{0, 2\}^d$.*

The matrix V and the vector $\boldsymbol{b}$ can be constructed such that $\boldsymbol{a}V + \boldsymbol{b} \in \{0, 2\}^d$ implies $\boldsymbol{a} \in \{0, 1\}^m$ and $a_1, \ldots, a_m$ corresponds to the values on the wires in a satisfying assignment for C with the first ℓ bits being the input wires.

Proof. We represent an assignment to the wires as a vector $\boldsymbol{a} \in \mathbb{Z}^m$. The assignment is a satisfying witness for the circuit if and only if the entries belong to $\{0, 1\}$, the entries respect all gates, and the output wire is 1.

It is easy to impose the condition $\boldsymbol{a} \in \{0, 1\}^m$ by requiring $\boldsymbol{a}(2I) \in \{0, 2\}^m$. (Alternatively, whenever $a_i \in \{0, 1\}$ is clear from the context, for instance for the public inputs $a_1, \ldots, a_{\ell_u}$, this check can be omitted.)

Since $\bar{a} = 1 - a$, $\bar{b} = 1 - b$ and $\bar{c} = 1 - c$ and after scaling some of the gate equations from Table 1 by a factor 2, we can write all gate equations in the form $\alpha a + \beta b + \gamma c + \delta \in \{0, 2\}$. We want the circuit output wire c_{out} to have value 1. We do that by adding the condition $3\bar{c}_{\text{out}}$ to the linearization of the output gate, since if $c_{\text{out}} = 0$ this adds 3 to the linear equation and brings us outside $\{0, 2\}$ regardless of the type of logic gate.

Define $G \in \mathbb{Z}^{m \times n}$ and $\boldsymbol{\delta} \in \mathbb{Z}^n$ such that $\boldsymbol{a}G + \boldsymbol{\delta} \in \{0,2\}^n$ corresponds to the linearization of the gates as described above, and let

$$V = [\ 2I \mid G\ ] \quad \text{and} \quad \boldsymbol{b} = (\ \mathbf{0} \mid \boldsymbol{\delta}\ ).$$

The existence of $\boldsymbol{a}$ such that

$$\boldsymbol{a}V + \boldsymbol{b} \in \{0,2\}^d$$

is equivalent to a satisfying assignment to the wires in the circuit. □

Note that V and $\boldsymbol{b}$ as we constructed them have some additional properties. The matrix V is sparse, since it only has $m+3n$ non-zero entries. The row vectors of V and $\boldsymbol{b}$ are all linearly independent. Furthermore, all entries in V and $\boldsymbol{b}$ are small integers. The small size of the integers gives us the following corollary.

Corollary 1. *For any circuit C with m wires and n fan-in 2 gates and for any $p \geq 8$ there exist a matrix $V \in \mathbb{Z}_p^{m \times d}$ (with $d = m + n$) and a vector $\boldsymbol{b} \in \mathbb{Z}_p^d$ (giving us $m+1$ linearly independent row vectors) such that C is satisfiable if and only if there exists a vector $\boldsymbol{a} \in \mathbb{Z}_p^m$ satisfying $\boldsymbol{a}V + \boldsymbol{b} \in \{0,2\}^d$. Furthermore, if $\boldsymbol{a}V + \boldsymbol{b} \in \{0,2\}^d$ then $\boldsymbol{a} \in \{0,1\}^m$ and $C(a_1, \ldots, a_\ell) = 1$.*

Relation to closest vector problem. There is an interesting connection between our construction of affine map constraints and the closest vector problem for integer lattices using the max-norm ℓ_∞. Consider a circuit made just from NAND-gates; then the affine map $\boldsymbol{a}V + \boldsymbol{b}$ constructed in the proof of Theorem 1 cannot take the value 1 for any index $i = 1, \ldots, d$, which means the circuit is satisfiable if and only if $\boldsymbol{a}V + \boldsymbol{b} - \mathbf{1} \in \{-1, 0, 1\}^d$. This is equivalent to saying that the lattice generated by the rows of V has a vector $\boldsymbol{a}V$ with distance at most 1 from the target vector $\boldsymbol{t} = \mathbf{1} - \boldsymbol{b}$, i.e., $||\boldsymbol{a}V - \boldsymbol{t}||_\infty \leq 1$, if and only if the circuit is satisfiable. Our construction therefore gives a very direct reduction of the closest vector problem in integer lattices to circuit satisfiability. The NP-hardness of the closest (nearest) vector problem was first demonstrated by van Emde Boas [vEB81] but using a more complicated reduction that relied on the partition problem.

2.2 The NP-completeness of square span programs

We will now connect affine maps to square span programs, which gives a reduction of square span programs to circuit satisfiability.

Corollary 1 can be reformulated to say that, for any circuit C and $p \geq 8$, there exist V and $\boldsymbol{b}$ such that C is satisfiable if and only there exists $\boldsymbol{a} \in \mathbb{Z}_p^m$ satisfying

$$(\boldsymbol{a}V + \boldsymbol{b}) \circ (\boldsymbol{a}V + \boldsymbol{b} - \mathbf{2}) = \mathbf{0},$$

where $\circ$ denotes the Hadamard product (entry-wise multiplication). We can rewrite this condition as

$$(\boldsymbol{a}V + \boldsymbol{b} - \mathbf{1}) \circ (\boldsymbol{a}V + \boldsymbol{b} - \mathbf{1}) = \mathbf{1}.$$

Let $r_1, \ldots, r_d$ be d distinct elements of $\mathbb{Z}_p$ for a prime $p \geq \max(d, 8)$. Define $v_0(x), v_1(x), \ldots, v_m(x)$ as the degree $d-1$ polynomials satisfying

$$v_0(r_j) = b_j - 1 \quad \text{and} \quad v_i(r_j) = V_{i,j}.$$

We can now reformulate Corollary 1 again. The circuit C is satisfiable if and only if there exists $\boldsymbol{a} \in \mathbb{Z}_p^m$ such that for all r_j

$$\left(v_0(r_j) + \sum_{i=1}^{m} a_i v_i(r_j)\right)^2 = 1.$$

Since the evaluations in $r_1, \ldots, r_d$ uniquely determine the polynomial $v(x) = v_0(x) + \sum_{i=1}^{m} a_i v_i(x)$ we can rewrite the condition as

$$\left(v_0(x) + \sum_{i=1}^{m} a_i v_i(x)\right)^2 \equiv 1 \mod \prod_{j=1}^{d}(x - r_j).$$

