Quadratic Span Programs and Succinct NIZKs without PCPs

Rosario Gennaro* Craig Gentry† Bryan Parno‡ 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 construction of succinct arguments of NP-statements that are quick to construct and verify. QSPs seem well-suited for this task, perhaps even better than Probabilistically Checkable Proofs (PCPs).

In 2010, Groth constructed a NIZK argument in the common reference string (CRS) model for Circuit-SAT consisting of only 42 elements in a bilinear group. Interestingly, his argument does not (explicitly) use PCPs. But his scheme has some disadvantages – namely, the CRS size and prover computation are both *quadratic* in the circuit size. In 2011, Lipmaa reduced the CRS size to quasi-linear, but with prover computation still quadratic.

Using QSPs we construct a NIZK argument in the CRS model for Circuit-SAT consisting of just *7 group elements*. The CRS size is *linear* in the circuit size, and prover computation is *quasi-linear*, making our scheme seemingly quite practical. (The prover only needs to do a linear number of group operations; the quasi-linear computation is a multipoint evaluation and interpolation.)

Our results are complementary to those of Valiant (TCC 2008) and Bitansky et al. (2012), who use "bootstrapping" (recursive composition) of arguments to reduce CRS size and prover and verifier computation. QSPs also provide a crisp mathematical abstraction of some of the techniques underlying Groth's and Lipmaa's constructions.

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

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

‡Microsoft Research. parno@microsoft.com

§Columbia University. mariana@cs.columbia.edu

1 Introduction

The PCP theorem [3–5, 22] provided *a new characterization of NP* that revolutionized the notion of "proof" – in particular, it showed that NP statements have probabilistically checkable proofs (PCPs) that can be verified in time polylogarithmic in the size of a classical proof. Kilian adapted this new characterization of NP to the cryptographic setting, showing that one can use PCPs to construct interactive *arguments* (i.e., computationally sound proof systems [13]) for NP that are *succinct* – i.e., polylogarithmic also in their communication complexity. Micali [40] showed how to make these arguments *non-interactive* by applying the Fiat-Shamir heuristic [23]: the prover applies a hash function, modeled as a random oracle [7], to its PCP string both as a form of commitment and to non-interactively generate the verifier's PCP queries. Recent works [8, 20, 31] (see also [18]) have improved Micali's construction by removing the random oracles, which are known to be uninstantiable [15], replacing them with "extractable collision-resistant hash functions" (ECRHs), whose security relies on the plausible, but *non-falsifiable* [41], 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 – namely, the entire PCP.

In short, the PCP theorem provides a spectacularly powerful characterization of NP that is useful for, among other things, constructing SNARGs and SNARKs. But, as remarkable as the PCP theorem is, it was not conceived exclusively for cryptographic applications. So, it seems reasonable to ask: Can we construct better SNARGs/SNARKs without (explicitly) using PCPs? Is there a different characterization of NP that is better suited for cryptographic applications?

1.1 Quadratic Span Programs: A New Characterization of NP

We introduce *quadratic span programs (QSPs)*, a new characterization of NP that allows us to construct very efficient SNARKs without PCPs.

To explain QSPs, it is helpful to recall span programs (SPs), a linear-algebraic model of computation introduced by Karchmer and Wigderson [37]. A SP over a field F consists of a nonzero target vector $\mathbf{t}$ over F, a set $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ of vectors, 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 $\cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}$. The SP is said to "compute" a function f if the following is true for all input assignments $u \in \{0,1\}^n$: the target vector is in the span of the vectors that "belong" to the input assignment u – namely, the vectors with indices in $\mathcal{I}_u = \mathcal{I}_{free} \cup_i \mathcal{I}_{i,u_i}$ – iff $f(u) = 1$. The size of the span program is m.

Functions with polynomial size SPs are in NC^2, since linear algebra is in NC^2. Consequently, it is widely believed that SPs are *not* powerful enough to efficiently compute all functions in P (or to 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 target polynomial $t(x)$, 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 $\cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}$.*

[1] We know that the security of succinct non-interactive arguments cannot be based on falsifiable assumptions via black box reductions [1, 27]; hence non-falsifiable "knowledge" assumptions seem unavoidable in this context.

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 $k \notin \mathcal{I}_u$, such that*

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

Q "computes" a boolean function $f : \{0,1\}^n \to \{0,1\}$ if it accepts exactly those inputs u where $f(u) = 1$. The size of Q is m. The degree of Q is $\deg(t(x))$.

QSPs seem like a natural extension of (linear) SPs. A SP accepts an input u iff the target vector can be expressed as a linear combination of vectors that "belong" to u. A QSP accepts an input u iff (a multiple of) the target *polynomial* can be expressed as a *product* of two linear combinations of vectors that "belong" to u, where "product" is defined as polynomial multiplication.

Importantly, unlike SPs, QSPs can efficiently compute any efficiently computable function, and so we can define NP as the set of languages with proofs that can be verified efficiently by QSPs.

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

At a high level, our "canonical" QSP works as follows. (See section 2 for details.) Suppose C is a boolean circuit that computes a function f. While we cannot necessarily construct a SP that "computes" f efficiently, we *can* construct a SP S that merely verifies that a bit string represents a valid assignment of C's wires, including the interior wires. Specifically, we construct S by concatenating SPs S_g for the individual gates g, where each S_g verifies that the wire values of gate g are consistent, and each S_g "owns" a few dimensions of the overall vector space that S lives in. The dimension of the vectors in S depends linearly on the number of gates in C. The vectors in S are all labeled and allotted to assigned wires in C, and if a wire participates in gate g, it indirectly receives vectors from S_g.

We embed the SP S into our QSP twice – both inside $\mathcal{V}$ and inside $\mathcal{W}$. Specifically, we pick some distinct values $r_1, \ldots, r_{m_v} \in F$, and interpolate $v_0(x)$ so that $(v_0(r_1), \ldots, v_0(r_{m_v}))$ is the target vector of S. We interpolate the other $v_k(x)$'s so that the vectors $(v_k(r_1), \ldots, v_k(r_{m_v}))$ correspond to the other vectors in S. When we set $t_v(x) = \prod_{i=1}^{m_v}(x - r_i)$ and set $t(x)$ to be some multiple of $t_v(x)$, the divisibility check of Equation 1 implicitly runs the SP S – i.e., it checks that $v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)$ is divisible by $t_v(x)$, hence is zero at the r_i's, hence the target vector $(v_0(r_1), \ldots, v_0(r_{m_v}))$ is in the span of the vectors $(v_k(r_1), \ldots, v_k(r_{m_v}))$, hence the linear combination $\{a_k\}$ is associated to a valid assignment of C's wires. (For $\mathcal{W}$, we construct $t_w(x)$ similarly but with different roots, make $t(x)$ divisible also by $t_w(x)$, and proceed analogously.)

The reason that the SP S does not already "compute" f is that it has labeled (non-free) vectors even for the interior wires of C, whereas an SP for f is only permitted to have labeled vectors for the input wires. So, a cheating prover is not required to play by S's rules: in its linear combinations, it can use vectors in S that correspond to conflicting assignments (both '0' and '1') of some interior wire in C. Still, S gives us something. It ensures that, if the prover *does* play by S's rules – if it commits to one assignment for each interior wire of C and only uses vectors in S that correspond to the committed assignment – then the span programs implicit in $\mathcal{V}$ and $\mathcal{W}$ will be satisfied only if the assignment is *valid*.

To ensure that the prover plays by S's rules, we construct another polynomial $t_Q(x)$, another product of linear terms, with roots distinct from $t_v(x)$ and $t_w(x)$, and set $t(x) = t_v(x) \cdot t_w(x) \cdot t_Q(x)$. We set the $v_k(x)$'s (similarly $w_k(x)$'s) via CRT to be equal the original $v_k(x)$'s modulo $t_v(x)$ (so that

the embedded SP S is undisturbed) and set the evaluations of the $v_k(x)$'s and $w_k(x)$'s at the roots of $t_Q(x)$ so that not playing by S's rules becomes fatal: if the prover tries to use a linear combination $(a_1, \ldots, a_m)$ that applies nonzero coefficients to $v_k(x)$'s corresponding to different assignments of the same wire, or if the linear combinations $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ are inconsistent in how they assign a wire, then the resulting product $v(x) \cdot w(x)$ cannot be divisible by $t_Q(x)$. The details here are a bit tricky, and we defer them to Section 2.

Our "canonical" QSP (Section 2) has performance conducive to constructing faster SNARKs.

