Quadratic Span Programs and Succinct NIZKs without PCPs

Rosario Gennaro*, Craig Gentry**,
Bryan Parno***, and Mariana Raykova†

Abstract. We introduce a new characterization of the NP complexity class, called *Quadratic Span Programs* (QSPs), which is a natural extension of *span programs* defined by Karchmer and Wigderson. Our main motivation is the quick construction of succinct, easily verified arguments for NP statements.

To achieve this goal, QSPs use a new approach to the well-known technique of *arithmetization* of Boolean circuits. Our new approach yields dramatic performance improvements. Using QSPs, we construct a NIZK argument – in the CRS model – for Circuit-SAT consisting of just 7 *group elements*. The CRS size and prover computation are *quasi-linear*, making our scheme seemingly quite practical, a result supported by our implementation. Indeed, our NIZK argument attains the shortest proof, most efficient prover, and most efficient verifier of any known technique. We also present a variant of QSPs, called *Quadratic Arithmetic Programs* (QAPs), that "naturally" compute *arithmetic* circuits over large fields, along with succinct NIZK constructions that use QAPs.

Finally, we show how QSPs and QAPs can be used to efficiently and publicly verify outsourced computations, where a client asks a server to compute $F(x)$ for a given function F and must verify the result provided by the server in considerably less time than it would take to compute F from scratch. The resulting schemes are the most efficient, general-purpose publicly verifiable computation schemes.

* City University of New York. rosario@ccny.cuny.edu. The research of this author was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

** IBM T.J.Watson Research Center. cbgentry@us.ibm.com

*** Microsoft Research. parno@microsoft.com

† IBM T.J.Watson Research Center, supported by NSF Grant No.1017660. mariana@cs.columbia.edu

1 Introduction

Arithmetization of Boolean computations is a well known technique: it maps a Boolean circuit to a set of polynomial (e.g., quadratic) equations over a field. The celebrated result IP=PSPACE [35, 41] used arithmetization as a crucial tool and set the stage for the PCP theorem [2–4, 20], which provided *a new characterization of NP* that revolutionized the notion of "proof" – in particular, it shows that NP statements have probabilistically checkable proofs (PCPs) that can be verified in time polylogarithmic in the size of a classical proof.

Cryptographers quickly seized on the potential applicability of PCPs to secure computation. Kilian [32] showed how to use PCPs to construct interactive *arguments* (i.e., computationally sound proof systems [14]) for NP that are *succinct* – i.e., polylogarithmic in their communication complexity. Micali [36] showed how to make these arguments *non-interactive* in the random oracle model. Recent work [8, 19, 26] (see also [17]) has improved Micali's construction by removing the random oracle, which is known to be uninstantiable [15], and replacing it with an "extractable collision-resistant hash function" (ECRH), whose security relies on the plausible, but *non-falsifiable* [37], assumption that for any algorithm that computes an image of the ECRH, there is an extractor (that watches the algorithm) that computes a pre-image.[1] These recent constructions have been called succinct non-interactive arguments (SNARGs) of *knowledge* (SNARKs), since, under the knowledge assumption, the SNARG permits "knowledge" extraction of the entire hash preimage – i.e., the entire PCP.

PCPs are not the only arithmetization technique for creating SNARKs. Groth shows how to arithmetize a Boolean circuit so that a proof of its satisfiability can be written using only a constant number of group elements [27] (after a single pre-processing stage to establish a common reference string (CRS) [9, 10]).

Our work provides a brand new form of arithmetization which we call *Quadratic Span Programs* (QSPs), since it is a generalization of the notion of Span Programs proposed by Karchmer and Wigderson [31]. We show that our new arithmetization technique yields far more efficient SNARKs than either PCP-based or Groth-like proofs. Using QSPs, we construct a NIZK argument in the CRS model for circuit SAT consisting of just 7 *group elements*. The CRS size and prover computation are *quasi-linear*

[1] We know that the security of succinct non-interactive arguments cannot be based on falsifiable assumptions via black box reductions [1, 23]; hence non-falsifiable "knowledge" assumptions seem unavoidable in this context.

in the circuit size, making our scheme quite practical, to the point where we have implemented and evaluated it (see Section 5). A variant of our technique works directly on *arithmetic* circuits over large fields, obtaining *Quadratic Arithmetic Programs* (QAPs) and avoiding the complexity of a Boolean description of an arithmetic computation (see Section 4).

1.1 Quadratic Span Programs

QSPs are a natural extension of span programs (SPs), a linear-algebraic model of computation introduced by Karchmer and Wigderson [31].[2] An SP of size m over a field F consists of a set $\mathcal{V} = \{v_0(x), v_1(x), \ldots, v_m(x)\}$ of polynomials of degree $d-1$, a partition of the indices $\mathcal{I} = \{1, \ldots, m\}$ into two sets $\mathcal{I}_{labeled}$ and $\mathcal{I}_{free}$, and a further partition of $\mathcal{I}_{labeled}$ as $\cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}$ meant to represent n Boolean inputs. The SP is said to "compute" a function f if the following is true for all input assignments $u \in \{0,1\}^n$: the polynomial $v_0(x)$ can be expressed as a linear combination of the polynomials that "belong" to the input assignment u – namely, the set of polynomials $\mathcal{V}_u$ with indices in $\mathcal{I}_u = \mathcal{I}_{free} \cup_i \mathcal{I}_{i,u_i}$ – iff $f(u) = 1$.

Functions with polynomial size SPs are in NC^2, since linear algebra is in NC^2. Consequently, it is widely believed that SPs *cannot* efficiently compute all functions in P (or verify all NP relations).

We define QSPs somewhat similarly to SPs.

Definition 1 (Quadratic Span Program). *A quadratic span program (QSP) Q over field F contains two sets of polynomials $\mathcal{V} = \{v_k(x) : k \in \{0, \ldots, m\}\}$ and $\mathcal{W} = \{w_k(x) : k \in \{0, \ldots, m\}\}$ and a divisor polynomial $D(x)$, all from $F[x]$. Q also contains a partition of the indices $\mathcal{I} = \{1, \ldots, m\}$ into two sets $\mathcal{I}_{labeled}$ and $\mathcal{I}_{free}$, and a further partition of $\mathcal{I}_{labeled}$ as $\cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}$.*

For input $u \in \{0,1\}^n$, let $\mathcal{I}_u = \mathcal{I}_{free} \cup_i \mathcal{I}_{i,u_i}$ be the set of indices that "belong" to input u. Q accepts *an input $u \in \{0,1\}^n$ iff there exist tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ from F^m, with $a_k = 0 = b_k$ for all $k \notin \mathcal{I}_u$:*

$$D(x) \quad \text{divides} \quad \left(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} b_k \cdot w_k(x)\right) . \quad (1)$$

Q "computes" a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if it accepts exactly those inputs u where $f(u) = 1$. Q has size m and degree $\deg(D(x))$.

[2] SPs were first defined [31] in terms of vectors $\{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_m\}$, rather than polynomials. The "target" vector $\mathbf{v}_0$ must be expressible as a linear combination of the vectors that "belong" to the input assignment u (as defined above). Our definition in terms of polynomials is equivalent [22]: just think of each vector as the evaluation of the corresponding polynomial on a fixed set of points.

QSPs are a natural extension of (linear) SPs. An SP accepts an input u if and only if the target polynomial can be written as an affine linear combination of polynomials that "belong" to u. A QSP accepts an input u if and only if the divisor polynomial divides a *product* of two affine linear combinations of polynomials that "belong" to u, where "product" is polynomial multiplication.

Unlike SPs, QSPs can efficiently compute any function in P, and the "canonical QSP" we build has performance parameters that yield faster SNARKs, as stated in the two theorems below.