Theorem 2. *A circuit C with m wires and n fan-in 2 gates has for any prime $p \geq \max(n, 8)$ a square span program of size m and degree $d = m+n$ that verifies it over $\mathbb{Z}_p$.*

Proof. From the discussion above, we see that for any circuit C with m wires and n gates there exists polynomials $v_0(x), v_1(x), \ldots, v_m(x)$ and distinct roots $r_1, \ldots, r_d$ such that C is satisfiable if and only if

$$\prod_{j=1}^{d}(x - r_j) \quad \text{divides} \quad \left(v_0(x) + \sum_{i=1}^{m} a_i v_i(x)\right)^2 - 1.$$

Define $t(x) = \prod_{j=1}^{d}(x - r_j)$ to get an SSP $Q = \big(v_0(x), v_1(x), \ldots, v_m(x), t(x)\big)$ that verifies C over $\mathbb{Z}_p$. □

2.3 Example

As a small example of the process of generating a square span program, consider a circuit consisting of a single XOR-gate $a_3 = a_1 \oplus a_2$ (here $\ell = \ell_u + \ell_w = 2$ with $\ell_u = 0$ and $\ell_w = 2$). To guarantee $a_1, a_2, a_3 \in \{0, 1\}$ and the XOR-gate is respected we use the constraints $2a_i \in \{0, 2\}$ and $a_1 + a_2 + a_3 \in \{0, 2\}$. The output should be $a_3 = 1$, which we represent with the constraint $3\bar{a}_3 = 3(1 - a_3) = 0$. We add the latter constraint to the output wire's equation to get the combined $a_1 + a_2 - 2a_3 + 3 \in \{0, 2\}$, which at the same time guarantees $a_3 = a_1 \oplus a_2$ and $a_3 = 1$. We can represent the constraints as

$$\boldsymbol{a}V + \boldsymbol{b} = (a_1, a_2, a_3) \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix} + (0, 0, 0, 3) \in \{0, 2\}^4.$$

The satisfiability of the circuit can therefore be represented by 4 quadratic equations

$$(2a_1 - 1)^2 = 1 \quad (2a_2 - 1)^2 = 1 \quad (2a_3 - 1)^2 = 1 \quad (a_1 + a_2 - 2a_3 + 2)^2 = 1$$

corresponding to $(\boldsymbol{a}V + \boldsymbol{b} - \mathbf{1}) \circ (\boldsymbol{a}V + \boldsymbol{b} - \mathbf{1}) = \mathbf{1}$.

To get a square span program, let $p \geq 8$ be a prime and r_1, r_2, r_3, r_4 be four distinct elements in $\mathbb{Z}_p$. Pick degree 3 polynomials $v_0(x), v_1(x), v_2(x), v_3(x)$ such that

$$\big(v_0(r_1), v_0(r_2), v_0(r_3), v_0(r_4)\big) = \boldsymbol{b} - 1 = (-1, -1, -1, 2)$$

and

$$\begin{pmatrix} v_1(r_1) & v_1(r_2) & v_1(r_3) & v_1(r_4) \\ v_2(r_1) & v_2(r_2) & v_2(r_3) & v_2(r_4) \\ v_3(r_1) & v_3(r_2) & v_3(r_3) & v_3(r_4) \end{pmatrix} = V = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix}.$$

Let $t(x) = (x - r_1)(x - r_2)(x - r_3)(x - r_4)$ to get a square span program $\big(v_0(x), v_1(x), v_2(x), v_3(x), t(x)\big)$ for the circuit such that

$$t(x) \quad \text{divides} \quad \Big(v_0(x) + a_1 v_1(x) + a_2 v_2(x) + a_3 v_3(x)\Big)^2 - 1$$

if and only if a_1, a_2, a_3 satisfy the circuit, i.e., $a_1, a_2 \in \{0, 1\}$, $a_3 = 1$ and $a_3 = a_1 \oplus a_2$.

2.4 Comparison to quadratic span programs

Square span programs can be seen as a simplification of quadratic span programs. Below we recall the definition of quadratic span programs given by Gennaro, Gentry, Parno and Raykova [GGPR13].

Definition 2. *A quadratic span program over a field $\mathbb{F}$ contains two sets of polynomials $\mathcal{V} = \{v'_0(x), \ldots, v_m(x)\}$ and $\mathcal{W} = \{w'_0(x), \ldots, w_m(x)\}$ and a target polynomial $t(x)$. It also contains a partition of the indices $\mathcal{I} = \{1, \ldots, m\}$ into $\mathcal{I} = \mathcal{I}_{\text{labeled}} \cup \mathcal{I}_{\text{free}}$ and a further partition $\mathcal{I}_{\text{labeled}} = \cup_{i=1,j=0}^{\ell,1} \mathcal{I}_{i,j}$.*

For input[4] *$y \in \{0,1\}^\ell$, let $\mathcal{I}_y = \mathcal{I}_{\text{free}} \cup_{i=1}^{\ell} \mathcal{I}_{i,y_i}$ be the set of indices that "belong" to y. The quadratic span program* accepts *an input $y \in \{0,1\}^\ell$ if and only if there exist $a_i, b_i \in \mathbb{F}$ such that*

$$t(x) \quad \text{divides} \quad \left(v'_0(x) + \sum_{i \in \mathcal{I}_y} a_i v_i(x)\right) \cdot \left(w'_0(x) + \sum_{i \in \mathcal{I}_y} b_i w_i(x)\right).$$

We say the quadratic span program verifies a boolean function $f : \{0,1\}^\ell \to \{0,1\}$ if it accepts exactly those inputs y where $f(y) = 1$. We say the size of the quadratic span program is m and the degree is $\deg(t(x))$.

[4] In the rest of the paper, we will be using inputs of the form $y = (u, w)$ where u of size ℓ_u is considered public and w is considered private.

Size and degree of Span Programs

	Size	Degree
Quadratic span programs [GGPR13]	$36n$	$130n$
Quadratic span programs (Lipmaa) [Lip13]	$14n - 14\ell - 2$	$11n - 12\ell - 2$
Square span programs	m	$m + n - \ell_u$

Table 2. Costs compared with prior work (ℓ input wires, out of which ℓ_u are public, m wires in total and n gates). In a circuit with fan-in 2 gates $m \leq 2n + 1$, so we get rough bounds of size $2n$ and degree $3n$ when computed as a function of the number of gates n only (ignoring ℓ_u).