Theorem 2. *(Informal) Given f whose circuit has s gates, computing the polynomials $t(x)$, $\mathcal{V}$ and $\mathcal{W}$ of our "canonical" QSP takes $O(s)$ work ($O(s)$ F operations). Given $u \in \{0,1\}^n$ for which $f(u) = 1$, 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 polynomial $h(x) = v(x) \cdot w(x)/t(x)$ takes $\tilde{O}(s)$ work.*

We obtain such performance by exploiting the fact that the $v_k(x)$'s and $w_k(x)$'s in our canonical QSP 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 $t(x)$, 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 intriguing open problem.

Beginning in Section 7, we present a variant of QSPs, called "quadratic arithmetic programs" (QAPs), that "naturally" compute *arithmetic* circuits over large fields, along with SNARK constructions that use QAPs. QAPs may often have performance advantages over QSPs.

1.2 From QSPs to SNARKs and NIZKs

Our SNARK for f uses a common reference string (CRS) in which the QSP polynomials (e.g., $\{v_k(x)\}$) are represented by terms $g^{v_k(s)}$ (etc.), where g is a generator of bilinear group, and $s \in F$ 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 $g_v = g^{v(s)}$, $g_w = g^{w(s)}$, $g_h = g^{h(s)}$ for $v(x)$, $w(x)$, $h(x)$ as above. The verifier confirms $e(g_v, g_w) = e(g_h, g^{t(s)})$, where e is the bilinear map. (The actual scheme is more complicated.) For security, we require a non-falsifiable "knowledge" assumption as in [32, 38].

It is easy to randomize our public-verifier SNARK to make it zero knowledge and obtain a non-interactive zero-knowledge (NIZK) argument. Essentially, the prover adds random $\delta_v t(s)$ and $\delta_w t(s)$ in the exponent to g_v and g_w and alters g_h accordingly. Our NIZK consists of only 7 group elements. Prover computation is quasi-linear in the circuit size, only due to the computation of $h(x)$. (To compare: Groth's [32] NIZK has 42 group elements, Lipmaa's [38] has 39, and both have prover computation that is quadratic.) Aside from a pre-processing step to compute the CRS, verifier computation is linear in the size of the statement (not witness). This can be reduced pragmatically by applying an *ordinary* (not extractable) collision-resistant hash to the statement so that verification is *constant* aside from the hashing, or asymptotically by applying a small PCP over the statement (not the witness, as in traditional PCP-based approaches). See the overview in Section 3, and Section 4, for details.

As also observed by Groth [32] and Lipmaa [38], our result immediately yields a sub-linear size Zap [21] (details in Section 4.4).

1.3 Comparisons to Other Work on Succinct Arguments without PCPs

Here, we review and compare other constructions of succinct argument systems for NP-complete languages, and delegation schemes for P-complete languages, that avoid the PCP theorem.[2]

Our work on succinct arguments is perhaps best seen as an abstraction and refinement of Groth's surprising recent results [32]. He showed that PCPs are not (explicitly) necessary at all for succinct NIZKs, as long as one permits a pre-processing stage to establish a common reference string (CRS). Groth and others [33, 34] had previously constructed NIZKs over bilinear groups with various attractive properties, but with size linear in the circuit. Recently, Groth essentially found a way to compress the proof components succinctly into a constant number of group elements. Security relies on a non-falsifiable "knowledge of exponent" assumption, similar to the one we use.

The main drawback of Groth's succinct NIZK is the prover complexity, which is *quadratic* in the circuit size. Lipmaa [38] showed how to reduce the size of the CRS in Groth's construction from quadratic to quasi-linear in the circuit size, but with prover complexity still quadratic. Recall that in our scheme the CRS size and prover complexity are linear, aside from the quasi-linear computation needed to compute the quotient polynomial $h(x)$.

Ishai, Kushilevitz and Ostrovsky (IKO) [36] were perhaps the first to seriously question whether succinct arguments need the PCP theorem, noting that such arguments have a peculiar structure: "the classical proof is expanded by introducing a significant amount of redundancy", but then the expanded proof is hashed and thus "dramatically shrinks in size". They presented a "direct approach, or a shortcut, that combines the two steps into one." IKO avoids the PCP theorem by using the simpler exponential-length Hadamard PCP [3] – that is, IKO still uses PCPs, but not "short" ones. Since the Hadamard PCP is *linear* – i.e., the prover's response is a linear function π of the verifier's query – it is possible for the prover, in polynomial time, to commit cryptographically to π, and therefore to its responses to all of the exponentially-many possible verifier queries. The verifier sets up this commitment scheme in a *pre-processing step*: it generates a key pair for an additively homomorphic encryption scheme, such as Paillier [42], encrypts (each entry of) a random vector r, and sends these ciphertexts to the prover. The prover commits to π by sending a ciphertext that encrypts $s = \pi(r)$, which it computes by using the additive homomorphism; the verifier decrypts and retains s. Afterwards, when the verifier wants to query q, it picks a random α and sends $(q, r + \alpha \cdot q)$. The prover responds with two ciphertexts that encrypt $(a, b) = (\pi(q), \pi(r + \alpha \cdot q))$, which it computes homomorphically. The verifier decrypts, and accepts a as the correct response to q if $b = s + \alpha \cdot a$. Security relies on a non-falsifiable "knowledge" assumption: that there is an extractor that can recover π from the commitment s.

A drawback of IKO is that, due to the inefficiency of the Hadamard PCP, the prover's computation (and also the verifier's computation in a pre-processing step) is quadratic in the size of the classical proof, similar to Groth [32]. (See [46] for an implementation of IKO that claims a factor of 10^{17} improvement over a naive implementation of IKO, yet still finds the prover complexity prohibitive in general, and therefore constructs more efficient PCPs for specific functions such as matrix multiplication.)

Gennaro, Gentry and Parno [24] (GGP) presented a *verifiable computation (VC)* scheme that allows a client to outsource an arbitrarily complex (efficiently computable) function to a worker

[2] Rothblum and Vadhan [45] show that PCPs are "inherent" in argument systems that (1) base security on certain *falsifiable* assumptions and (2) are *robust* (secure when the prover has access to a verification oracle). The systems we discuss here are based on non-falsifiable assumptions [9, 12, 32, 36, 38, 47] or are not robust [2, 17, 24].

and verify the worker's computation in constant time, where the worker's complexity is only *linear* in the size of the circuit. GGP and related schemes [2, 17] embed a one-time VC scheme, such as a Yao garbled circuit, inside a fully homomorphic encryption (FHE) scheme [25, 26, 44] to obtain reusability. These VC schemes have the appealing feature of being based on a falsifiable assumption – namely, the semantic security of FHE. A drawback, aside from the current impracticality of FHE, is that these schemes are *not robust*: the client must redo an expensive preprocessing step (linear in the circuit size) after it detects that the worker has cheated. Our SNARKs immediately give us VC schemes that are robust and have better practical performance.

Valiant [47] asked how to construct *incrementally verifiable* succinct arguments for the correctness of a computation whose space complexity is much less than its circuit size. He showed that, for argument systems with certain properties, arguments π_1, π_2 can be *composed* into a single argument π (of equal length) that "I have seen convincing proofs π_1, π_2." Then, π can succinctly replace π_1, π_2 assuming an extractor that watches the generation of π can recover π_1, π_2. By recursively composing SNARKs for $(M : c_i \xrightarrow{j-i} c_j)$ ("running TM M from configuration c_i for $j-i$ steps gives configuration c_j"), one can build a SNARK for the correctness of M's entire computation. Verifier preprocessing, and CRS size, are polynomial only in the *space complexity* of the computation (the size of the configurations), not the circuit size. Unfortunately, at the time of Valiant's work, suitable argument systems existed only heuristically in the random oracle model. Very recently, Boneh et al. [12] and Bitansky et al. [9] observed that Groth's SNARKs [32] are suitable. Bitansky et al. provide other extensions of Valiant's results, including allowing dynamic computations.

Our results are complementary to [9, 12, 47]. Bitansky et al. observe that recursive composition of SNARKs is analogous to "bootstrapping" in FHE. This analogy holds also with respect to performance: while recursive composition (of SNARKs, and within FHE) improves certain aspects of performance (it reduces dependence on circuit size), it also degrades performance in terms of the security parameter. In particular, the quadratic prover computation in Groth's and Lipmaa's SNARKs increases the complexity of the bootstrapping step in Bitansky et al.'s construction. Our SNARKs, with quasi-linear prover computation, make bootstrapping more practical.