Theorem 1. *(Informal) For any Boolean circuit C with s gates and any field F of size at least $d = O(s)$, there is a QSP of size and degree $O(s)$ (with small constants) over F that computes C.*

Theorem 2. *(Informal) Given a circuit C with s gates, computing the polynomials $D(x)$, $\mathcal{V}$ and $\mathcal{W}$ of our "canonical" QSP, which computes C, takes $O(s)$ work ($O(s)$ F operations). Given $u \in \{0,1\}^n$ for which $C(u) = 1$, computing suitable tuples $(a_1, \ldots, a_m)$, $(b_1, \ldots, b_m) \in \{0,1\}^m$ that satisfy Equation 1 takes $O(s)$ work. Given $(a_1, \ldots, a_m)$, computing $v(x) = v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)$ takes $O(s)$ work. (Similarly for $w(x)$.) Computing the quotient $h(x) = v(x) \cdot w(x)/D(x)$ takes $\tilde{O}(s)$ work.*

We obtain such performance by exploiting the sparseness of the polynomials $v_k(x)$'s and $w_k(x)$'s in our canonical QSP. In particular, they behave similarly to Lagrange basis polynomials $\ell_j(x) = \prod_{i \neq j}(x - r_i)/(r_j - r_i)$ in that they each evaluate to 0 at almost all roots of $D(x)$, which is a product of linear terms. This makes it easy to compute $v(x)$ and $w(x)$ in linear time by representing them by their evaluation at these roots. Computing $h(x)$ in purely linear, versus quasi-linear, time remains an open problem.

1.2 From QSPs to SNARKs, NIZKs, & Verifiable Computing

We use QSPs to build SNARKs and NIZKs in the CRS model [9, 10].

SNARKs. Our SNARK for f uses a CRS in which the QSP polynomials (e.g., $\{v_k(x)\}$) are represented by terms $g^{v_k(\sigma)}$ (etc.), where g is a generator of a bilinear group [12], and $\sigma \in F$ is secret. The CRS size is linear in the circuit size of f. To oversimplify, to compute a SNARK, the prover uses its satisfying input to compute tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$, and then uses them and the CRS to compute $g_v = g^{v(\sigma)}$, $g_w = g^{w(\sigma)}$, $g_h = g^{h(\sigma)}$ for $v(x)$, $w(x)$, $h(x)$ as defined in Theorem 2. The verifier confirms that $e(g_v, g_w) = e(g_h, g^{t(\sigma)})$, where e is the bilinear map. (The

actual scheme is more complicated – see Section 3.2.) For security, we require a non-falsifiable "knowledge" assumption which, as noted above, is necessary [1, 23].

NIZK. It is straightforward to randomize our public-verifier SNARK to make it statistical zero knowledge and obtain a non-interactive zero-knowledge (NIZK) argument [9, 10]. Details are in Section 3.3.

Verifiable Computation. In the full version [22], we use our QSP-based SNARK to achieve a very efficient scheme for public verifiable computation [21, 39].

Remark on Efficiency and Adaptivity. In the description above, the CRS (the QSP polynomials) depend on a particular language or relation. We can achieve an "adaptive" solution (where first the CRS is fixed, and then the language or relation is selected) by applying our QSP construction to the universal circuit, at the cost of expanding the circuit by a logarithmic factor, yielding quasi-linear complexity for CRS size and prover computation.

1.3 Comparisons to Other Work on Succinct Arguments

PCP-based Protocols. Ishai, Kushilevitz and Ostrovsky [30] were perhaps the first to seriously investigate how to tweak PCP-techniques to yield the best possible succinct arguments. However in their solution the prover's computation (and also the verifier's computation in a pre-processing step) is *quadratic* in the size of the classical proof.

Very recently, Ben-Sasson et al. present a new PCP scheme with quasi-linear complexity for the prover and the CRS [7]. Our direct construction of QSPs yields better asymptotic performance, even before these PCPs are converted into SNARKs.

Groth-like schemes. Groth et al. [28, 29] previously constructed NIZKs over bilinear groups with various attractive properties, but with size linear in the circuit. More recently, Groth essentially found a way to compress the proof into a constant number of group elements [27] (still higher than ours – 42 group elements versus 7 for ours). Security relies on a non-falsifiable "knowledge of exponent" assumption, similar to the one we use.

The main drawbacks of Groth's succinct NIZK are the prover complexity and the CRS size, which are both *quadratic* in the circuit size. Lipmaa [33] showed how to reduce the size of the CRS in Groth's construction from quadratic to quasi-linear in the circuit size, but prover complexity remains quadratic.

2 Quadratic Span Programs (QSPs)

Above, we defined Quadratic Span Programs (QSPs) in a manner that is superficially similar to that of span programs (SPs). The crucial difference is that QSPs can compute any efficiently computable function. We demonstrate this via an explicit construction of a QSP for any circuit C.[3] The construction uses two components: a *gate checker* and a *wire checker*.

2.1 A Gate Checker

While we do not know how to efficiently construct SPs for arbitrary functions $f \in \mathrm{P}$, we *can* always efficiently construct an SP for a function *related* to f, called the *gate checker function* for f, which ensures that a set of wire values is consistent with the gates in a circuit for f.

Definition 2 (Gate Checker Function). *Let $f : \{0,1\}^n \to \{0,1\}$ be a function whose Boolean circuit C has s gates. Let $N = n + s$ – the total number of wires in C (wires that fan out are considered one wire). Define $\phi : \{0,1\}^N \to \{0,1\}$ to be a function that outputs '1' iff the input is a valid assignment of C's wires with output wire set to '1'. We say that ϕ is the* gate checker function *for f.*

An SP for the gate checker function ϕ does *not*, however, compute the function f; such an SP has labeled (non-free) polynomials even for the interior wires of C, whereas an SP for f is only permitted to have labeled polynomials for C's input wires. If we simply move the polynomials for the interior wires to the "free" set, then we might introduce additional valid linear combinations that do not satisfy C; in particular these linear combinations could use polynomials that correspond to conflicting assignments (both '0' and '1') for some interior wire in C.

What we prove however, is that these conflicting assignments are the only possible problem we introduce by moving the polynomials for the interior wires to the "free" set. In other words, if we restrict the linear combination to use polynomials associated with *at most* one value per wire, then the SP for ϕ can also be used to compute the function f. The following lemma formalizes the property.

Lemma 1. *Let $S = (\{v_0(x), \ldots, v_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i \in [N], j \in \{0,1\}} \mathcal{I}_{ij})$ be an SP that computes the gate checker function ϕ of f. Then, for all $u \in \{0,1\}^n$, the following is true iff $f(u) = 1$: there exists a tuple $(a_1, \ldots, a_m)$ satisfying the following constraints:*

[3] The full version of the paper [22] gives a formal reduction from circuit SAT to a QSP satisfiability problem, hence proving that QSP SAT is NP complete.

– ***Target in Span:*** $v_0(x) = \sum_k a_k \cdot v_k(x)$.
– ***Correct Inputs:*** *For all* $k \in \cup_{i=1}^{n} I_{i\overline{u}_i}$, *we have* $a_k = 0$.
– ***No Double Assignments:*** *For all* $i \in \{n+1, \ldots, N\}$ *and all* $k_1 \in I_{i0}$ *and* $k_2 \in I_{i1}$, *at most one of* a_{k_1}, a_{k_2} *is nonzero.*

In particular, if $f(u) \neq 1$, *then a linear combination that satisfies the first and second constraints must violate the third – i.e., must make a "double assignment" of some wire* $i \in \{n+1, \ldots, N\}$.