A square span program uses the simpler condition

$$t(x) \quad \text{divides} \quad \left(v_0(x) + \sum_{i=1}^{m} a_i v_i(x)\right)^2 - 1,$$

which is equivalent to

$$t(x) \quad \text{divides} \quad \left(v_0(x) + 1 + \sum_{i=1}^{m} a_i v_i(x)\right) \cdot \left(v_0(x) - 1 + \sum_{i=1}^{m} a_i v_i(x)\right).$$

A square span program can therefore be seen as a particularly simple type of quadratic span program where $w'_0(x) = v'_0(x) - 2$ and $w_i(x) = v_i(x)$ and $a_i = b_i$. Furthermore, $\mathcal{I}_{\text{labeled}} = \{1, \ldots, \ell\}$ with $\mathcal{I}_{i,y_i} = \{i\}$ and $\mathcal{I}_{i,\bar{y}_i} = \emptyset$, and $\mathcal{I}_{\text{free}} = \{\ell + 1, \ldots, m\}$.

The compilation of a circuit into a quadratic span programs in [GGPR13] has a significant overhead. For a circuit with ℓ input wires and m wires in total and n gates, the size of the resulting quadratic span program is $36n$ and the degree is $130n$. Lipmaa [Lip13] gave a class of more efficient quadratic span programs. Included in this class is a quadratic span program of size $14n - 14\ell - 2$ and degree $11n - 12\ell - 2$. In comparison with these works our (square) quadratic span programs are much more compact with size $m - \ell_u$ and degree $m + n - \ell_u$ (assuming the verifier checks its inputs are all in $\{0, 1\}$.) These costs are summarised in Table 2.

A further advantage compared to the previous works is that we consider all types of logic gates, whereas they only consider NAND, AND and OR gates. We would expect that their constructions can be generalized to handle other logic gates but do not know whether this would increase the cost.

Remark. All three results prove that a circuit—fixed when the quadratic span program is generated—is satisfied for public input u and private input w. Universal circuits allow using a single program for all n' gate circuits at the cost of $n = n' \cdot 19 \log n'$ [Val76].

3 Succinct non-interactive arguments of knowledge

We will now use square span programs to construct succinct non-interactive zero-knowledge arguments of knowledge using bilinear groups.

Notation. 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 write $y = A(x; r)$ when the 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.

3.1 Non-interactive zero-knowledge arguments of knowledge

Let $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of families of efficiently decidable binary relations R. For pairs $(u, w) \in R$ we call u the statement and w the witness. A non-interactive argument for $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is a quadruple of efficient algorithms (Setup, Prove, Vfy, Sim) working as follows:

$(\sigma, \tau) \leftarrow \text{Setup}(1^\lambda, R)$**:** the setup algorithm takes as input a security parameter λ and a relation $R \in \mathcal{R}_\lambda$ and returns a common reference string σ and a simulation trapdoor τ for the relation R.

$\pi \leftarrow \text{Prove}(\sigma, u, w)$**:** the prover algorithm takes as input a common reference string σ for a relation R and $(u, w) \in R$ and returns an argument π.

$0/1 \leftarrow \text{Vfy}(\sigma, u, \pi)$**:** the verification algorithm takes as input a common reference string, a statement u and an argument π and returns 0 (reject) or 1 (accept).

$\pi \leftarrow \text{Sim}(\tau, u)$**:** the simulator takes as input a simulation trapdoor and a statement u and returns an argument π.

Definition 3. *We say* (Setup, Prove, Vfy, Sim) *is a perfect non-interactive zero-knowledge argument of knowledge for* $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ *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 $\lambda \in \mathbb{N}$, $R \in \mathcal{R}_\lambda$, $(u, w) \in R$

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

Perfect zero-knowledge. An argument is zero-knowledge if it does not leak any information besides the truth of the statement. We say (Setup, Prove, Vfy,

Sim) is perfect zero-knowledge if for all $\lambda \in \mathbb{N}, R \in \mathcal{R}_\lambda, (u,w) \in R$ and all adversaries $\mathcal{A}$, we have

$$\begin{aligned} & \Pr\left[(\sigma,\tau) \leftarrow \mathrm{Setup}(1^\lambda, R); \pi \leftarrow \mathrm{Prove}(\sigma,u,w) : \mathcal{A}(\sigma,\tau,\pi) = 1\right] \\ = \; & \Pr\left[(\sigma,\tau) \leftarrow \mathrm{Setup}(1^\lambda, R); \pi \leftarrow \mathrm{Sim}(\tau,u) : \mathcal{A}(\sigma,\tau,\pi) = 1\right]. \end{aligned}$$

Computational knowledge soundness. We call $(\mathrm{Setup}, \mathrm{Prove}, \mathrm{Vfy}, \mathrm{Sim})$ 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 sequences $(R_\lambda)_{\lambda\in\mathbb{N}}$ of polynomially bounded relations in $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ and non-uniform polynomial time adversaries $\mathcal{A}$, there exists a non-uniform polynomial time extractor $\mathcal{X}_\mathcal{A}$ such that

$$\Pr\left[\begin{array}{ll} (\sigma,\tau) \leftarrow \mathrm{Setup}(1^\lambda, R_\lambda) & \\ ((u,\pi);w) \leftarrow (\mathcal{A} \parallel \mathcal{X}_\mathcal{A})(\sigma) & : \begin{array}{c}(u,w) \notin R_\lambda \\ \mathrm{Vfy}(\sigma,u,\pi) = 1\end{array} \end{array}\right] \approx 0.$$

Remark. Our notion of knowledge soundness guarantees security against an *adaptive* adversary, cf. [AF07], that chooses the instance u depending on the CRS σ. However, to get adaptive security for circuit satisfiability, $\mathcal{R}_\lambda$ has to be universal, i.e., it has to check that a circuit u is satisfiable. For performance reasons, this is usually not what one wants, and adaptive soundness for a more restrictive $\mathcal{R}_\lambda$ is preferable. See Lipmaa [Lip14] for how to achieve adaptive soundness for some NP-complete languages, not including circuit satisfiability, while avoiding universal circuits.