2 Quadratic Span Programs (QSPs)

In the Introduction, we defined Quadratic Span Programs (QSPs) in a manner that is superficially similar to that of span programs (SPs). In this section, we construct a "canonical" QSP for a function f indirectly: first we construct a *traditional SP* to check the evaluation of f; we then combine this with a *consistency checker* that ensures the check is itself performed correctly.

Suppose C is a Boolean circuit that computes a function f. While we cannot necessarily construct a SP that "computes" f efficiently, we *can* construct a SP S that merely verifies that a bit string represents a valid assignment of C's wires, including the interior wires (Section 2.1). However, the SP S does not "compute" f; it has labeled (non-free) vectors even for the interior wires of C, whereas an SP for f is only permitted to have labeled vectors for the input wires. If we simply move the vectors for the interior wires to the "free" set, then a cheating prover need not play by S's rules: in its linear combinations, it can use vectors in S that correspond to conflicting assignments (both '0' and '1') of some interior wire in C. Still, S gives us something. It ensures that, if the prover *does* play by S's rules – if it selects exactly one assignment for each interior wire of C and uses only vectors in S that correspond to that assignment – then the span program will be satisfied only if the assignment is *valid*, i.e., corresponds to a valid evaluation of f.

To ensure the prover plays by the rules, i.e., chooses either vectors for 0 or vectors for 1 for each interior wire, our canonical QSP incorporates a "consistency checker", built out of consistency checkers for each wire (Section 2.2). We can express both the consistency checks and the SP S as polynomials and compose them into the polynomials that constitute the QSP (Section 2.3).

We provide details of our construction below.

2.1 Component 1: A Useful Linear Span Program

We do not know how to efficiently construct SPs for arbitrary functions $f \in$ P. But we *can* always efficiently construct a SP for a function *related* to f, called the *circuit checker function* for f.

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

Lemma 1. *Suppose that f consists of Boolean gates from some set Γ – e.g, $\Gamma = \{\text{NAND}\}$. Suppose that, for each gate $g \in \Gamma$, there is a SP of size m' that computes whether an input is a satisfying assignment of g's input/output wires. If $f : \{0,1\}^n \to \{0,1\}$ has a Boolean circuit C with s gates from Γ, there is SP S of size $m = s \cdot m'$ that computes f's circuit checker function ϕ. S is a straightforward composition of SPs $\{S_g\}$ for the individual gates g of f.*

Lemma 1 says that there is an SP $S = (\mathbf{t}, \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$ for ϕ that is linear in f's circuit size (and linear in the size of NAND's SP). The proof is constructive. The target vector of S is literally a concatenation of target vectors for the S_g's. S also inherits its other vectors from the S_g's, but somewhat modified: S_g "owns" a few dimensions of the vector space that S lives in, and S_g's vectors are otherwise padded with 0's to fill out S's space. If a wire $i \in [N]$ participates in gate g, it indirectly receives vectors from S_g – i.e., $\{\mathbf{v}_k : k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1} \subset [m]\}$ includes (modified) vectors from S_g, which depend on whether i was a left input, right input, or output of g. To optimize performance, we will exploit the extreme sparseness of vectors in $\mathcal{V}$.

Proof. (Lemma 1) For convenience, assume that each of the s gates of C each have three wires: a left input, a right input, and an output. (The proof easily extends to the more general case.) For a gate $g \in [s]$, let $g_l, g_r, g_o \in [m]$ be the indices of the wires left, right and output. Of course, a wire may connect to many gates – it may be the output of one gate, and an input to many gates – so we may have, for example, $g_o = g'_l$ for different gates g, g'.

For each g, we will define a SP. Let $m^{(g)}, d^{(g)}$ be parameters denoting the number of rows and columns, respectively, in the SP for g. Our span program $S = (\mathbf{t}, \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$ for ϕ will have m row vectors of dimension d, where $m = \sum_g m^{(g)}$ and $d = \sum_g d^{(g)}$. In S's $m \times d$ matrix of row vectors, each gate g "owns" a rectangle of $m^{(g)}$ consecutive rows and $d^{(g)}$ consecutive columns – call this block of row indices $\mathcal{I}^{(g)} = \mathcal{I}^{(g)}_{free} \cup \mathcal{I}^{(g)}_{labeled}$, where $\mathcal{I}^{(g)}_{labeled} = \cup_{i\in\{g_l,g_r,g_o\},j\in\{0,1\}}\mathcal{I}^{(g)}_{ij}$). Therefore, we write the SP for g as $S^{(g)} = (\mathbf{t}^{(g)}, \mathcal{V}^{(g)} = \{\mathbf{v}^{(g)}_k : k \in \mathcal{I}^{(g)}\}, \mathcal{I}^{(g)}_{free}, \mathcal{I}^{(g)}_{labeled} = \cup_{i\in\{g_l,g_r,g_o\},j\in\{0,1\}}\mathcal{I}^{(g)}_{ij})$.

In S, the target vector $\mathbf{t}$ is a simple concatenation of the target vectors for the gates: $\mathbf{t} = \mathbf{t}^{(1)} \| \cdots \| \mathbf{t}^{(s)}$. For each $k \in \mathcal{I}^{(g)}$, the vector $\mathbf{v}_k$ is the same as one of the vectors in $\mathcal{V}^{(g)}$, except that it is shifted so that its column indices cover $\mathcal{J}^{(g)}$, the columns "owned" by g. The free vectors are a combination of the free vectors from each gate, similarly adjusted: $\mathcal{I}_{free} = \cup_{g\in[s]}\mathcal{I}^{(g)}_{free}$. Also,

$\mathcal{I}_{ij} = \cup_{i \in \{g_l, g_r, g_o\}} \mathcal{I}_{ij}^{(g)}$.

It remains to show that S computes ϕ. Suppose u' is such that $\phi(u') = 1$. Then u' corresponds to a satisfying assignment of the wires of C, the boolean circuit for f. Therefore, for each gate g of C, the assignment $u'_{g_l}, u'_{g_r}, u'_{g_o}$ to the wires (g_l, g_r, g_o) is satisfying, and there is a linear combination of the vectors in $\mathcal{V}^{(g)}$ with indices in $\mathcal{I}_{g_l,u'_{g_l}} \cup \mathcal{I}_{g_r,u'_{g_r}} \cup \mathcal{I}_{g_o,u'_{g_o}}$ that equals the target vector $\mathbf{t}^{(g)}$. This linear combination is restricted to indices in $\mathcal{I}^{(g)}$, and therefore is independent of indices associated to other gates. Since the column sets associated to different gates are also disjoint, we can compose the linear combinations for the individual gates to get a linear combination, restricted to indices in $\cup_{i,u'_i} I_{iu'_i}$, that equals the target vector $\mathbf{t}$.

Conversely, suppose that, for some u', there is a linear combination, restricted to indices in $\cup_{i,u'_i} I_{iu'_i}$, that equals the target vector $\mathbf{t}$. Then, for each gate g, there is a linear combination, restricted to indices in $\mathcal{I}_{g_l,u'_{g_l}} \cup \mathcal{I}_{g_r,u'_{g_r}} \cup \mathcal{I}_{g_o,u'_{g_o}}$, that equals the target vector $\mathbf{t}^{(g)}$. Therefore, the assignment u' satisfies each individual gate of C. By the definition of ϕ, we have $\phi(u') = 1$. ■

It is straightforward to construct a small SP for NAND.[3]

Lemma 2. *There is a conscientious SP for NAND consisting of 12 vectors of dimension 9.*

Proof. (Lemma 2) Let $(\mathbf{v}_{l0}, \mathbf{v}'_{l0}, \mathbf{v}_{l1}, \mathbf{v}'_{l1}, \mathbf{v}_{r0}, \mathbf{v}'_{r0}, \mathbf{v}_{r1}, \mathbf{v}'_{r1}, \mathbf{t})$ be 9 linearly independent vectors of dimension 9. The target vector is $\mathbf{t}$, and the pair of vectors $\mathcal{V}_{l0} = (\mathbf{v}_{l0}, \mathbf{v}'_{l0})$ belongs to the assignment of 0 to the left wire, etc. Set $\mathbf{v}_{o0} = \mathbf{t} - \mathbf{v}_{l1} - \mathbf{v}_{r1}$ and $\mathcal{V}_{o0} = \{\mathbf{v}_{o0}\}$, so that one can express $\mathbf{t}$ as a linear combination of vectors in $\mathcal{V}_{l1} \cup \mathcal{V}_{r1} \cup \mathcal{V}_{o0}$. Set $\mathbf{v}_{o1} = \mathbf{t} - \mathbf{v}_{l0} - \mathbf{v}_{r0}$, $\mathbf{v}'_{o1} = \mathbf{t} - \mathbf{v}'_{l0} - \mathbf{v}'_{r1}$, and $\mathbf{v}''_{o1} = \mathbf{t} - \mathbf{v}'_{l1} - \mathbf{v}'_{r0}$, and $\mathcal{V}_{o1} = \{\mathbf{v}_{o1}, \mathbf{v}'_{o1}, \mathbf{v}''_{o1}\}$, so that one can express $\mathbf{t}$ as a linear combination of vectors associated to the other satisfying gate assignments.