Proof. (Lemma 1) If $f(u) = 1$, then we can assign the wires of C validly with the output wire set to 1. Therefore, we can extend $u \in \{0,1\}^n$ to an input $u' \in \{0,1\}^N$ that satisfies ϕ. Since u' satisfies ϕ, there is a linear combination $(a_1, \ldots, a_m)$ such that $v_0(x) = \sum_k a_k \cdot v_k(x)$ and $a_k = 0$ for all $k \in \cup_{i=1}^{n} I_{i\overline{u'}_i}$, thus satisfying the constraints listed in the lemma.

Conversely, suppose that $(a_1, \ldots, a_m)$ satisfies the constraints. Then, since S computes ϕ, there is an extension $u' \in \{0,1\}^N$ of $u \in \{0,1\}^n$ such that $\phi(u') = 1$ and such that u' "agrees" with the tuple $(a_1, \ldots, a_m)$ in the sense that $a_k = 0$ for all $k \in I_{i\overline{u'}_i}$, $i \in [N]$. Since $\phi(u') = 1$ where u' is an extension of u, and since ϕ tests the satisfaction of f's Boolean circuit, we must have $f(u) = 1$. ■

Looking ahead, our construction will use the span program for ϕ to obtain efficient proofs about the correct evaluation of f. The second component of our construction, the wire checker, will efficiently verify that the **No Double Assignments** property holds.

2.2 A Wire Checker

To prevent double wire assignments, we introduce some additional polynomials in the form of a *wire checker*, defined as follows.[4]

Definition 3 (Aggregate Wire Checker). *Let* $\mathcal{I} = \cup_{i \in [N], j \in \{0,1\}} \mathcal{I}_{ij}$ *be a partition of* $[m]$. *An aggregate wire checker for* $\mathcal{I}$ *consists of polynomials* $D(x)$, $\mathcal{V} = \{v_k(x) : k \in \mathcal{I}\}$ *and* $\mathcal{W} = \{w_k(x) : k \in \mathcal{I}\}$ *such that*

$$D(x) \quad \text{divides} \quad \Big(\sum_{k \in \mathcal{I}} a_k \cdot v_k(x)\Big) \cdot \Big(\sum_{k \in \mathcal{I}} b_k \cdot w_k(x)\Big) \tag{2}$$

if $\{a_k\}$ *and* $\{b_k\}$ *indicate consistent bit assignments of all* N *bits (i.e., for each* $i \in [N]$, *for some bit* B_i, $a_k = b_k = 0$ *for all* $k \in \mathcal{I}_{i\bar{B}_i}$*), but not if* $\{a_k\}$ *and* $\{b_k\}$ *indicate inconsistent bit assignments of any of the* N *bits in the following sense: For some* $i \in [N]$,

[4] While Definition 3 resembles a QSP, a wire checker is *not*, on its own, a QSP.

– *There exist* $k_a \in \mathcal{I}_{i0}$ *and* $k'_a \in \mathcal{I}_{i1}$ *and* $k_b \in \mathcal{I}_{i0} \cup I_{i1}$ *such that* $a_{k_a} \neq 0$, $a_{k'_a} \neq 0$ *and* $b_{k_b} \neq 0$, *or*
– *There exist* $k_a \in \mathcal{I}_{i0} \cup I_{i1}$ *and* $k_b \in \mathcal{I}_{i0}$ *and* $k'_b \in \mathcal{I}_{i1}$ *such that* $a_{k_a} \neq 0$, $b_{k_b} \neq 0$ *and* $b_{k'_b} \neq 0$.

The size of the wire checker is $|\mathcal{I}|$, *and the degree is* $\deg(D(x))$.

To construct an aggregate wire checker, we first construct a checker for a single wire. Let $\mathcal{I}_0 = \{1, \ldots, L_0\}$, $\mathcal{I}_1 = \{L_0 + 1, \ldots, L_0 + L_1\}$, and $\mathcal{I} = \mathcal{I}_0 \cup \mathcal{I}_1$ be the indices associated with the wire.

Construction of a Wire Checker.

1. Let $L_{max} = \max(L_0, L_1)$. For $L' = 3L_{max} - 2$, select distinct roots $\mathcal{R}^{(0)} = \{r_1^{(0)}, \ldots, r_{L'}^{(0)}\}$ and $\mathcal{R}^{(1)} = \{r_1^{(1)}, \ldots, r_{L'}^{(1)}\}$ from F. Set $\mathcal{R} = \mathcal{R}^{(0)} \cup \mathcal{R}^{(1)}$. Set $D(x) = \prod_{r \in \mathcal{R}}(x - r)$.
2. Interpolate the polynomials in $\{v_k(x)\}$ and $\{w_k(x)\}$ to have degree $(L' + L_0 - 1)$ if $k \in \mathcal{I}_0$ and $(L' + L_1 - 1)$ if $k \in \mathcal{I}_1$, and to satisfy:
   (a) For $k \in \mathcal{I}_0$, $v_k(r) = 0$ for all $r \in \mathcal{R}^{(0)} \cup \{r_1^{(1)}, \ldots, r_{L_0}^{(1)}\}$ except $v_k(r_k^{(1)}) = 1$, and $w_k(r) = 0$ for all $r \in \mathcal{R}^{(1)} \cup \{r_1^{(0)}, \ldots, r_{L_0}^{(0)}\}$ except $w_k(r_k^{(0)}) = 1$.
   (b) For $k \in \mathcal{I}_1$, $v_k(r) = 0$ for all $r \in \mathcal{R}^{(1)} \cup \{r_1^{(0)}, \ldots, r_{L_1}^{(0)}\}$ except $v_k(r_{k-L_0}^{(0)}) = 1$, and $w_k(r) = 0$ for all $r \in \mathcal{R}^{(0)} \cup \{r_1^{(1)}, \ldots, r_{L_1}^{(1)}\}$ except $w_k(r_{k-L_0}^{(1)}) = 1$.

Lemma 2. *The construction above is a wire checker.*

Proof. (Lemma 2) Clearly, $D(x)$ divides the product in Equation 2 – i.e., $(\sum_{k \in \mathcal{I}} a_k \cdot v_k(r)) \cdot (\sum_{k \in \mathcal{I}} b_k \cdot w_k(r)) = 0$ for all $r \in \mathcal{R}$ – if $\{a_k\}$, $\{b_k\}$ indicate consistent assignments.

If $\{a_k\}$ indicates a double assignment and $\{b_k\}$ is nonzero, then $\sum_{k \in \mathcal{I}_0} a_k \cdot v_k(x)$ has at most $L_0 - 1$ roots in $\mathcal{R}^{(1)}$, since it is nonzero of degree $L' + L_0 - 1$ and already has $\mathcal{R}^{(0)}$ as roots. A similar analysis shows that $\sum_{k \in \mathcal{I}_1} a_k \cdot v_k(x)$ has at most $L_1 - 1$ roots in $\mathcal{R}^{(0)}$. Note that $\sum_{k \in \mathcal{I}} a_k \cdot v_k(x)$ has exactly the same roots in $\mathcal{R}^{(1)}$ that $\sum_{k \in \mathcal{I}_0} a_k \cdot v_k(x)$ does, since the other part of the sum – namely, $\sum_{k \in \mathcal{I}_1} a_k \cdot v_k(x)$ – has everything in $\mathcal{R}^{(1)}$ as a root. Similarly, $\sum_{k \in \mathcal{I}} a_k \cdot v_k(x)$ has exactly the same roots in $\mathcal{R}^{(0)}$ that $\sum_{k \in \mathcal{I}_1} a_k \cdot v_k(x)$ does. So, $\sum_{k \in \mathcal{I}} a_k \cdot v_k(x)$ has at most $L_0 + L_1 - 2 \leq 2L_{max} - 2$ roots in $\mathcal{R}$. Since $\sum_{k \in \mathcal{I}} b_k \cdot w_k(x)$ is nonzero and degree-$(L' + L_{max} - 1)$, it has at most $L' + L_{max} - 1$ roots in $\mathcal{R}$. So, the overall product has at most $L' + 3L_{max} - 3 < 2L'$ roots, and is therefore not divisible by $D(x)$. ∎

Using the Chinese Remainder Theorem, we compose the wire checkers for individual wires into an aggregate wire checker for the whole circuit.