3.2 Bilinear groups

Let $\mathcal{G}$ be a bilinear group generator that, on security parameter λ, returns $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$ with the following properties:

- $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are groups of prime order p;
- $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ is a bilinear map, that is, for all $U \in \mathbb{G},\ V \in \hat{\mathbb{G}},\ a, b \in \mathbb{Z}$, we have $e(U^a, V^b) = e(U,V)^{ab}$;
- if G is a generator for $\mathbb{G}$ and $\hat{G}$ is a generator for $\hat{\mathbb{G}}$ then $e(G,\hat{G})$ is a generator for $\mathbb{G}_T$; and
- 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.

There are many ways to set up bilinear groups both as symmetric bilinear groups where $\mathbb{G} = \hat{\mathbb{G}}$ and as asymmetric bilinear groups where $\mathbb{G} \neq \hat{\mathbb{G}}$. Our construction works for both symmetric and asymmetric bilinear groups. Currently, asymmetric bilinear groups are more efficient and therefore the most appropriate choice in practice [GPS08].

THE q-POWER KNOWLEDGE OF EXPONENT ASSUMPTION. The knowledge of exponent assumption (KEA) introduced by Damgård [Dam91] says that given $G, G' = G^\alpha$ it is infeasible to create V, V' such that $V' = V^\alpha$ without knowing a such that $V = G^a$ and $V' = G'^a$. Bellare and Palacio [BP04] extended this to the KEA3 assumption, which says that given G, G^s, G', G'^s it is infeasible to create $V, V' = V^\alpha$ without knowing a_0, a_1 such that $V = G^{a_0}(G^s)^{a_1}$. This assumption has been used also in symmetric bilinear groups by Abe and Fehr [AF07] who called it the extended knowledge of exponent assumption.

The q-power knowledge of exponent assumption is a generalization of these assumptions in bilinear groups. It says that given $(G, \hat{G}, G^s, \hat{G}^s, \ldots, G^{s^q}, \hat{G}^{s^q})$ it is infeasible to create $V, \hat{V}$ such that $e(V, \hat{G}) = e(G, \hat{V})$ without knowing $a_0, \ldots, a_q$ such that $V = \prod_{i=0}^{q} (G^{s^i})^{a_i}$. The q-power knowledge of exponent assumption was introduced in [Gro10] for symmetric bilinear groups using $\hat{G} = G^\alpha$ with α chosen at random. Here we adapt it with minor modifications to the general setting where it may be the case that $\mathbb{G} \neq \hat{\mathbb{G}}$ and $G, \hat{G}$ belong to different groups.

Definition 4 (q-PKE). *The $q(\lambda)$-power knowledge of exponent assumption holds relative to $\mathcal{G}$ for the class $\mathcal{Z}$ of auxiliary input generators if, for every non-uniform polynomial time auxiliary input generator $Z \in \mathcal{Z}$ and non-uniform polynomial time adversary $\mathcal{A}$, there exists a non-uniform polynomial time extractor $\mathcal{X}_\mathcal{A}$ such that*

$$\Pr\left[\begin{array}{l} gk := (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda); G \leftarrow \mathbb{G}^* \\ s \leftarrow \mathbb{Z}_p^*; z \leftarrow Z(gk, G, \ldots, G^{s^q}); \hat{G} \leftarrow \hat{\mathbb{G}}^* \\ (V, \hat{V}; a_0, \ldots, a_q) \leftarrow (\mathcal{A} \parallel \mathcal{X}_\mathcal{A})(gk, G, \hat{G}, \ldots, G^{s^q}, \hat{G}^{s^q}, z) \ : \\ e(V, \hat{G}) = e(G, \hat{V}) \ \wedge \ V \neq G^{\sum_{i=0}^{q} a_i s^i} \end{array}\right] \approx 0.$$

An adaptation of the proof in Groth [Gro10] shows that the q-PKE assumption holds in the generic bilinear group model.

As demonstrated by Bitansky, Canetti, Paneth and Rosen [BCPR13], if indistinguishability obfuscators [BGI+12,GGH+13] exist, then there are auxiliary input generators for which the q-PKE assumption does not hold. However, their counterexample is specifically tailored to make extraction difficult and, as they explain, the q-PKE assumption may hold for "benign" auxiliary input generators. We will later use auxiliary input generators that generate group elements in $\mathbb{G}$ and $\hat{\mathbb{G}}$ in a specific manner according to the relations R_λ and we will conjecture that such auxiliary input generators are benign and that the q-PKE assumption holds with respect to them.

THE q-POWER DIFFIE-HELLMAN ASSUMPTION. The q-power Diffie-Hellman assumption says given $(G, \hat{G}, \ldots, G^{s^q}, \hat{G}^{s^q}, G^{s^{q+2}}, \hat{G}^{s^{q+2}}, \ldots, G^{s^{2q}}, \hat{G}^{s^{2q}})$ it is hard to compute the missing element $G^{s^{q+1}}$.

Definition 5 (q-PDH). *The $q(\lambda)$-power Diffie-Hellman assumption holds relative to $\mathcal{G}$ if for all non-uniform probabilistic polynomial time adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{l} gk := (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda); G \leftarrow \mathbb{G}^*; \hat{G} \leftarrow \hat{\mathbb{G}}^*; s \leftarrow \mathbb{Z}_p^* \\ Y \leftarrow \mathcal{A}(gk, G, \hat{G}, \ldots, G^{s^q}, \hat{G}^{s^q}, G^{s^{q+2}}, \hat{G}^{s^{q+2}}, \ldots, G^{s^{2q}}, \hat{G}^{s^{2q}}) \ : \\ Y = G^{s^{q+1}} \end{array}\right] \approx 0.$$

An adaptation of the proof in Groth [Gro10] shows that the q-PDH assumption holds in the generic bilinear group model.

The q-target group strong Diffie-Hellman assumption. We adapt the strong Diffie-Hellman assumption [BB08] in the target group [PHGR13] to the asymmetric setting. It says that given $(G, \hat{G}, \ldots, G^{s^q}, \hat{G}^{s^q})$ it is hard to find an $r \in Z_p$ and compute $e(G, \hat{G})^{\frac{1}{s-r}}$.