First, let us prove that the SP is conscientious – i.e., that a linear combination associated to a satisfying gate assignment must use at least one vector associated to each wire. For the satisfying assignment $(0, 0, 1)$, we have the vectors $(\mathbf{v}_{l0}, \mathbf{v}'_{l0}, \mathbf{v}_{r0}, \mathbf{v}'_{r0}, \mathbf{t} - \mathbf{v}_{l0} - \mathbf{v}_{r0}, \mathbf{t} - \mathbf{v}'_{l0} - \mathbf{v}'_{r1}, \mathbf{t} - \mathbf{v}'_{l1} - \mathbf{v}'_{r0})$. It is clear that no linear combination of just the left and right vectors – i.e., $(\mathbf{v}_{l0}, \mathbf{v}'_{l0}, \mathbf{v}_{r0}, \mathbf{v}'_{r0})$ – can give $\mathbf{t}$, since these vectors are linearly independent from $\mathbf{t}$. Wlog, consider the left vectors and the output vectors: here also there is no linear combination that gives $\mathbf{t}$, since none of the output vectors can be used, since they each contain at least one right vector that does not appear elsewhere.

Now, let us prove that there is no way of expressing the target vector as a linear combination of vectors associated to an unsatisfying assignment. First, consider the unsatisfying assignment $(1, 1, 1)$. No linear combination that sums to $\mathbf{t}$ can use any of the vectors associated to output 1, since each of them contains one of the following vectors $(\mathbf{v}_{l0}, \mathbf{v}'_{l0}, \mathbf{v}_{r0}, \mathbf{v}'_{r0})$ associated to assigning 0 to a left or right wire, and moreover each of these terms appears only once, in only one of the output vectors, and nowhere else. Therefore the linear combination must be restricted to $(\mathbf{v}_{l1}, \mathbf{v}'_{l1}, \mathbf{v}_{r1}, \mathbf{v}'_{r1})$, but of course this is impossible. Finally, consider an unsatisfying assignment where the output bit is 0, and at least one bit of the input is 0. We have $\mathbf{v}_{o0} = \mathbf{t} - \mathbf{v}_{l1} - \mathbf{v}_{r1}$. Since one of the input bits is 0, either $\mathbf{v}_{l1}$ or $\mathbf{v}_{r1}$ (or both) appears inside $\mathbf{v}_{o0}$, but nowhere else, and therefore the linear combination cannot use $\mathbf{v}_{o0}$. Since the SP is conscientious, there is no hope that $\mathbf{t}$ can be expressed as a linear combination of vectors associated only to the input bits. ■

[3]The same techniques apply to AND and OR. They also have exactly one input pair that gives one of the outputs.

When we embed the SP S into our QSP, we can only explicitly constrain the prover to consistently use the vectors corresponding to f's input, while the vectors for f's interior wires will be free. However, if the prover is nice enough to play by S's rules – if it uses vectors only from one of the two sets for each wire, then the SP S will be satisfied only if the wire assignment is a *valid* execution of f. This is formalized in Lemma 3 below. Later, we will construct a consistency checker designed to ensure that the prover abides by S's rules within our QSP.

Lemma 3. *Let $S = (\mathbf{t}, \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$ be a SP for the circuit 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:*

- $\mathbf{t} = \sum_k a_k \cdot \mathbf{v}_k$.
- *For all $k \in \cup_{i=1}^n I_{i\overline{u}_i}$, we have $a_k = 0$.*
- *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 third – i.e., must make a "double assignment" of some wire $i \in \{n+1, \ldots, N\}$.

Proof. (Lemma 3) 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 $\mathbf{t} = \sum_k a_k \cdot \mathbf{v}_k$ 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 $f(u) = 1$. ■

We make some tweaks to f's circuit and the SP for ϕ before embedding it in our canonical QSP.

First, the construction of our QSP is cleaner when f has low fan-out – e.g., two. We could reduce fan-out to two by using *split gates* while increasing the number of gates by only a constant factor, but prefer to keep things simple by using only NAND/AND gates. Functionally, a split gate is equivalent to an AND gate with fan-out 2 with a dummy wire set to '1'. Hence Lemma 4, which says we can reduce fan-out to 2 by adding dummy AND gates and increase the number of gates by only a constant factor. The tradeoff is that we need to add a single dummy input wire, always set to '1', that participates in many gates.

Lemma 4. *Let $f : \{0,1\}^n \to \{0,1\}$ be a function whose Boolean circuit C has s NAND/AND/OR gates. There is a function $f^\dagger : \{0,1\}^{n+1} \to \{0,1\}$ with $f^\dagger(1,u) = f(u)$, whose Boolean circuit $C^\dagger$ has at most $3s$ NAND/AND/OR gates, where all wires except the first are inputs to at most 2 gates, and each bit of u is an input to only 1 AND gate (where the other is the first (dummy) input). The circuit checker function $\phi^\dagger$ for $f^\dagger(1,\cdot)$ has a SP of size $\leq 36s$ and dimension $27s$, and sparseness of the vectors is preserved. Also, for each $\mathcal{I}_{ij} \in \mathcal{I}_{labeled}$, we have $|\mathcal{I}_{ij}| = 1$.*

Proof. (Lemma 4) Suppose that C has a gate g with fan-out d. To construct $C^\dagger$, we reduce g's fan-out to two and connect to the output wire of g an inverted binary tree of AND gates, each of which has fan-out two and takes one input from the gate before it and the other input is the added dummy input. To eliminate the fan-out d, we need to add at most d such gates. We perform this process independently for the gates in C that have high fan-out. (Of course, we do not perform this process for the dummy input, else we would have an infinite loop.) For all inputs to C (independent of how many gates an input may be connected to in the original circuit), we reduce the number

of input wires per input to one, by making it the input to an AND gate (with the other input the dummy input), and then dealing with fanout in the usual way. Since we know that the dummy input is 1, we can simplify the SPs for the input gates by discarding the AND SP vectors associated to a 0 possibility for the dummy input. Via this simplification, $|\mathcal{I}_{ij}| = 1$ for all of the labeled sets of indices.

Now we bound the number of gates in $C^\dagger$. In C, when there was a fan-out of d, this fan-out d participated in at least "$d/2$ gates" in the rough sense that the d wires were "half of the input" to d gates. Since eliminating fan-out d added at most d gates, $C^\dagger$ has at most 3 times as many gates as C: hence, at most $3s$ gates.

The size of the SP for the circuit checker function for $f^\dagger$ is thus at most $3s \cdot m'$ by Lemma 1, where m' is a bound on the size of the SPs for the individual AND/NAND gates: thus at most $36s$ by Lemma 2. By Lemma 2, the dimension of the SP is at most $27s$. To construct a SP for $\phi^\dagger$ (the circuit checker function for $f^\dagger(1, \cdot)$), we simply eliminate all of the vectors in $f^\dagger$ that corresponded to assigning the first (dummy) input to '0', and put all of the indices for vectors that correspond to assigning the first (dummy) input to '1' in $\mathcal{I}_{free}$. (Thus, in the SP for $\phi^\dagger$, we do not need to consider the dummy input to $C^\dagger$ as an actual input.)

The sparseness property holds for $\phi^\dagger$ – namely, each vector in the SP for $\phi^\dagger$ (aside from the target vector) has only a small constant number of nonzero coefficients, since the vectors in the SP for $\phi^\dagger$ are inherited from the small SPs for the individual gates of $C^\dagger$. ■

Second, to combine the SP with the consistency checker, it will be convenient to view our SP S as consisting of polynomials rather than vectors. (This also permits one to view the composition of S from the gate SPs $\{S_g\}$ as occurring via the Chinese Remainder Theorem.) Suppose $S = (\mathbf{t}, \mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i \in [N], j \in \{0,1\}} \mathcal{I}_{ij})$ has size m and dimension d. Generate d distinct values $r_i \in F$. For each vector $\mathbf{v}_k$, define the polynomial $v_k(x)$ (of degree at most $d-1$) such that $v_k(r_i) = v_{ki}$. In addition, define the polynomial $v_0(x)$ such that $v_0(r_i) = -t_i$. Finally, let $t(x) = \prod_{i=1}^{d}(x - r_i)$. Define the polynomial version of S to be: $S_{poly} = (t(x), \mathcal{V} = \{v_0(x), \ldots, v_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i \in [N], j \in \{0,1\}} \mathcal{I}_{ij})$.