Construction of an Aggregate Wire Checker.

1. *Generate all of the roots and the divisor polynomial.* For each wire $i \in [N]$, select distinct roots for $\mathcal{R}^{(i0)}$ and $\mathcal{R}^{(i1)}$ from F as in the single-wire checker. Note that the roots are distinct across the i's as well. Set $\mathcal{R} = \cup_i \mathcal{R}^{(i0)} \cup \mathcal{R}^{(i1)}$. Set the aggregate wire checker's divisor polynomial to $D(x) = \prod_i D_i(x) = \prod_{r \in \mathcal{R}}(x - r)$.
2. *Generate polynomials for the individual wire checkers.* For each wire $i \in [N]$, construct the sets of polynomials $\mathcal{V}^{(i)}$ and $\mathcal{W}^{(i)}$ as in the single-wire checker.
3. *Compose individual wire checkers via CRT.* For $i \in [N]$, for $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$, interpolate $v_k(x)$ to be of degree at most $\deg(D(x)) - 1$ and satisfy $v_k(x) = v_k^{(i)}(x) \bmod D_i(x)$ and $v_k(x) = 0 \bmod D(x)/D_i(x)$. Analogously for $w_k(x)$. Set $\mathcal{V} = \{v_k(x)\}$ and $\mathcal{W} = \{w_k(x)\}$.

Lemma 3. *The above construction is an aggregate wire checker.*

Proof. (Lemma 3) If $\{a_k\}$, $\{b_k\}$ indicate consistent assignments, then they are consistent on the i-th bit for k restricted to $\mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. Hence, $D_i(x)$ divides the product in Eqn.2 when the summations are restricted to $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. Since $v_k(x)$ and $w_k(x)$ are divisible by $D_i(x)$ for all $k \notin \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$, the overall (unrestricted) product in Eqn.2 is divisible by $D_i(x)$. Since this holds for all i, the product is divisible by $D(x)$.

If, for some i, $\{a_k\}$ indicates a double assignment of the i-th bit and $\{b_k\}$ is nonzero over $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$, then, by Lemma 2, $D_i(x)$ does not divide the product in Eqn.2 when the summations are restricted to $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. As above, $D_i(x)$ divides everything else, and thus the overall product in Eqn.2 is not divisible by $D_i(x)$, and thus not divisible by $D(x)$. ■

2.3 Conscientious Span Programs

Notice that the aggregate wire checker definition above enforces a slightly weaker condition than forbidding double assignments: it states that double assigning a wire with the $\{a_k\}$ (i.e., using non-zero a_k values from both $\mathcal{I}_{i0}$ *and* $\mathcal{I}_{i1}$) is forbidden, unless the $\{b_k\}$ indicate a *non*-assignment of that wire – i.e., all the corresponding $b_k = 0$ (and vice versa for a double assignment in the $\{b_k\}$).

To compensate for the weakness of the wire checker, we require the SP being checked to be *conscientious*, which guarantees that every satisfying

linear combination uses *at least* one polynomial from the sets associated with its input. In our canonical QSP, we will use the wire checker above on *two* instances of a conscientious SP for ϕ. Conscientiousness guarantees that each instance includes a non-zero coefficient for each wire used in the satisfying assignment, and hence the wire checker will always catch double assignments in either instance.

More formally, we define a conscientious SP as follows:

Definition 4. *Let* $S = (\{v_0(x), \ldots, v_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n],j\in\{0,1\}} \mathcal{I}_{ij})$ *be an SP. We say that* S *is a* conscientious *SP for* $f : \{0,1\}^n \to \{0,1\}$ *if, for any tuple* $(a_1, \ldots, a_m)$ *that satisfies the usual SP requirements that* $f(u) = 1$ *for* $u \in \{0,1\}^n$ *iff (1)* $v_0(x) = \sum_k a_k \cdot v_k(x)$ *and (2) for all* $k \in \cup_{i=1}^n I_{i\bar{u}_i}$ *we have* $a_k = 0$*, we also have the property that for all* $i \in [n]$*, there exists* $k \in I_{iu_i}$ *such that* $a_k \neq 0$*. Let* m *be the size of the SP and* $\deg(v_0(x)) + 1$ *be the degree of the SP.*

To construct a conscientious SP for ϕ, we first build a conscientious SP for a single NAND gate.

Lemma 4. *There is a degree-9 conscientious SP for NAND of size 12.*

Proof. (Lemma 4) Choose a set of 9 distinct roots in F to get $\mathcal{R} = (r_0, r_{l0}, r'_{l0}, r_{l1}, r'_{l1}, r_{r0}, r'_{r0}, r_{r1}, r'_{r1})$. Define 9 "linearly independent" polynomials $\{v_0(x), v_{l0}(x), v'_{l0}(x), v_{l1}(x), v'_{l1}(x), v_{r0}(x), v'_{r0}(x), v_{r1}(x), v'_{r1}(x)\}$ to be the corresponding Lagrange basis polynomials for $\mathcal{R}$; that is, they are the degree-8 polynomials obtained by interpolating such that $\forall r \in \mathcal{R}$, $v_0(r) = 0$, except that $v_0(r_0) = 1$; $v_{l0}(r) = 0$, except that $v_{l0}(r_{l0}) = 1$, and so on. We will use the convention that the pair of polynomials $\mathcal{V}_{l0} = (v_{l0}(x), v'_{l0}(x))$ belongs to the assignment of 0 to the left wire, etc.

Set $v_{o0}(x) = v_0(x) - v_{l1}(x) - v_{r1}(x)$ and $\mathcal{V}_{o0} = \{v_{o0}(x)\}$, so that one can express $v_0(x)$ as a linear combination of polynomials in $\mathcal{V}_{l1} \cup \mathcal{V}_{r1} \cup \mathcal{V}_{o0}$.

Set $v_{o1}(x) = v_0(x) - v_{l0}(x) - v_{r0}(x)$, $v'_{o1}(x) = v_0(x) - v'_{l0}(x) - v'_{r1}(x)$, and $v''_{o1}(x) = v_0(x) - v'_{l1}(x) - v'_{r0}(x)$, and $\mathcal{V}_{o1} = \{v_{o1}(x), v'_{o1}(x), v''_{o1}(x)\}$, so that one can express $v_0(x)$ as a linear combination of polynomials associated to the other satisfying gate assignments.

That the above polynomials define a conscientious SP for NAND of the claimed size and degree follows by inspection. The details are elaborated in the full version. ■

To obtain a conscientious SP for an entire circuit, we build a conscientious SP for each gate, using a distinct set of roots $\mathcal{R}_i$ for each SP, and then compose the gate SPs together using the Chinese Remainder Theorem, just as we did when building the aggregate wire checker.