Definition 6 (q-TSDH). *The $q(\lambda)$-target group strong Diffie-Hellman assumption holds relative to $\mathcal{G}$ if for all non-uniform probabilistic polynomial time adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{l} (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda); G \leftarrow \mathbb{G}^*; \hat{G} \leftarrow \hat{\mathbb{G}}^*; s \leftarrow \mathbb{Z}_p^* \\ (r, Y) \leftarrow \mathcal{A}(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^q}, \hat{G}^{s^q}) \ : \\ r \in \mathbb{Z}_p \setminus \{s\} \ \wedge \ Y = e(G, \hat{G})^{\frac{1}{s-r}} \end{array}\right] \approx 0.$$

An adaptation of the proof in Boneh and Boyen [BB08] shows that the q-TSDH assumption holds in the generic bilinear group model.

3.3 Succinct perfect NIZK arguments

We will now construct succinct and perfect NIZK arguments of knowledge for any functions ℓ_u, ℓ_w and families $\{\mathcal{R}\}_\lambda$ of relations R of pairs $(u, w) \in \{0,1\}^{\ell_u(\lambda)} \times \{0,1\}^{\ell_w(\lambda)}$ that can be computed by polynomial size circuits with $m(\lambda)$ wires and $n(\lambda)$ gates for a total size of $d(\lambda) = m(\lambda) + n(\lambda)$.

$(\sigma, \tau) \leftarrow \text{Setup}(1^\lambda, R)$**:** Run $gk := (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$. Parse R as a boolean circuit $C_R : \{0,1\}^{\ell_u} \times \{0,1\}^{\ell_w} \to \{0,1\}$. Generate a square span program $Q = \big(v_0(x), \ldots, v_m(x), t(x)\big)$ that verifies C_R over $\mathbb{Z}_p$. Pick $G \leftarrow \mathbb{G}^*$ and $\hat{G}, \tilde{G} \leftarrow \hat{\mathbb{G}}^*$ and $\beta, s \leftarrow \mathbb{Z}_p^*$ such that $t(s) \neq 0$. Return

$$\begin{array}{rcl} \sigma & = & (gk, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d}, \{G^{\beta v_i(s)}\}_{i > \ell_u}, G^{\beta t(s)}, \tilde{G}, \tilde{G}^\beta, Q) \\ \tau & = & (\sigma, \beta, s). \end{array}$$

$\pi \leftarrow \text{Prove}(\sigma, u, w)$**:** Parse u as $(a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$ and use w to compute $a_{\ell_u+1}, \ldots, a_m$ such that $t(x)$ divides $\left(v_0(x) + \sum_{i=1}^m a_i v_i(x)\right)^2 - 1$.
Pick $\delta \leftarrow \mathbb{Z}_p$ and let

$$h(x) = \frac{\left(v_0(x) + \sum_{i=1}^m a_i v_i(x) + \delta t(x)\right)^2 - 1}{t(x)}.$$

Use linear combinations of the elements in σ to compute

$$H = G^{h(s)} \qquad V_w = G^{\sum_{i>\ell_u}^{m} a_i v_i(s) + \delta t(s)}$$
$$B_w = G^{\beta\left(\sum_{i>\ell_u}^{m} a_i v_i(s) + \delta t(s)\right)} \qquad \hat{V} = \hat{G}^{v_0(s) + \sum_{i=1}^{m} a_i v_i(s) + \delta t(s)}$$

and return $\pi = (H, V_w, B_w, \hat{V})$.

$0/1 \leftarrow \mathrm{Vfy}(\sigma, u, \pi)$**:** Parse u as $(a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$ and π as $(H, V_w, B_w, \hat{V}) \in \mathbb{G}^3 \times \hat{\mathbb{G}}$. Compute $V = G^{v_0(s) + \sum_{i=1}^{\ell_u} a_i v_i(s)} V_w$ and return 1 if and only if

$$e(V, \hat{G}) = e(G, \hat{V}) \quad e(H, \hat{G}^{t(s)}) = e(V, \hat{V}) e(G, \hat{G})^{-1} \quad e(V_w, \tilde{G}^{\beta}) = e(B_w, \tilde{G}).$$

$\pi \leftarrow \mathrm{Sim}(\tau, u)$**:** Parse u as $(a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$ and pick $\delta_w \leftarrow \mathbb{Z}_p$ at random. Let

$$h = \frac{\left(v_0(s) + \sum_{i=1}^{\ell_u} a_i v_i(s) + \delta_w\right)^2 - 1}{t(s)}$$

and return $\pi = (G^h, G^{\delta_w}, G^{\beta\delta_w}, \hat{G}^{v_0(s) + \sum_{i=1}^{\ell_u} a_i v_i(s) + \delta_w})$.

Let $\mathcal{Z}$ be a family of non-uniform polynomial time auxiliary input generators Z such that each of them corresponds to sequences of relations $(R_\lambda)_{\lambda\in\mathbb{N}}$ in a family of relations $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$. They work such that Z corresponding to $(R_\lambda)_{\lambda\in\mathbb{N}}$ on input $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \ldots, G^{s^q})$ generates the final part of the common reference string, i.e., returns $z = (\{G^{\beta v_i(s)}\}_{i>\ell_u}, G^{\beta t(s)}, \tilde{G}, \tilde{G}^{\beta}, Q)$.

Theorem 3. *The construction above is a perfect NIZK argument for the family of relations $\{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ bounded by $d(\lambda)$ with computational knowledge soundness if the $d(\lambda)$-PKE, $d(\lambda)$-PDH and $d(\lambda)$-SDH assumptions hold relative to $\mathcal{G}$ and the family of auxiliary input generator $\mathcal{Z}$ defined above.*

Proof. Perfect completeness follows by direct verification.

Perfect zero-knowledge follows from observing that both a real argument and a simulated argument have a uniformly random V_w because $t(s) \neq 0$ and δ, δ_w are chosen uniformly at random. Once V_w has been fixed, the verification equations uniquely determine $B_w, \hat{V}$ and H. This means that for any $(u, w) \in R$ both the real arguments and the simulated arguments are chosen uniformly at random such that the verification equations will be satisfied.