Lemma 5. *S is satisfied iff S_{poly} is satisfied in the sense that there exists $(a_1, \ldots, a_m)$ such that $t(x)$ divides $v(x) = v_0(x) + \sum a_k \cdot v_k(x)$.*

Proof. (Lemma 5) Suppose S is satisfied by $(a_1, \ldots, a_m)$ such that $-\mathbf{t} + \sum a_k \mathbf{v}_k = 0$. Since $-\mathbf{t}$ is the vector of evaluations (at $r_1, \ldots, r_d$) of $v_0(x)$, and $\mathbf{v}_k$ is the vector of evaluations of $v_k(x)$, the above holds iff $v_0(x) + \sum a_k v_k(x)$ evaluates to 0 at all r_i, and hence is divisible by $t(x)$. ■

2.2 Component 2: A Consistency Checker

The second component of our QSP, a *consistency checker*, ensures that the two pairs of linear coefficients, $\{a_k\}$ and $\{b_k\}$, in our QSP obey the rules of the embedded SP S for ϕ – i.e., that they make no "double assignments" to wires in f's circuit C. A double assignment of a wire i occurs when $\{a_k\}$ (or $\{b_k\}$) includes non-zero terms for vectors from both sets $\mathcal{I}_{i0}$ and $\mathcal{I}_{i1}$.

In fact we construct a one-wire consistency checker that enforces a slightly weaker condition: double assignments in $\{a_k\}$ are forbidden, unless $\{b_k\}$ indicates a *non*-assignment – i.e., all $b_k = 0$ (and vice versa). We compose the checkers for individual wires via CRT, and then compose this aggregate with the SP S via CRT to construct the QSP.

The weak consistency condition suffices for our canonical QSP, since each set of linear coefficients $\{a_k\}$ and $\{b_k\}$ comes from an SP instantiation. The conscientious property of the SP requires that we use vectors from at least one of the sets $\mathcal{I}_0$ and $\mathcal{I}_1$ when assigning the corresponding a_k and b_k values. The consistency checker then further guarantees that the non-zero values a_k and b_k used for a particular wire correspond to the same bit assignment.

Definition 3 (Consistency Checker)**.** *For some L, let $\mathcal{I}_0 = \{1, \ldots, L\}$, $\mathcal{I}_1 = \{L+1, \ldots, 2L\}$, $\mathcal{I} = \mathcal{I}_0 \cup \mathcal{I}_1$. A consistency checker for coefficients $\{a_k\}$ and $\{b_k\}$ over $\mathcal{I}$ consists of polynomials $t(x)$, $\mathcal{V} = \{v_k(x) : k \in \mathcal{I}\}$ and $\mathcal{W} = \{w_k(x) : k \in \mathcal{I}\}$ such that*

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

if $\{a_k\}$ and $\{b_k\}$ indicate consistent bit assignments (i.e., for some bit B, $a_k = b_k = 0$ for all $k \in \mathcal{I}_{\bar{B}}$), but not if $\{a_k\}$ and $\{b_k\}$ indicate inconsistent bit assignments in the following sense:

- *There exist $k_a \in \mathcal{I}_0$ and $k'_a \in \mathcal{I}_1$ and $k_b \in \mathcal{I}$ such that $a_{k_a} \neq 0$, $a_{k'_a} \neq 0$ and $b_{k_b} \neq 0$, or*
- *There exist $k_a \in \mathcal{I}$ and $k_b \in \mathcal{I}_0$ and $k'_b \in \mathcal{I}_1$ such that $a_{k_a} \neq 0$, $b_{k_b} \neq 0$ and $b_{k'_b} \neq 0$.*

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

Construction of a Consistency Checker.

1. For $L' = 3L - 2$, 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 $t(x) = \prod_{r\in\mathcal{R}}(x - r)$.
2. Interpolate the polynomials in $\mathcal{V}$ and $\mathcal{W}$ to have degree $(L' + L - 1)$ and satisfy:
   (a) For $k \in \mathcal{I}_0$, $v_k(r) = 0$ for all $r \in \mathcal{R}^{(0)} \cup \{r_1^{(1)}, \ldots, r_L^{(1)}\}$ except $v_k(r_k^{(1)}) = 1$, and $w_k(r) = 0$ for all $r \in \mathcal{R}^{(1)} \cup \{r_1^{(0)}, \ldots, r_L^{(0)}\}$ except $w_k(r_k^{(0)}) = 1$.
   (b) For $k \in \mathcal{I}_1$, $v_k(r) = 0$ for all $r \in \mathcal{R}^{(1)} \cup \{r_1^{(0)}, \ldots, r_L^{(0)}\}$ except $v_{k-L}(r_{k-L}^{(0)}) = 1$, and $w_k(r) = 0$ for all $r \in \mathcal{R}^{(0)} \cup \{r_1^{(1)}, \ldots, r_L^{(1)}\}$ except $w_{k-L}(r_{k-L}^{(1)}) = 1$.

Lemma 6. *The above construction is a consistency checker.*

Proof. (Lemma 6) Clearly, $t(x)$ 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 $L - 1$ roots in $\mathcal{R}^{(1)}$, since it is nonzero of degree $L' + L - 1$ and already has $\mathcal{R}^{(0)}$ as roots. A similar analysis applies to $\sum_{k\in\mathcal{I}_1} a_k \cdot v_k(x)$. 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 $2L - 2$ roots in $\mathcal{R}$. Since $\sum_{k\in\mathcal{I}} b_k \cdot w_k(x)$ is nonzero and degree-$(L' + L - 1)$, it has at most $L' + L - 1$ roots in $\mathcal{R}$. So, the overall product has at most $L' + 3L - 3 < 2L'$ roots, and is therefore not divisible by $t(x)$. ■

We can construct an "aggregate" consistency checker for the partition used in the SP S_{poly} by using CRT to compose consistency checkers for the individual wires.

Definition 4 (Aggregate Consistency Checker)**.** *Let $\mathcal{I} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij}$ be a partition. A aggregate consistency checker for $\mathcal{I}$ consists of polynomials $t(x)$, $\mathcal{V} = \{v_k(x) : k \in \mathcal{I}\}$ and $\mathcal{W} = \{w_k(x) :$*

$k \in \mathcal{I}\}$ *such that*

$$t(x) \quad \text{divides} \quad \left(\sum_{k\in\mathcal{I}} a_k \cdot v_k(x)\right) \cdot \left(\sum_{k\in\mathcal{I}} b_k \cdot w_k(x)\right) \tag{3}$$

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* $i \in [N]$*, there exist* $k_a \in \mathcal{I}_{i0}$ *and* $k'_a \in \mathcal{I}_{i1}$ *and* $k_b \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$ *such that* $a_{k_a} \neq 0$*,* $a_{k'_a} \neq 0$ *and* $b_{k_b} \neq 0$*, or*
- *For some* $i \in [N]$*, there exist* $k_a \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$ *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* $b_{k'_b} \neq 0$*.*

Lemma 7. *We obtain an aggregate consistency checker by merging consistency checkers for the individual indices* $i \in [N]$*, and using disjoint sets of roots* $\mathcal{R} = \{\mathcal{R}^{(i0)}, \mathcal{R}^{(i1)} : i \in [N]\}$*, an aggregate target polynomial* $t(x) = \prod_{r\in\mathcal{R}}(x - r)$*, and CRT to set the polynomials in* $\mathcal{V}$ *and* $\mathcal{W}$*. For the partition of* $\mathcal{I}_{labeled}$ *from the SP* S_{poly} *that computes the circuit checker function* $\phi^\dagger$ *for function* $f^\dagger$ *(the fan-out-2 version of* f*, where* f *has* s *gates), the size of the aggregate consistency checker is* $|\mathcal{I}_{labeled}| \leq 24s$ *and the degree is* $76s$*.*

The aggregate consistency checker is built as follows. For $i \in [N]$, let L_i be an upper bound on $|\mathcal{I}_{i0}|$ and $|\mathcal{I}_{i1}|$.

Construction of an Aggregate Consistency Checker.