Lemma 5. *Suppose a circuit C consists of s Boolean gates from some set Γ – e.g, $\Gamma = \{\text{NAND}\}$. Suppose that, for each gate $g \in \Gamma$, there is a conscientious SP of size m' and degree d' that computes whether its input is a satisfying assignment of g's input/output wires. Then there is a conscientious SP S of size $m = s \cdot m'$ and degree $d = s \cdot d'$ that computes the gate checker function ϕ for C. S is a straightforward composition of SPs $\{S_g\}$ for the individual gates g of C.*

Intuition. The proof is constructive. For each gate g, build an SP $S^{(g)}$ following Lemma 4, obtaining from each a unique set of roots $\mathcal{R}^{(g)}$ and polynomials $\{v_0^{(g)}(x)\} \cup \mathcal{V}^{(g)}$. Let $\mathcal{R} = \cup_g \mathcal{R}^{(g)}$. Let $v_0(x)$ be a polynomial such that $v_0(r) = v_0^{(g)}(r)$ for all $r \in \mathcal{R}^{(g)}$ and all gates g in the circuit. For each gate g, extend g's polynomials such that for all $v(x) \in \mathcal{V}^{(g)}$, and $r \in \mathcal{R}/\mathcal{R}^{(g)}$, $v(r) = 0$. The aggregate SP's set of polynomials $\mathcal{V}$ will consist of $v_0(x)$ along with all of the extended polynomials. Since the roots used in each SP are unique across all SPs, this composition preserves all of the local linearity relationships created by Lemma 4; it also does not introduce any new relationships, since the unique roots prevent "interactions" across the gate SPs. See the full version of the paper [22] for the full proof.

2.4 The Canonical Quadratic Span Program

We now describe how to take any polynomial-time computable function f, and construct a polynomial-size QSP that computes f. The construction uses the Chinese Remainder Theorem (CRT) to merge the two components above, the gate checker and the wire checker, so that the quadratic test (Eq. 1) checks both at once. The wire checker's guarantee of no double assignments relies on the fact that the SP for the gate checker is conscientious, and hence must use at least one polynomial for each wire to arrive at a satisfying linear combination. Thus, we can conclude that the wire values are consistent with the circuit's gates, and that no wire is set to both 0 and 1.

More specifically, we build two copies of the conscientious SP for the gate checker, ensuring that all of the roots used are distinct. One copy will become the $\mathcal{V}$ polynomials in the QSP, while the other copy will become the $\mathcal{W}$ polynomials. We then construct the polynomials for the aggregate wire checker described above, using a third set of distinct roots. Since all of the divisor polynomials from the different components have different roots, they are relatively prime. Hence, we can use the CRT to define the final QSP polynomials so that they match the value of the constituent polynomials from each component.

The Canonical QSP: $Q_{can,f}$.

1. Take as input the Boolean circuit C for $f : \{0,1\}^n \to \{0,1\}$, which has s gates.
2. Using disjoint sets of roots $\mathcal{R}^{(\mathcal{V})}$ and $\mathcal{R}^{(\mathcal{W})}$, construct two instances of the conscientious gate checker SP for C – namely, $S^{(\mathcal{V})} = (\hat{\mathcal{V}} = \{\hat{v}_0(x), \ldots, \hat{v}_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled})$ and $S^{(\mathcal{W})} = (\hat{\mathcal{W}} = \{\hat{w}_0(x), \ldots, \hat{w}_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled})$.
3. Define $\hat{D}^{(\mathcal{V})}(x) = \prod_{r \in \mathcal{R}^{(\mathcal{V})}}(x - r)$ and $\hat{D}^{(\mathcal{W})}(x) = \prod_{r \in \mathcal{R}^{(\mathcal{W})}}(x - r)$. Note that because we use distinct roots for each incarnation, the resulting divisor polynomials $\hat{D}^{(\mathcal{V})}(x)$ and $\hat{D}^{(\mathcal{W})}(x)$ are relatively prime.
4. Using disjoint sets of roots $\mathcal{R} = \{\mathcal{R}^{(i0)}, \mathcal{R}^{(i1)} : i \in [N]\}$ and the partition of $\mathcal{I}_{labeled}$, construct the aggregate wire checker from Lemma 3, which consists of the following polynomials: $D'(x) = \prod_{r \in \mathcal{R}}(x - r)$, $\mathcal{V}' = \{v'_1(x), \ldots, v'_m(x)\}$ and $\mathcal{W}' = \{w'_1(x), \ldots, w'_m(x)\}$.
5. Define $D(x) = \hat{D}^{(\mathcal{V})}(x) \cdot \hat{D}^{(\mathcal{W})}(x) \cdot D'(x)$.
6. Finally, define $\mathcal{V} = \{v_0(x), \ldots, v_m(x)\}$ and $\mathcal{W} = \{w_0(x), \ldots, w_m(x)\}$ using the CRT to interpolate $v_k(x)$ and $w_k(x)$ as follows:

$$v_k(x) = \begin{cases} \hat{v}_k(x) \bmod \hat{D}^{(\mathcal{V})}(x) \\ v'_k(x) \bmod D'(x) \\ 1 \bmod \hat{D}^{(\mathcal{W})}(x) \text{ if } k = 0 \\ 0 \bmod \hat{D}^{(\mathcal{W})}(x) \text{ if } k \neq 0 \end{cases} \quad w_k(x) = \begin{cases} \hat{w}_k(x) \bmod \hat{D}^{(\mathcal{W})}(x) \\ w'_k(x) \bmod D'(x) \\ 1 \bmod \hat{D}^{(\mathcal{V})}(x) \text{ if } k = 0 \\ 0 \bmod \hat{D}^{(\mathcal{V})}(x) \text{ if } k \neq 0 \end{cases}$$

7. Output $Q_{can,f} = (\mathcal{V}, \mathcal{W}, D(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i \in [n], j \in \{0,1\}} \mathcal{I}_{ij})$, where the labeled indices $\cup_{i \in [n+1,N]} \mathcal{I}_{ij}$ from the gate checker SP for C have been moved to $\mathcal{I}_{free}$.

The proof of the following theorem is in the full version [22].

Theorem 3. *For any Boolean circuit C with n inputs, s gates, and $N = n + s$ total wire values, the canonical QSP computes C.*

2.5 Performance and Technical Issues

By Lemmas 4 and 5, given a function f whose Boolean circuit has s (NAND) gates, we have a conscientious SP of size $12s$ and degree $9s$ for f's gate-checker function. However, for performance and technical reasons, we use a larger conscientious SP of size $36s$ and degree $27s$.

The first reason we use a larger SP is that we transform f's Boolean circuit to one with fan-out two (except that one "dummy" input, set to

'1', may feed into multiple gates). The resulting circuit may be larger by a constant factor. We reduce fan-out to two before applying the SP composition lemma (Lemma 5) because we want the evaluation vectors $\{(v_k(r_1), \ldots, v_k(r_d)), (w_k(r_1), \ldots, w_k(r_d)) : k \in [m], r \in \mathcal{R}\}$ of our QSP to be *sparse* – i.e., to have only constant nonzero support. Sparseness allows us, for example, to compute $v(x) = v_0(x) + \sum a_k \cdot v_k(x)$ very quickly in evaluation representation, in time linear in the degree of the QSP.

The second reason is that we obtain a *strong QSP*.

Definition 5 (Strong QSP). *A QSP $Q = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij})$ is a strong QSP if $|\mathcal{I}_{ij}| = 1$ for all $i \in [n], j \in \{0,1\}$ and the QSP divisibility requirement (Eq.1) holds only if $\{a_k\}$, $\{b_k\}$ are "unequivocally" bound to some input $u \in \{0,1\}^n$ – in particular, $a_k = 1 = b_k$ for all $\{k = \mathcal{I}_{iu_i}\}$ and $a_k = 0 = b_k$ for all $\{k = \mathcal{I}_{i\bar{u}_i}\}$.*

In a strong QSP, the labeled sets are singletons, and the QSP can be satisfied only by applying an unequivocal 0/1 linear combination to the labeled vectors. Ultimately, this property helps improve the performance of our cryptographic constructions for NIZKs and verifiable computation, since a verifier who knows part of the circuit input (e.g., the statement u portion of the input to a relation) will be able to "predict" the portion of the QSP linear combination that corresponds to u (and therefore this portion does not need to be "sent" by the prover).