We now describe the witness-extractor for computational knowledge soundness. The setup algorithm first generates a bilinear group $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$ and picks $G \leftarrow \mathbb{G}^*$ and $s \leftarrow \mathbb{Z}_p^*$, which are used to compute $G, \ldots, G^{s^d}$. This is exactly like the input given to the auxiliary input generator in a d-PKE challenge. The setup algorithm now generates a square span program Q over $\mathbb{Z}_p$ for the relation R_λ and elements $\{G^{\beta v_i(s)}\}_{i>\ell_u}$ and $\tilde{G}, \tilde{G}^{\beta}$. We can consider this as part of the auxiliary input z that Z outputs in the d-PKE definition. More precisely, let $\mathcal{A}'$ be the d-PKE adversary that, on $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d})$ and auxiliary input $z = (\{G^{\beta v_i(s)}\}_{i>\ell_u}, G^{\beta t(s)}, \tilde{G}, \tilde{G}^{\beta}, Q)$ runs $(u, H, V_w, B_w, \hat{V}) \leftarrow \mathcal{A}(\sigma)$ with $\sigma = (p, \ldots, \hat{G}^{s^d})$ and returns $(V, \hat{V})$ where

$V = G^{v_0(s)+\sum_{i=1}^{\ell_u} a_i v_i(s)} V_w$ when $u = (a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$. Let $\mathcal{X}_{\mathcal{A}'}$ be the corresponding extractor according to the d-PKE assumption that returns $c_0, \ldots, c_d$ such that $V = G^{\sum_{i=0}^{d} c_i s^i}$ when $e(V, \hat{G}) = e(G, \hat{V})$. Our witness-extractor $\mathcal{X}_{\mathcal{A}}$ given σ runs $(V, \hat{V}; c_0, \ldots, c_d) \leftarrow (\mathcal{A}' \parallel \mathcal{X}_{\mathcal{A}'})(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d}, z)$, which defines a polynomial $\sum_{i=0}^{d} c_i x^i$. Define $\delta = c_d$ to get a degree $d-1$ polynomial $v(x) = \sum_{i=0}^{d} c_i x^i - \delta t(x)$. If it is possible to write the polynomial on the form $v(x) = v_0(x) + \sum_{i=1}^{m} a_i v_i(x)$ such that $(a_1, \ldots, a_m) \in \{0,1\}^m$ is a satisfying assignment for the circuit C_R with $u = (a_1, \ldots, a_{\ell_u})$ then the extractor returns $w = (a_{\ell_u+1}, \ldots, a_{\ell_u+\ell_w})$.

We will now show that with all but negligible probability the extracted polynomial $v(x)$ does indeed provide a valid witness $w \in \{0,1\}^{\ell_w}$ such that $(u, w) \in \mathcal{R}_\lambda$. Let Q be the square span program $(v_0(x), \ldots, v_m(x), t(x))$ specified in σ that verifies R_λ over $\mathbb{Z}_p$. We know by Theorem 2 that if $t(x)$ divides $v(x)^2 - 1$ and $v_{\text{mid}}(x) = \sum_{i=0}^{d} c_i x^i - v_0(x) - \sum_{i=1}^{\ell_u} a_i v_i(x)$ belongs to the span of $\{v_i(x)\}_{i>\ell_u}$ then indeed $w \in \{0,1\}^{\ell_w}$ and $(u, w) \in \mathcal{R}_\lambda$. So we will in the following show that the two cases, $t(x)$ does not divide $v(x)^2 - 1$ or $v_{\text{mid}}(x)$ is not in the appropriate span both happen with negligible probability breaking the d-TSDH assumption or the d-PDH assumption respectively.

Given a d-TSDH challenge $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d})$, we pick $\beta \leftarrow \mathbb{Z}_p^*$ and roots $r_1, \ldots, r_d$ in the same way the setup algorithm does and simulate a common reference string σ. Suppose the adversary and extractor return $u = (a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$, a valid proof $\pi = (H, V_w, B_w, \hat{V})$ and $c_0, \ldots, c_d$ such that $V = G^{v_0(s)+\sum_{i=1}^{\ell_u} a_i v_i(s)} V_w = G^{\sum_{i=0}^{d} c_i s^i}$. Let $v(x) = \sum_{i=0}^{d} c_i x^i - \delta t(x)$ with $\delta = c_d$ as before and define $p(x) = (v(x) + \delta t(x))^2 - 1$ and suppose $p(x)$ is not divisible by $t(x)$. Let r_i be a root of $t(x)$ such that $x - r_i$ does not divide $p(x)$. We can write $p(x) = a(x)(x - r_i) + b$, where $a(x)$ is a degree $2d-1$ polynomial in $\mathbb{Z}_p[x]$ and $b \in \mathbb{Z}_p^*$. The verification equation $e(H, \hat{G}^{t(s)}) = e(V, \hat{V}) e(G, \hat{G})^{-1}$ gives us $e(H, \hat{G}^{\frac{t(s)}{s-r_i}}) = e(G, \hat{G})^{a(s)+\frac{b}{s-r_i}}$. The adversary can use generic group operations on the d-TSDH challenge to compute $\hat{G}^{\frac{t(s)}{s-r_i}}$ and $e(G, \hat{G})^{a(s)}$, which allows it to deduce $e(G, \hat{G})^{\frac{b}{s-r_i}}$. Rasing this to the power b^{-1} gives a solution $(r_i, e(G, \hat{G})^{\frac{1}{s-r_i}})$ to the d-TSDH challenge.

Given a d-PDH challenge $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d}, G^{s^{d+2}}, \hat{G}^{s^{d+2}}, \ldots, G^{s^{2d}}, \hat{G}^{s^{2d}})$ we pick a random degree d polynomial $a(x)$ such that $a(x)v_i(x)$ has coefficient 0 for x^d for all $\ell_u < i \leq m$ and $a(x)t(x)$ also has coefficient 0 for x^d. There are $d + \ell_u - m - 1 > 0$ degrees of freedom in choosing $a(x)$ so for a polynomial $v_{\text{mid}}(x)$ outside the span of $\{v_i(x)\}_{i=\ell_u}^{m}$ and $t(x)$ the polynomial $a(x)v_{\text{mid}}(x)$ has a random coefficient for x^d.