1. *Generate all of the roots and the target polynomial.* For all $i \in [N]$, for $L'_i = 3L_i - 2$, select distinct roots $\mathcal{R}^{(i0)} = \{r_1^{(i0)}, \ldots, r_{L'_i}^{(i0)}\}$ and $\mathcal{R}^{(i1)} = \{r_1^{(i1)}, \ldots, r_{L'}^{(i1)}\}$ from F. (The roots are distinct across the i's as well.) Set $\mathcal{R} = \cup_i \mathcal{R}^{(i0)} \cup \mathcal{R}^{(i1)}$. Set $t_i(x) = \prod_{r\in\mathcal{R}^{(i0)}\cup\mathcal{R}^{(i1)}}(x - r)$ and $t(x) = \prod_i t_i(x) = \prod_{r\in\mathcal{R}}(x - r)$.
2. *Generate polynomials for the individual checkers.* For $i \in [N]$, construct sets of polynomials $\mathcal{V}^{(i)}$ and $\mathcal{W}^{(i)}$ by interpolating polynomials to have degree $(L'_i + L_i - 1)$ and satisfy the following. (For convenience, for this step, we view $\mathcal{I}_{i0}$ as being the set of indices $\{1, \ldots, L_i\}$ and $\mathcal{I}_{i1}$ being the set of indices $\{L_i + 1, \ldots, 2L_i\}$.)
   (a) For $k \in \mathcal{I}_{i0}$, $v_k^{(i)}(r) = 0$ for all $r \in \mathcal{R}^{(i0)} \cup \{r_1^{(i1)}, \ldots, r_{L_i}^{(i1)}\}$ except $v_k^{(i)}(r_k^{(i1)}) = 1$, and $w_k^{(i)}(r) = 0$ for all $r \in \mathcal{R}^{(i1)} \cup \{r_1^{(i0)}, \ldots, r_{L_i}^{(i0)}\}$ except $w_k^{(i)}(r_k^{(i0)}) = 1$.
   (b) For $k \in \mathcal{I}_{i1}$, $v_k^{(i)}(r) = 0$ for all $r \in \mathcal{R}^{(i1)} \cup \{r_1^{(i0)}, \ldots, r_{L_i}^{(i0)}\}$ except $v_{k-L_i}^{(i)}(r_{k-L_i}^{(i0)}) = 1$, and $w_k^{(i)}(r) = 0$ for all $r \in \mathcal{R}^{(i0)} \cup \{r_1^{(i1)}, \ldots, r_{L_i}^{(i1)}\}$ except $w_{k-L_i}^{(i)}(r_{k-L_i}^{(i1)}) = 1$.
3. *Compose individual 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 $\deg(t(x)) - 1$ and satisfy $v_k(x) = v_k^{(i)}(x) \bmod t_i(x)$ and $v_k(x) = 0 \bmod t(x)/t_i(x)$. Analogously for $w_k(x)$. Set $\mathcal{V} = \{v_k(x)\}$ and $\mathcal{W} = \{w_k(x)\}$.

Now, let us consider the performance aspects of the aggregate consistency checker when applied to the partition $\mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij}$ of our SP S (or S_{poly}). Recall that, for a function f with s gates, we defined a circuit checker function ϕ that has a SP of size $12s$ and dimension $9s$. (An improved conscientious SP for NAND would improve this number.) However, if f did not originally have low fan-out, we first transformed f to a function $f^\dagger$ such that $f^\dagger(1, u) = f(u)$ that has a circuit with at most $3s$ gates and fan-out 2, and then defined a circuit checker function $\phi^\dagger$

for $f^\dagger$ that has a SP of size $36s$ and dimension $27s$. We will pessimistically focus on the latter SP, and assume that S_{poly} is derived from that.

While the size of the SP can be up to $36s$, potentially many of the vectors in S (S_{poly}) are free, and so $|\mathcal{I}_{labeled}|$ may be much smaller than $36s$. In particular, all of the wires emanating from the dummy input of $f^\dagger$ are free. This is because we know *a priori* that we will always assign '1' to this bit, and so the SP only needs to provide vectors for that assignment and there is no need to make them labeled (non-free). Half of the wires added by the transformation from f to $f^\dagger$ emanate from the dummy input. So, the size of $\mathcal{I}_{labeled}$ at most doubles, not triples. In particular, $|\mathcal{I}_{labeled}| \leq 24s$.

Regarding the degree of the checker, since the fanout in the circuit is 2, each wire participates in at most 3 gates, one as an output and two as an input. For each assignment, each wire receives up to two vectors per NAND gate that it participates in as an input, and up to three vectors per NAND gate that it participates in as an output, for a total of 7 vectors. So, L is bounded by 7 for all wires, and thus L' is bounded by 19. Since there are at most $2s$ wires in the circuit (excluding, as above, the wires from the dummy input) and since the degree of each individual checker is $2L'$, the overall degree of the aggregate checker is at most $76s$.

Proof. (Lemma 7) If $\{a_k\}$, $\{b_k\}$ indicate consistent assignments, then they indicate consistent assignments of the i-th bit for k restricted to $\mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. Hence, $t_i(x)$ divides the product in Equation 3 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 $t_i(x)$ for all $k \notin \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$, the overall (unrestricted) product in Equation 3 is divisible by $t_i(x)$. Since this is true of all i, the product is divisible by $t(x)$.

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 6, $t_i(x)$ does not divide the product in Equation 3 when the summations are restricted to $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. As above, $t_i(x)$ divides everything else, and thus the overall product in Equation 3 is not divisible by $t_i(x)$, and thus not divisible by $t(x)$.

As described above, for the partition of $\mathcal{I}_{labeled}$ from the SP S_{poly} that computes the circuit checker function ϕ for a function f whose circuit has s gates, the size of the aggregate consistency checker is $|\mathcal{I}_{labeled}| \leq 24s$ and the degree is $76s$. ■

2.3 Our Canonical QSP

Here we 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 to merge the polynomial version of the SP for the circuit checker function ϕ, described in Section 2.1, with the polynomials for the consistency checker from Section 2.2.

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. Construct a Boolean circuit $C^\dagger$ with only fan-out-2 gates for $f^\dagger$, per Lemma 4.
3. Construct a SP $S = (\mathbf{t}, \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$ for the circuit checker function $\phi^\dagger$ of $f^\dagger$.
4. From S, using disjoint sets of roots $\mathcal{R}^{(\mathcal{V})}$ and $\mathcal{R}^{(\mathcal{W})}$, construct two incarnations of the SP S_{poly} for $\phi^\dagger$ – namely, $S^{(\mathcal{V})}_{poly} = (\hat{t}^{(\mathcal{V})}(x), \hat{\mathcal{V}} = \{\hat{v}_0(x), \ldots, \hat{v}_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$ and $S^{(\mathcal{W})}_{poly} = (\hat{t}^{(\mathcal{W})}(x), \hat{\mathcal{W}} = \{\hat{w}_0(x), \ldots, \hat{w}_m(x)\}, \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[N],j\in\{0,1\}}\mathcal{I}_{ij})$. Note that because we use distinct roots for each incarnation, the resulting target polynomials $\hat{t}^{(\mathcal{V})}(x)$ and $\hat{t}^{(\mathcal{W})}(x)$ are relatively prime.

5. 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 consistency checker from Lemma 7, which consists of the following polynomials: $t'(x) = \prod_{r\in\mathcal{R}}(x-r)$, $\mathcal{V}' = \{v'_1(x), \ldots, v'_m(x)\}$ and $\mathcal{W}' = \{w'_1(x), \ldots, w'_m(x)\}$.
6. Define $t(x) = \hat{t}^{(\mathcal{V})}(x)\hat{t}^{(\mathcal{W})}(x)t'(x)$.
7. Finally, for $k \in [m]$, define $\mathcal{V} = \{v_0(x), \ldots, v_m(x)\}$ and $\mathcal{W} = \{w_0(x), \ldots, w_m(x)\}$ using the Chinese Remainder Theorem to interpolate $v_k(x)$ and $w_k(x)$ as follows:

$$v_k(x) = \begin{cases} \hat{v}_k(x) & \bmod \hat{t}^{\mathcal{V}}(x) \\ v'_k(x) & \bmod t'(x) \\ 1 & \bmod \hat{t}^{\mathcal{W}}(x) \quad \text{if } k = 0 \\ 0 & \bmod \hat{t}^{\mathcal{W}}(x) \quad \text{otherwise} \end{cases} \qquad w_k(x) = \begin{cases} \hat{w}_k(x) & \bmod \hat{t}^{\mathcal{W}}(x) \\ w'_k(x) & \bmod t'(x) \\ 1 & \bmod \hat{t}^{\mathcal{V}}(x) \quad \text{if } k = 0 \\ 0 & \bmod \hat{t}^{\mathcal{V}}(x) \quad \text{otherwise} \end{cases}$$