When it is applied to the partition $\mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij}$ of the SP for the gate checker function, the size of the aggregate wire checker is $|\mathcal{I}_{labeled}| \leq 24s$ and the degree is $76s$. (See full version [22] for details.) Since the QSP has two SPs and one aggregate wire checker, and since composing the SPs with the wire checker does not increase the size, the QSP has size $36s$ and degree $130s$.

3 Overview of Cryptographic Constructions and Security

We build SNARKs and NIZKs in the common reference string (CRS) model [9, 10] for relations $R(u, w)$ with n'-bit statements and $(n - n')$-bit witnesses. We apply our QSPs for n-bit inputs to the circuit computing R.

Groth's construction [27] specifically targets the circuit SAT relation; in particular, he takes u to be a circuit that can be chosen adaptively and uses $R(u, w) = u(w)$. The CRS size and prover computation grow quadratically with $|u|$. The verifier computation is $O(|u|)$, but it can be reduced to $O(1)$ in an amortized sense with u-dependent pre-processing. To compare directly with Groth, we can handle u being an adaptively-chosen circuit by constructing R from a universal circuit. In this case, the size of

the circuit computing R may be larger than $|u|$ by a logarithmic factor, which correspondingly increases the CRS size and prover computation to $\tilde{O}(|u|)$. The verifier computation is $O(|u|)$, but it can be reduced to $O(1)$ in an amortized sense just as in Groth. If u, or any part of u, can be chosen non-adaptively, our scheme becomes more efficient.

We present our constructions with their proof intuition, deferring the formal proofs to the full version [22].

3.1 Definitions

First, we define a SNARK for a Prover P who holds a witness w which he can use to convince a Verifier V of a statement u.

Definition 6 (SNARK). *We say that $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is a succinct non-interactive argument of knowledge (*SNARK*) with security parameter κ for an* NP *language L with a corresponding* NP *relation R with n'-bit statements and $(n-n')$-bit witnesses , if it satisfies the following properties:*

Perfect Completeness: *For all $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c|c} \mathsf{V}(\mathsf{priv}, u, \pi) = 1 & (\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa) \\ \textit{if } (u, w) \in R & (u, w) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w) \end{array}\right] = 1,$$

where $\mathsf{P}(\mathsf{crs}, u, w)$ runs in time $\mathrm{poly}(\kappa, n)$.

Soundness: *For all efficient $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c|c} \mathsf{V}(\mathsf{priv}, u, \pi) = 1 & (\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa) \\ u \notin L & (u, \pi) \leftarrow \mathcal{A}(1^\kappa, \mathsf{crs}) \end{array}\right] = \mathtt{negl}(\kappa).$$

Succinctness: *The proof length is $|\pi| = \mathrm{poly}(\kappa)$.*

Extraction: *For any poly-size prover P^*, there exists a poly-size extractor $\mathcal{E}_{P^*}$, such that for any auxiliary information $z \in \{0,1\}^\kappa$, the following holds*

$$\Pr\left[\begin{array}{c|c} \mathsf{V}(\mathsf{priv}, u, \pi) = 1 & (\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa) \\ (u, w) \notin \mathcal{R} & (u, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}, z) \\ & w \leftarrow \mathcal{E}_{P^*}(\mathsf{crs}, z) \end{array}\right] = \mathtt{negl}(\kappa).$$

We omit the standard definition of NIZKs. Note that, to build a NIZK, it suffices to build (as we do) a SNARK that is statistical zero-knowledge.

3.2 Our SNARK Construction

We can create a SNARK for an NP relation $R = \{(u, w)\}$ with n'-bit statements and $(n - n')$-bit witnesses by building a canonical QSP for the function f such that $f(u, w) = 1$ iff $(u, w) \in R$. At a high-level, the prover uses his inputs to evaluate the circuit for f, hence obtaining linear combinations for the QSP that satisfy Eq.1. He uses these combinations to compute $v(x) = v_0(x) + \sum a_k \cdot v_k(x)$ (and similarly for $w(x)$), and convinces the verifier that the QSP's quadratic property holds (Eq.1), which implies $f(u, w) = 1$, by calculating $h(x)$ such that $h(x) \cdot D(x) = v(x)w(x)$.

To protect against malicious provers, all of the calculations described above are performed over *encoded* values. Specifically, the CRS holds an encoding of the evaluation of each polynomial (e.g., the $\{v_k(x)\}$) at a secret point σ. The encoding permits homomorphic operations, which allow the prover to calculate $v(\sigma)$, $w(\sigma)$, and $h(\sigma)$ inside the encoding. The encoding also permits a quadratic equality check so that the verifier can check that Equation 1 holds.

An encoding scheme $\mathcal{E}$ has two algorithms (Setup, E), where Setup takes the security parameter and generates parameters for the scheme, and E (possibly randomized) produces an encoding for an element. Our preferred encoding is exponentiation within a bilinear group: $E(v_k(\sigma)) = g^{v_k(\sigma)}$, in which case, the quadratic equality check is performed via a pairing. One may also use an additively homomorphic encryption scheme, e.g., Paillier[5]: $E(v_k(\sigma)) = \mathbf{Enc}_{pk}(v_k(\sigma))$. In this case, the verifier needs a secret key sk to remove the encoding and perform the quadratic check, and hence the SNARK is designated-verifier.

As a final note, to ensure the prover uses circuit inputs matching u, the verifier calculates the portion of $v(\sigma)$ that corresponds to u independently, leaving the portion of $v(\sigma)$ that corresponds to the witness to the prover.

To base the security of our scheme on an existing knowledge of exponent assumption [27], we add terms to the CRS of the form $E(\alpha\sigma^i)$, $E(\alpha v_k(\sigma))$, $E(\alpha w_k(\sigma))$, $E(\beta_v v_k(\sigma))$, $E(\beta_w w_k(\sigma))$, and extend the proof with relations between these terms and those in the basic proof (see Section 3.4).

CRS generation Gen**:** On input security parameter κ, construct a common random string $CRS = (\mathsf{crs}_P, \mathsf{crs}_V)$. Let f be the function checking the relation $R(u, w)$ and let $Q_f = (\mathcal{V}, \mathcal{W}, D(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} =$

[5] Technically, our constructions apply only where the encoding space is a field, and the plaintext space of Paillier is a ring, not a field. However, it would be easy to extend our results to Paillier, using the fact that one is unlikely to encounter encodings of nontrivial zero divisors in $\mathbb{Z}_N$ unless one is able to factor N.