Now pick at random $b \leftarrow \mathbb{Z}_p$ and define $\beta(x) = a(x)x + b$ and let $\beta = \beta(s)$. Observe that $G^{\beta v_i(s)} = G^{(a(s)s+b)v_i(s)}$ can be constructed from our challenge without knowing $G^{s^{d+1}}$; and the same goes for $G^{\beta t(s)}$. Pick $\rho \leftarrow \mathbb{Z}_p^*$ at random and compute $\tilde{G} = \hat{G}^{\rho t(s)}$ and $\tilde{G}^{\beta} = \hat{G}^{\rho \beta t(s)}$. Give to the adversary a simulated

common reference string

$$\sigma = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d}, \{G^{\beta v_i(s)}\}_{i>\ell_u}, G^{\beta t(s)}, \tilde{G}, \tilde{G}^{\beta}, Q).$$

Suppose the adversary and extractor return $u = (a_1, \ldots, a_{\ell_u}) \in \{0,1\}^{\ell_u}$, a valid proof $\pi = (H, V_w, B_w, \hat{V})$ and $c_0, \ldots, c_d$ such that $V = G^{\sum_{i=0} c_i s^i}$. Define $v_{\text{mid}}(x) = \sum_{i=0}^{d} c_i x^i - v_0(x) - \sum_{i=1}^{\ell_u} a_i v_i(x)$. Due to the random choice of b the value $\beta(s) = a(s)s + b$ does not reveal anything about $a(x)$, so if $v_{\text{mid}}(x)$ is outside the span of $\{v_i(x)\}_{i>\ell_u}$ and $t(x)$ then $a(x)v_{\text{mid}}(x)$ has a random coefficient for x^{d+1}. With probability $1 - \frac{1}{p}$ this means the adversary returns $B_w = G^{\beta(s)v_{\text{mid}}(s)}$ where $\beta(x)v_{\text{mid}}(x) = \sum_{i=0}^{2d} b_i x^i$ is a known polynomial with a non-trivial coefficient $b_{d+1} \neq 0$ for x^{d+1}. We can now take an appropriate linear combination of B_w and the elements $G, \ldots, G^{s^d}, G^{s^{d+2}}, \ldots, G^{s^{2d}}$ to compute $G^{s^{d+1}}$, which solves the d-PDH challenge. □

The proof of Theorem 3 suffers a computational overhead in the reduction by using an extractor $\mathcal{X}_{\mathcal{A}}$ for $\mathcal{A}$. Except for this computational overhead, the security reduction for knowledge soundness is tight. It is possible to eliminate the q-TSDH assumption and rely solely on the q-PKE and q-PDH assumptions, but then the security reduction loses a factor q and is therefore not tight.

3.4 Efficiency

In this section, we will assume our NIZK argument is instantiated with the square span program that we constructed in Section 2.2. This choice of square span program enables a number of optimizations that makes the argument highly efficient.

The prover has to compute

$$\begin{aligned} V_w &= G^{\sum_{i>\ell_u}^{m} a_i v_i(s) + \delta t(s)} \\ B_w &= G^{\beta\left(\sum_{i>\ell_u}^{m} a_i v_i(s) + \delta t(s)\right)} \\ \hat{V} &= \hat{G}^{v_0(s) + \sum_{i=1}^{m} a_i v_i(s) + \delta t(s)}. \end{aligned}$$

It is possible to compute the polynomials $\sum_{i>\ell_u}^{m} a_i v_i(x) + \delta t(x)$ and $v_0(x) + \sum_{i=1}^{m} a_i v_i(x) + \delta t(x)$ and then compute the appropriate exponentiations of the polynomials evaluated in s using the elements $G, \hat{G}, \ldots, G^{s^d}, \hat{G}^{s^d}, \{G^{\beta v_i(s)}\}_{i>\ell_u}$, and $G^{\beta t(s)}$ from the common reference string. However, this requires $O(d)$ exponentations to the coefficients of the polynomials. Following [GGPR13] a significant saving can be made by precomputing $\{G^{v_i(s)}\}_{i>\ell_u}$, $G^{t(s)}$ and $\{\hat{G}^{v_i(s)}\}_{i=0}^{m}$, $\hat{G}^{t(s)}$. Since each $a_i \in \{0,1\}$ this makes it possible to compute V_w, B_w and $\hat{V}$ using at most $3m + 1 - 2\ell_u$ multiplications and 3 exponentiations. (Pragmatically, taking advantage of our uniform support for all gates, we can profile the SSP and 'flip' internal values from a_i to $\overline{a}_i$ to ensure that a_i is more often equal to 0 than to 1, thereby on average performing less than half of those multiplications.)

The prover also has to compute $H = G^{h(s)}$, where $h(x) = \frac{(v(x)+\delta t(x))^2 - 1}{t(x)}$ with $v(x) = v_0(x) + \sum_{i=1}^{m} a_i v_i(x)$ and $t(x) = \prod_{i=1}^{d}(x - r_i)$. We can evaluate $h(x)$ in

d points $r'_1, \ldots, r'_d$ using two discrete Fourier transforms as follows. The degree $d-1$ polynomial $v(x)$ is uniquely determined by its evaluation in the d points $r_1, \ldots, r_d$. In our square span program the evaluations in the points $r_1, \ldots, r_d$ can be computed easily given the values of the wires in the circuit. Using an inverse discrete Fourier transform, we compute the coefficients of $v(x) = \sum_{i=0}^{d-1} c_i x^i$. Let $\gamma \in \mathbb{Z}_p^*$ be given such that $r'_1, \ldots, r'_d$ defined as $r'_i = \gamma^i r_i$ gives us $2d$ distinct values $r_1, \ldots, r_d, r'_1, \ldots, r'_d$. Compute $c'_i = \gamma^i c_i$ to get the coefficients of the polynomial $v'(x) = \sum_{i=0}^{d-1} c'_i x^i$ and use a discrete Fourier transform to evaluate $v'(x)$ in $r_1, \ldots, r_d$. This gives us evaluations of $v(x)$ in the points $r'_1, \ldots, r'_d$ since $v(r'_j) = v'(r_j)$. We have $h(x) = \frac{v(x)^2-1}{t(x)} + 2\delta v(x) + \delta^2 t(x)$. Assuming $t(r'_1)^{-1}, \ldots, t(r'_d)^{-1}$ have been precomputed, it only costs $3d$ multiplications in $\mathbb{Z}_p$ to evaluate $\frac{v(x)^2-1}{t(x)} + 2\delta v(x)$ in the d points $r'_1, \ldots, r'_d$. Using Lagrange interpolation in the exponent, this allows us to compute

$$G^{\frac{v(s)^2-1}{t(s)}+2\delta v(s)} = \prod_{j=1}^{d} (G^{\ell'_j(s)})^{\frac{v(r'_j)^2-1}{t(r'_j)}+2\delta v(r'_j)}$$

where $\ell'_j(x)$ is the Lagrange basis polynomial for r'_j. By multiplying with $(G^{t(s)})^{\delta^2}$ we then get $G^{h(s)}$.