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

Theorem 3. *For any boolean circuit f with n inputs, s gates, and $N = n + s$ total wire values, the canonical QSP computes f. The size of the canonical QSP is $36s$ and the degree is $130s$.*

Proof. (Theorem 3) Suppose $f(u) = 1$. Then we can evaluate the circuit C for f to find wire values that satisfy C; hence the circuit checker ϕ evaluates to 1, and Lemma 5 tells us there must be a set of coefficients $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ for each of our polynomial span programs $S^{(\mathcal{V})}_{poly}$ and $S^{(\mathcal{W})}_{poly}$ such that $\hat{t}^{(\mathcal{V})}(x)$ divides $\hat{v}(x) = \hat{v}_0(x) + \sum a_k \cdot \hat{v}_k(x)$ and $\hat{t}^{(\mathcal{W})}(x)$ divides $\hat{w}(x) = \hat{w}_0(x) + \sum b_k \cdot \hat{w}_k(x)$. Since $\{a_k\}$ and $\{b_k\}$ indicate consistent bit assignments of wires in the evaluation of $f(u)$, the consistency checker is happy; that is, by Lemma 6, we have that $t'(x)$ divides $v'(x)w'(x)$ where $v'(x) = \sum_{k\in[m]} a_k v'_k(x)$ and $w'(x) = \sum_{k\in[m]} b_k w'_k(x)$.

Letting $v(x) = v_0(x) + \sum a_k \cdot v_k(x)$ and $w(x) = w_0(x) + \sum b_k \cdot w_k(x)$, we have that $v(x)w(x) \bmod t'(x) = v'(x)w'(x) = 0 \bmod t'(x)$. Similarly, $v(x)w(x) = \hat{v}(x) \cdot 1 = 0 \bmod \hat{t}^{\mathcal{V}}(x)$, and $v(x)w(x) = 1 \cdot \hat{w}(x) = 0 \bmod \hat{t}^{\mathcal{W}}(x)$. In other words, $v(x)w(x)$ is divisible by $t'(x)$, $\hat{t}^{\mathcal{V}}(x)$, and $\hat{t}^{\mathcal{W}}(x)$ (none of which share any roots), and hence is divisible by $t(x)$.

For the other direction, now suppose that there exist tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ such that $v(x)w(x) = 0 \bmod t(x)$ where again $v(x) = v_0(x) + \sum_k a_k v_k(x)$ and $w(x) = w_0(x) + \sum_k b_k w_k(x)$. (We will not use the additional hypothesis that the tuples correspond to u – i.e., that $a_k = 0 = b_k$ for all $k \notin \mathcal{I}_u$ – until a little later.) Since $v(x)w(x) = 0 \bmod t(x)$, we also have that $v(x)w(x) = 0 \bmod t'(x)$. As before, we have that $v(x)w(x) = v'(x)w'(x) \bmod t'(x)$, and hence $v'(x)w'(x) = 0 \bmod t'(x)$. As stated in Lemma 6, divisibility by $t'(x)$ implies that $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ are consistent, i.e., that neither makes a double assignment of any wire $i \in [N]$ (e.g., uses nonzero $b_{k_1} \in \mathcal{I}_{i0}$ and $b_{k_2} \in \mathcal{I}_{i1}$) unless the other makes no assignment.

Suppose that, wlog, $\{a_k\}$ makes no assignment for the wire i – i.e., that $a_k = 0$ for all $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i1}$. Since the polynomial span program $S^{(\mathcal{V})}_{poly}$ is *conscientious*, it is impossible that $\hat{t}^{(\mathcal{V})}(x)$ divides $\hat{v}(x) = \hat{v}_0(x) + \sum a_k \cdot \hat{v}_k(x)$ when $\{a_k\}$ makes some non-assignment. But if $\hat{t}^{(\mathcal{V})}(x)$ does not divide $\hat{v}(x)$, then $t(x)$ cannot divide $v(x)w(x)$, since $v(x)w(x) = \hat{v}(x) \cdot 1 = \hat{v}(x) \bmod \hat{t}^{(\mathcal{V})}(x)$. by $\hat{t}^{(\mathcal{V})}(x)$. Therefore, $t(x)$ does not divide $v(x)w(x)$, a contradiction. We conclude that there must be no non-assignments of wires, and thus (by the consistency checker) no double assignments either. For each wire $i \in [N]$, $\{a_k\}$ and $\{b_k\}$ must indicate *unequivocal* strongly consistent (no non- or double-) assignments.

We claim that $|\mathcal{I}_{ij}| = 1$ for all $i \in [n], j \in \{0, 1\}$, and moreover that $\{a_k\}$, $\{b_k\}$ 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 $\{k\} = \mathcal{I}_{iu_i'}$. This claim and previous two paragraphs establish that our canonical QSP has the extra property required of strong QSPs: that $t(x)$ divides $v(x)w(x)$ only if the sets $\mathcal{I}_{ij}$ for the input wires are singletons and the linear combinations properly specify an input. To see that the claim is true, notice that we have already established that the linear combinations must be unequivocal in the sense of consistently indicating an assignment (with no non- or double- assignments). The fact that the $\mathcal{I}_{ij}$ are singletons for $i \in [n]$ comes from Lemma 4 and our transformation from f to $f^\dagger$, by which we simplified all of the input gates to AND gates where one of the inputs is always a dummy input set to '1', and from Lemma 2, wherein the SP for a (simplified) AND gate (SPs for AND have the same structure as SPs for NAND) in which one input is a dummy input always set to '1' has a very simple structure – in particular, there is exactly one vector in the simplified SP that must be chosen for the other input (depending on its assignment).

Now, we include the hypothesis that $a_k = 0 = b_k$ for all $k \notin \mathcal{I}_u$. As described above, since $v(x)w(x) = 0 \bmod t(x)$, we must have that $\hat{v}(x) = \hat{v}_0(x) + \sum a_k \cdot \hat{v}_k(x) = 0 \bmod \hat{t}^{(\mathcal{V})}(x)$ – i.e., that our polynomial span program $S^{(\mathcal{V})}_{poly}$ is satisfied. By Lemmas 3 and 5, this implies $f(u) = 1$.

The size of the canonical QSP is $36s$, the same size as the SP S for the circuit checker function. Since $t(x)$ is the product of the three polynomials $t'(x)$, $\hat{t}^{\mathcal{V}}(x)$, and $\hat{t}^{\mathcal{W}}(x)$, of degrees at most $76s$, $27s$ and $27s$ respectively, $t(x)$ has degree $130s$. ■

Our proof of Theorem 3 actually shows something more: that our canonical QSP is 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 equation*

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

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 $\{k\} = \mathcal{I}_{i\bar{u}_i}$.

Theorem 4. *Our canonical QSP is a strong QSP.*

Proof. (Theorem 4) The above proof of Theorem 3 showed that $a_k = 1 = b_k$ for all $\{k\} = \mathcal{I}_{iu_i}$ and $a_k = 0 = b_k$ for all $\{k\} = \mathcal{I}_{i\bar{u}_i}$ for the canonical QSP construction. ■

2.4 QSP Efficiency Considerations

To this point, we have bounded the size and degree of our QSPs. Here, we analyze other efficiency aspects of working with QSPs, before these issues become entangled with the details of our cryptographic protocols. A similar efficiency analysis (see Section 7.5) holds for quadratic arithmetic programs (QAPs), the version of QSPs for arithmetic circuits.

In particular, given a Boolean circuit C, we address the following questions:

1. Given a circuit C, how efficiently can we output the set of polynomials $\{v_k(x)\}$, etc., associated to the QSP that computes C?
2. Given the QSP and an input u, how efficiently can we compute the appropriate linear combination $a_1, \ldots, a_m$ to apply to $\{v_k(x)\}$?
3. Given the linear combination $a_1, \ldots, a_m$, how efficiently can we compute the polynomial $v(x) = \sum_k a_k \cdot v_k(x)$?
4. Letting $w(x) = \sum_k b_k \cdot w_k(x)$, how efficiently can we compute the quotient polynomial $h(x) = (v_0(x) + v(x)) \cdot (w_0(x) + w(x))/t(x)$?