$\cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}, \mathcal{I} = \mathcal{I}_{free} \cup \mathcal{I}_{labeled}$) be a QSP of size m and degree d for the functionality f[6]. Let $\mathcal{I}_{in} = \cup_{i=1}^{n'}\mathcal{I}_{ij}$ and $\mathcal{I}_{mid} = \mathcal{I} \setminus \mathcal{I}_{in}$. Generate public and private parameters (pk, sk) for the encoding scheme E. Generate uniformly at random $\alpha, \sigma, \beta_v, \beta_w, \gamma \leftarrow F^*$ and set the output:

$$\begin{aligned}
\mathsf{crs}_P = \big(&pk, Q_f, n', \{E(\sigma^i)\}_{i\in[0,d]}, \{E(\alpha\sigma^i)\}_{i\in[0,d]},\\
&\{E(v_k(\sigma))\}_{k\in\mathcal{I}_{mid}}, \{E(w_k(\sigma))\}_{k\in\mathcal{I}},\\
&\{E(\alpha v_k(\sigma))\}_{k\in\mathcal{I}_{mid}}, \{E(\alpha w_k(\sigma))\}_{k\in\mathcal{I}},\\
&\{E(\beta_v v_k(\sigma))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(\sigma))\}_{k\in\mathcal{I}}\big)\\
\mathsf{crs}_V = \big(&pk, sk, E(1), E(\alpha), E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma),\\
&\{E(v_k(\sigma))\}_{k\in\{0\}\cup\mathcal{I}_{in}}, E(w_0(\sigma)), E(D(\sigma))\big).
\end{aligned}$$

Prove P**:** On input crs_P, statement $u \in \{0,1\}^{n'}$ and witness w, P evaluates Q_f to obtain $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ and polynomial $h(x)$ such that

$$h(x) \cdot D(x) = \left(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} b_k \cdot w_k(x)\right).$$

Let $v_{mid}(x) = \sum_{k\in\mathcal{I}_{mid}} a_k \cdot v_k(x)$ and $w(x) = \sum_{k\in\mathcal{I}} b_k \cdot w_k(x)$. Then, P uses the encoding's homomorphism to output the following proof:

$$\begin{aligned}
\pi = \Big(&E(v_{mid}(\sigma)), E(w(\sigma)), E(h(\sigma)),\\
&E(\alpha v_{mid}(\sigma)), E(\alpha w(\sigma)), E(\alpha h(\sigma)), E(\beta_v v_{mid}(\sigma) + \beta_w w(\sigma))\Big).
\end{aligned}$$

Verify V**:** On input crs_V, u, and $\pi = (\pi_{v_{mid}}, \pi_w, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \pi_y)$, V confirms that the terms are in the support of validly encoded elements. Let V_{mid}, W, H, V'_{mid}, W', H', and Y be what is encoded. V computes an encoding $E(v_{in}(\sigma))$ of $v_{in}(\sigma) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(\sigma)$. V confirms that the following equations hold:

$$\begin{gathered}
H \cdot D(\sigma) = (v_0(\sigma) + v_{in}(\sigma) + V_{mid}) \cdot (w_0(\sigma) + W),\\
V'_{mid} = \alpha V_{mid}, W' = \alpha W, H' = \alpha H, \gamma Y = (\beta_v\gamma)V_{mid} + (\beta_w\gamma)W.
\end{gathered}$$

3.3 Making the SNARK Statistical Zero-Knowledge (NIZKs)

In our NIZK construction, the prover simply *randomizes* each of the terms $v_0(\sigma) + v_{in}(\sigma) + V_{mid}$ and $w_0(\sigma) + W$ so that their product is still

[6] For example, with circuit SAT, f is a universal circuit.

divisible by $D(\sigma)$, but the terms reveal nothing more about the original values. We achieve this by adding random multiples of $D(\sigma)$ to both terms, which preserves the divisibility property for their product. We supplement crs_P with additional terms to facilitate computation of the remainder of the randomized proof. Specifically, we include: $E(D(\sigma))$, $E(\alpha D(\sigma))$, $E(\beta_v D(\sigma))$, $E(\beta_w D(\sigma))$, $E(v_0(\sigma))$, $E(\alpha v_0(\sigma))$, $E(w_0(\sigma))$ and $E(\alpha w_0(\sigma))$.

After generating a proof π as above, the prover randomizes it as follows. He picks random $\delta_{v_{mid}}, \delta_w \leftarrow F$ and outputs the following proof:

$$\pi' = \big(E(v'_{mid}(\sigma)), E(w'(\sigma)), E(h'(\sigma)),\\ E(\alpha v'_{mid}(\sigma)), E(\alpha w'(\sigma)), E(\alpha h'(\sigma)), E(\beta_v v'_{mid}(\sigma) + \beta_w w'(\sigma))\big),$$

where $v'_{mid}(x) = v_{mid}(x) + \delta_{v_{mid}} D(x)$, $w'(x) = w(x) + \delta_w D(x)$, $h'(x) = (v_0(x) + v_{in}(x) + v'_{mid}(x)) \cdot (w_0(x) + w'(x))/D(x)$, and $v_0(x)$, $v_{in}(x)$, $v_{mid}(x)$ and $w(x)$ are the values computed in the SNARK construction from the previous section. The encodings in the new proof π' can be computed efficiently from the encodings in π and the augmented crs_P.

3.4 Security

We base security on two assumptions, the q-power Diffie-Hellman (q-PDH) assumption and the q-power knowledge of exponent (q-PKE) assumption. When we instantiate our construction and the q-PDH and q-PKE assumptions with an encoding scheme $E(a) = g^a$ over a bilinear group, the q-PDH and q-PKE assumptions are virtually identical to those used by Groth in his NIZK construction [27].[7] Also, the bilinear group version of our q-PDH assumption is very similar to, but weaker than, assumptions that were used to construct hierarchical identity-based encryption and broadcast encryption schemes with short ciphertexts [11, 13].

The q-PDH assumption is a "conventional" falsifiable assumption, though still somewhat unusual in its dependence on q, which is related to the size of the circuits for the functions computed by our SNARKs.

Assumption 1 (q-PDH)**.** *Let κ be a security parameter, and $q = \mathrm{poly}(\kappa)$. The q-power Diffie-Hellman (q-PDH) assumption holds for encoding $\mathcal{E}$ if for all non-uniform probabilistic polynomial time adversaries $\mathcal{A}$ we have*

$$\Pr\left[\begin{array}{c} pk \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)\ ;\ \sigma \leftarrow F^*\ ; \\ \tau \leftarrow (pk, E(1), E(\sigma), \ldots, E(\sigma^q), E(\sigma^{q+2}), \ldots, E(\sigma^{2q}))\ ; \\ y \leftarrow \mathcal{A}(\tau)\ :\ y = E(\sigma^{q+1}) \end{array}\right] = \mathtt{negl}(\kappa).$$

[7] Our q-PDH assumption is actually weaker than his q-CPDH assumption, and our q-PKE assumption is identical to Groth's [27] and Lipmaa's [33], except that we extend the assumption to handle auxiliary inputs.

The q-PKE assumption is a non-falsifiable "knowledge" assumption, similar in spirit to (but more complicated than) early knowledge-of-exponent assumptions (KEAs) [6, 18].

Assumption 2 (q-PKE)**.** *Let κ be a security parameter, and $q = \mathrm{poly}(\kappa)$. The q-power knowledge of exponent (q-PKE) assumption holds for encoding $\mathcal{E}$ if for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$, there exists a non-uniform probabilistic polynomial time extractor $\chi_{\mathcal{A}}$ such that*

$$\Pr\left[\begin{array}{c} pk \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa)\ ;\ \alpha, \sigma \leftarrow F^*; \\ \tau \leftarrow (pk, E(1), E(\sigma), \ldots, E(\sigma^q), E(\alpha), E(\alpha\sigma), \ldots, E(\alpha\sigma^q)); \\ (E(c), E(\hat{c}); a_0, \ldots, a_q) \leftarrow (\mathcal{A}||\chi_{\mathcal{A}})(\tau, z) : \\ \hat{c} = \alpha c \wedge c \neq \sum_{k=0}^{q} a_k \sigma^k \end{array}\right] = \mathtt{negl}(\kappa)$$

for any auxiliary information $z \in \{0,1\}^{\mathrm{poly}(\kappa)}$ that is independent of α.

Next we state our main security theorem.