To speed up the computation, we can set up a modified common reference string for the prover

$$\sigma_{\text{Prove}} = \left(\begin{array}{c} p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \hat{G}, \{G^{v_i(s)}\}_{i>\ell_u}, \{\hat{G}^{v_i(s)}\}_{i>\ell_u}, \{G^{\beta v_i(s)}\}_{i>\ell_u}, \\ G^{\beta t(s)}, \tilde{G}, \tilde{G}^{\beta}, \gamma, \{t(r'_j)^{-1}\}_{j=1}^{d}, \{G^{\ell'_j(s)}\}_{j=1}^{d}, G^{t(s)}, Q \end{array} \right).$$

The computational cost for the prover is dominated by d exponentations in $\mathbb{G}$ and 2 discrete Fourier transforms in $\mathbb{Z}_p$. The two discrete Fourier transforms cost $O(d \log^2 d)$ multiplications in general but the computation can be reduced to $O(d \log d)$ multiplications when $\mathbb{Z}_p$ is of a form amenable to using the fast Fourier transform.

The verifier needs to compute $V = G^{v_0(s)+\sum_{i=1}^{\ell_u} a_i v_i(s)} V_w$ and evaluate three pairing product equations $e(V, \hat{G}) = e(G, \hat{V})$, $e(H, \hat{G}^{t(s)}) = e(V, \hat{V}) e(G, \hat{G})^{-1}$, and $e(V_w, \tilde{G}^{\beta}) = e(B_w, \tilde{G})$. The verifier does not need the full common reference string but can use a more compact common reference string

$$\sigma_{\text{Vfy}} = \left(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, G, \{G^{v_i(s)}\}_{i=0}^{\ell_u}, \hat{G}, \hat{G}^{t(s)}, \tilde{G}, \tilde{G}^{\beta} \right),$$

which only has $\ell_u + 6$ group elements.[5] Verification is also computationally efficient, in the worst case it requires $\ell_u + 1$ multiplications in $\mathbb{G}$, one multiplication in $\mathbb{G}_T$ and 6 pairings if we precompute $e(G, \hat{G})^{-1}$.

For a large circuit, the cost of verification can be much smaller than the cost of evaluating the circuit itself, even if the witness w is known to the verifier. This

[5] Using the binary representation of the public input u from [PHGR13] this can be further reduced to $\lceil \frac{\ell_u}{\lambda} \rceil + O(1)$ group elements.

Proof size and verification cost comparison with Pinocchio

	Proof Size (elements)	Verification cost
Pinocchio [PHGR13]	8	$14P + (\ell_u + 4)\mathbb{G} + 1\mathbb{G}_T$
This work	4	$6P + (\ell_u + 1)\mathbb{G} + 1\mathbb{G}_T$

Table 3. Size in number of group elements (either $\mathbb{G}$ or $\hat{\mathbb{G}}$), performance in terms of pairings (P) or multiplications in $\mathbb{G}$ or $\mathbb{G}_T$ respectively.

makes the NIZK argument a succinct non-interactive argument of knowledge that is suitable for verifiable computation protocols.[6]

Partly due to the lack of benchmarks, it is hard to compare the performance of SNARK protocols quantitatively without carefully reimplementing them. Table 3 compares the proof sizes and operations performed by the verifier between our protocol and Pinocchio, arguably the state of the art in terms of proof size and verification speed for QAPs. On this basis and the numbers reported in [PHGR13], we conservatively estimate that an SSP implementation based on the Pinocchio library would offer 160-byte proofs verified in less than 6 ms.

4 Conclusion

We introduce a representation of logic circuits, or predicates on propositional formulae, using quadratic constraints on an affine map. The map is built using a linearization of each gate, and a set of constraints to ensure all values of wires are binary. This leads to a simple and elegant formulation of square span programs, and in turn to efficient, minimalistic constructions for NIZKs and SNARKs.

The simplifications are twofold: (i) our representation of boolean functions no longer requires wire checkers and (ii) square span programs consist of only a single set of polynomials that are summed and squared. The former improves prover efficiency, while the key advantage of the latter are SNARKs with an extremely compact proof, consisting of only four group elements, and an efficient verification procedure compared to more generic QSP characterisations of the same program.

As can be expected, binary programs such as SSPs remain less efficient than arithmetic programs for verifying computations on integers, involving e.g. 32-bit additions and multiplications. Those operations have to be encoded as binary adders and multipliers, leading to a significant blow-up in circuit size and computation costs for the prover. It remains an open problem how to extend the SSP approach with ideas from QAPs to verify such computations without sacrificing its conceptual simplicity and short proofs.

[6] In some cases, for instance when outsourcing computation, the verifier may be the one that sets up the common reference string. In that case the verifier may know β and s, which can further decrease the cost of verification.

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.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, 2008.

[BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. Indistinguishability obfuscation vs. auxiliary-input extractable functions: One must fall. IACR Cryptology ePrint Archive, Report 2013/641, 2013.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012.

[BP04] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 48–62, 2004.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Practical decentralized anonymous e-cash from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE, May 2014.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456, 1991.

[DFKP13] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In Martin Franz, Andreas Holzer, Rupak Majumdar, Bryan Parno, and Helmut Veith, editors, *PETShop@CCS*, pages 27–30. ACM, 2013.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 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.

[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.

[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.

[KW93] M. Karchmer and A. Wigderson. On span programs. In *In Proc. of the 8th IEEE Structure in Complexity Theory*, pages 102–111. IEEE Computer Society Press, 1993.

[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.

[Lip14] Helger Lipmaa. Almost optimal short adaptive non-interactive zero knowledge. Cryptology ePrint Archive, Report 2014/396, 2014. http://eprint.iacr.org/.

[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.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *STOC*, pages 196–203, 1976.

[vEB81] Peter van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical report at http://staff.science.uva.nl/ peter/vectors/mi8104c.html, 1981.