Theorem 4. *If the q-PDH and d-PKE assumptions hold for some $q \geq \max\{2d-1, d+2\}$, then the NIZK scheme defined in Section 3.3, instantiated with a QSP of degree d, is secure under Definition 6.*

Here, we provide some intuition, using a simpler version of our scheme, which has the following 6 element proof:

$$\pi = (E(v_{mid}(\sigma)), E(w(\sigma)), E(h(\sigma)), E(\alpha_v v_{mid}(\sigma)), E(\alpha_w w(\sigma)), E(\alpha_h h(\sigma)))\,.$$

For the version above, the intuition is that it is hard for the prover, who knows the CRS but not α_w, to output any pair $(E(W), E(W'))$ with $W' = \alpha_w W$ unless he *knows* a representation $\{b_k : k \in \mathcal{I}\}$ of W such that $W = \sum b_k w_k(\sigma)$. Knowledge of exponent assumptions (KEAs)[8] formalize this intuition: they say that for any algorithm that outputs a pair of encoded elements with ratio α_w, there is an extractor that "watches" the algorithm's computation and outputs the representation (the linear combination). In the security proof, extractors for the v, w and h terms extract out polynomials $v_{mid}(x)$, $w(x)$, $h(x)$ that are in the spans of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$, $\{w_k(x) : k \in \mathcal{I}\}$, $\{x^i : i \in [d]\}$. If the proof verifies, then $(v_0(\sigma) + v(\sigma)) \cdot (w_0(\sigma) + w(\sigma)) = h(\sigma) \cdot D(\sigma)$ for $v(x) = v_{mid}(x) + \sum_{k \in \mathcal{I}_{in}} v_k(x)$. If indeed $(v_0(x)+v(x)) \cdot (w_0(x)+w(x)) = h(x) \cdot D(x)$ as polynomials, then the soundness of our QSP implies that we have extracted a *true* proof. Otherwise, $(v_0(x)+v(x)) \cdot (w_0(x)+w(x)) - h(x) \cdot D(x)$ is a nonzero polynomial having σ as a root, which allows the simulator to solve a hard problem.

[8] KEAs [6, 18, 24] exist for Paillier/RSA [19, 24], bilinear groups [27, 33], and even lattices [34].

We modified this simpler scheme to the more complicated SNARK construction in order to base security on assumptions slightly weaker than Groth's [27]. With these assumptions, we can only extract representations of the encoded terms with respect to the power basis $\{x^i\}$ (as in [27]), not with respect to $\{v_k(x) : k \in \mathcal{I}_{mid}\}$. Thus, this extraction does not guarantee that $v_{mid}(x)$ and $w(x)$ are in their proper spans. We ensure this via the final term $E(\beta_v v_{mid}(\sigma) + \beta_w w(\sigma))$, from which the simulator can solve a hard problem if $v_{mid}(x)$ or $w(x)$ lies outside its proper span.

3.5 Efficiency

Next we state the complexities for our SNARK construction and refer the reader to the full version of the paper [22] for the proofs.

Prover's Work. The prover computation requires a number of group operations linear in the size of the QSP, aside from the computation of $h(x)$, which can be computed in $O(d \cdot \log^2(d))$ time, where d is the degree of the QSP, via multipoint evaluation and interpolation. When we construct a SNARK for circuit SAT, we use a QSP for a universal circuit, which has size $O(|C| \log |C|)$ where $|C|$ is the maximum size of the circuits in the satisfiability problem.

Verifier's Work. The verification of the SNARK is proportional to the statement size and independent of the size of the witness. We can further reduce the verification work [22] to a constant plus a hash function evaluation by applying an ordinary hash function to the statement and proving a new relation which takes the the hash output as the statement.

4 Quadratic Programs for Arithmetic Circuits

We also construct *Quadratic Arithmetic Programs* (QAPs), a natural extension of QSPs which "naturally" compute arithmetic circuits modulo the group order p. For some functions, arithmetic circuits are much smaller than their Boolean counterparts, suggesting that, in such cases, QAPs are a more attractive option. In fact, it turns out (see [38]) that QAPs are *more efficient* than QSPs, even for the Boolean case.

The full details of the QAP construction appear in the final version [22]; here we present the definition of QAPs and our main result about them.

Definition 7 (Quadratic Arithmetic Programs (QAP)). *A quadratic arithmetic program (QAP) Q over field F contains three sets of polynomials $\mathcal{V} = \{v_k(x) : k \in \{0, \ldots, m\}\}$, $\mathcal{W} = \{w_k(x) : k \in \{0, \ldots, m\}\}$, $\mathcal{Y} = \{y_k(x) : k \in \{0, \ldots, m\}\}$, and a divisor polynomial $D(x)$, all from $F[x]$.*

Let $f : F^n \longrightarrow F^{n'}$ be a function having input variables with labels $1, \ldots, n$ and output variables with labels $m-n'+1, \ldots, m$. We say that Q is a QAP that computes f if the following is true: $a_1, \ldots, a_n, a_{m-n'+1}, \ldots, a_m \in F^{n+n'}$ is a valid assignment to the input/output variables of f iff there exist $(a_{n+1}, \ldots, a_{m-n'}) \in F^{m-n-n'}$ such that $D(x)$ divides:

$$\Big(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\Big) \cdot \Big(w_0(x) + \sum_{k=1}^{m} a_k \cdot w_k(x)\Big) - \Big(y_0(x) + \sum_{k=1}^{m} a_k \cdot y_k(x)\Big).$$

The size of Q is m. The degree of Q is $\deg(D(x))$.

We prove that we can build very efficient QAPs for arbitrary circuits.

Theorem 5. *Let C be an arithmetic circuit with input from F^n that has s multiplication gates, each with fan-in two, and whose output gates are all multiplication gates. There is a QAP with size $n + s$ and degree s that computes C.*

5 Concrete Performance

We developed a system called Pinocchio [38] that includes a compiler that transforms a subset of C into either a QSP or QAP, and a set of programs for generating the CRS, creating proofs, and verifying proofs. It supports NIZK proofs and VC proofs, with both designated and public verifiers. We use a pairing-based encoding, with a 256-bit BN-curve [5] that provides 128 bits of security.

We find that QAPs outperform QSPs, and that Pinocchio significantly outperforms state-of-the-art systems [16, 40] based on PCPs [2, 25, 30].[9] For example, we measured the time for NxN matrix multiplication using random 32-bit matrix entries. For $N = 25$ to 100, Pinocchio's verifier takes 8-13ms, making it 5-7 *orders of magnitude* faster than previous work, while the worker takes 8.9-776.4s, making it $19 - 60\times$ faster.

Acknowledgments

We thank Nir Bitansky, Jens Groth, Yuval Ishai, Seny Kamara, Helger Lipmaa, and the anonymous reviewers for all of their helpful suggestions.

[9] We are not aware of any implementations based on other non-PCP-based approaches (e.g., Groth [27] or Lipmaa [33]), making direct comparisons difficult.

References

[1] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC*, 2007.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[3] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[4] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[5] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography (SAC)*, pages 319–331, 2006.

[6] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO*, pages 273–289, 2004.

[7] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete-efficiency threshold of probabilistically-checkable proofs. In *STOC*, 2013. To appear.

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

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

[10] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.

[11] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[12] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

[13] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[14] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[15] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[16] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.

[17] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In *CiE*, pages 175–185, 2008.

[18] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[19] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[20] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[21] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[22] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. Cryptology ePrint Archive, Report 2012/215, 2012.

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

[24] Kristian Gjøsteen. *Subgroup membership problems and public key cryptosystems*. PhD thesis, Norwegian University of Science and Technology, 2004.

[25] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[26] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[27] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

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

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

[30] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, 2007.

[31] Mauricio Karchmer and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

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

[33] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, volume 7194, pages 169–189, 2012.

[34] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. In *Selected Areas in Cryptography*, pages 55–72, 2011.

[35] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[36] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000. Extended abstract in FOCS '94.

[37] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[38] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2013.

[39] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, 2012.

[40] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of USENIX Security*, August 2012.

[41] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.