Regarding the first question, if $\{v_k(x)\}$ "behaved" like a set of m "random" polynomials of degree d, we could not hope to compute (or even "write") all of the polynomials in time less than $m \cdot d$. However, in our canonical QSP construction, the polynomials are actually quite structured. In particular, in our canonical QSP, let $\mathcal{R}^{\mathcal{V}}$ be the roots of $\hat{t}^{\mathcal{V}}(x)$ (the target polynomial for the polynomial span program $S^{(\mathcal{V})}_{poly}$), let $\mathcal{R}^{\mathcal{W}}$ be the roots of $\hat{t}^{\mathcal{W}}(x)$, let $\mathcal{R}'$ be the roots of $t'(x)$ (the target polynomial for our consistency checker) and let $\mathcal{R} = \mathcal{R}^{\mathcal{V}} \cup \mathcal{R}^{\mathcal{W}} \cup \mathcal{R}'$ be the roots of $t(x) = \hat{t}^{\mathcal{V}}(x) \cdot \hat{t}^{\mathcal{W}}(x) \cdot t'(x)$ (the target polynomial for our canonical QSP). For any $k \neq 0$, all but a constant number of the values $\{v_k(r) : r \in \mathcal{R}\}$ are zero. Specifically, recall how we set $v_k(x)$ in our canonical QSP:

$$v_k(x) = \begin{cases} \hat{v}_k(x) & \text{mod } \hat{t}^{\mathcal{V}}(x) \\ v'_k(x) & \text{mod } t'(x) \\ 1 & \text{mod } \hat{t}^{\mathcal{W}}(x) \quad \text{if } k = 0 \\ 0 & \text{mod } \hat{t}^{\mathcal{W}}(x) \quad \text{otherwise} \end{cases}$$

Recall that $\hat{v}_k(x)$ comes from $S^{(\mathcal{V})}_{poly}$, and its evaluations at the roots of $\hat{t}^{\mathcal{V}}(x)$ correspond precisely to the vector v_k in the SP S for the circuit checker function $\phi^\dagger$ associated to the function $f^\dagger$. Each vector v_k in the SP S has only a small (constant) number of nonzero coefficients, since it is associated to a single SP S_g for a single gate g in the circuit for $f^\dagger$. Recall that $v'_k(x)$ comes from the aggregate consistency checker. The roots $\mathcal{R}'$ of $t'(x)$ partition into $\cup_{i=1}^{N} \mathcal{R}'_i$, where $\mathcal{R}'_i$ is the set of roots of $t'_i(x) = \prod_{r \in \mathcal{R}'_i}(x - r)$, the target polynomial for the consistency checker associated to wire i. By the construction of the aggregate consistency checker from the individual consistency checkers via CRT, $v'_k(r)$ is nonzero only when, for some i, $k \in \mathcal{I}_{i0} \cup \mathcal{I}_{i0}$ and $r \in \mathcal{R}'_i$. Again, for each k, there are only a constant number of such nonzero evaluations.

So, rather than writing out $v_k(x)$ in coefficient representation, we may instead use a "compressed" representation in terms of its evaluations. In particular, let $\mathcal{J}_k$ be the set of indices j for which $v_k(r_j)$ is nonzero, We will write $v_k(x)$ as $(\mathcal{J}_k, \{c_j^{(k)} \in F : j \in \mathcal{J}_k\})$, where $c_j^{(k)} = v_k(r_j)$. To address the first question above, generating the compressed representations of all of the $v_k(x)$'s and $w_k(x)$'s using $O(s)$ F operations, where s is the number of gates in the circuit, is straightforward.[4]

Regarding the second question, we need to evaluate the constant-size gate-specific span programs as we traverse the circuit. This has complexity $O(s)$. More specifically, for our current SP implementation of NAND gates as described in the proof of Lemma 2, within each NAND gate, for each possible assignment of the wires, we pick exactly one vector for each wire. Since the circuit $C^\dagger$ for $f^\dagger$ has at most $3s$ gates, there are at most $9s$ nonzero coefficients in the linear combination $\{a_k\}$, and all of the nonzero coefficients are small – in particular, they equal 1.

Regarding the third question above, since each polynomial $v_k(x)$ with $k \neq 0$ has only a constant number of nonzero evaluations over $\mathcal{R}$, they "behave" like Lagrange basis polynomials $\ell_j(x) = \prod_{i \neq j}(x - r_i)/(r_j - r_i)$. For any $(a_1, \ldots, a_d) \in F^d$, one can output a representation of $u(x) = \sum_{j \in [d]} a_j \cdot \ell_j(x)$ in terms of its evaluations at $\{r_j\}$ using only $O(d)$ F operations. Similarly, since each of our $v_k(x)$'s has only a constant number of nonzero evaluations over $\mathcal{R}$, one can output a representation of $v(x) = \sum_{k \in [m]} a_k \cdot v_k(x)$ using only $O(m)$ F operations. We state this more formally below.

Lemma 8. *For $k \in [m]$, let $a_k \in F$, and let $v_k(x)$ be a polynomial of degree at most $d-1$, represented as $(\mathcal{J}_k, \{c_j^{(k)} \in F : j \in \mathcal{J}_k\})$, where $v_k(x) = \sum_{j \in \mathcal{J}_k} c_j^{(k)} \cdot \ell_j(x)$ and $\ell_j(x) = \prod_{i \in [d], i \neq j}(x - r_i)/(r_j - r_i)$*

[4]Technically, each index in $\mathcal{J}_k$ has $O(\log s)$ bits, so these indices take $\theta(s \log s)$ space. We ignore this bookkeeping technicality, since representing each F element takes $\Omega(\log s)$ space anyway.

is a Lagrange basis polynomial. Let $n = \sum_k |\mathcal{J}_k|$. *There is an algorithm that outputs* $v(x) \leftarrow \sum_{k\in[m]} a_k \cdot v_k(x)$, *represented by its evaluations at* $(r_1, \ldots, r_d)$, *using* $O(d+n)$ *field operations.*

Proof. Initialize an array $A = (A_1, \ldots, A_d)$ that will hold $v(x)$'s evaluations at $r_1, \ldots, r_d$. For $k = 1$ to m, and for $j \in \mathcal{J}_k$, set $A_j \leftarrow A_j + a_k \cdot c_j^{(k)}$. Output A. The algorithm uses $O(d+n)$ field operations. ■

Theorem 5. *In the canonical QSP, we can output* $v(x)$ *and* $w(x)$ *using* $O(s)$ *field operations.*

Proof. In our canonical QSP, $d = O(s)$ and $\max_k |\mathcal{J}_k|$ is constant. The theorem follows. ■

Regarding the fourth question, once we have $v(x)$ and $w(x)$, we can compute the quotient polynomial $h(x) = (v_0(x)+v(x))\cdot(w_0(x)+w(x))/t(x)$ in time $\tilde{O}(d)$, where d is the maximum degree of the polynomials, using generic techniques. For example, let $\mathcal{R}_2$ be a set of d values distinct from the roots $\mathcal{R}$ of $t(x)$. Using fast-Fourier evaluation, we can compute the evaluations of $v(x)$ and $w(x)$ (the evaluations of $v_0(x)$, $w_0(x)$ and $t(x)$ are pre-computed), and hence the evaluations of $h(x)$ in quasi-linear time. Computing $h(x)$ via multipoint evaluation and interpolation is faster – e.g., $O(s \log s)$ versus $O(s \log^2 s)$ – when $\mathcal{R}_2$ consists of D-th roots of unity for some $D = O(d) = O(s)$. However, when using QSPs in our cryptographic constructions, some care must be taken that using F with such roots of unity does not weaken security.

Computing $h(x)$ in purely linear, versus quasi-linear, time remains an intriguing open problem. One may state a more generic version of the problem in terms of Lagrange basis polynomials as follows. Let $r_1, \ldots, r_d \in F$ be distinct, let $t(x) = \prod(x - r_i)$ and let $\ell_j(x) = \sum_{i\neq j}(x - r_i)/(r_j - r_i)$ be the Lagrange basis polynomials. Given $(a_1, \ldots, a_d) \in \{0,1\}$, output $(\sum a_j \cdot \ell_j(x)) \cdot (\sum \bar{a_j} \cdot \ell_j(x))/t(x)$ in linear time. (Here $\bar{a_j} = 1 - a_j$. Note that $(\sum a_j \cdot \ell_j(x)) \cdot (\sum \bar{a_j} \cdot \ell_j(x))$ will always be evenly divisible by $t(x)$. It would be fine to output $h(x)$ expressed in terms of its evaluations.)

3 Overview of Our Cryptographic Constructions and Security

In this section, we give a high-level overview of our QSP-based cryptographic constructions, while Section 4 provides formal definitions and detailed constructions.

3.1 Our SNARK Construction

Let $R = \{(u, w)\}$ be a NP relation with n'-bit statements and $(n - n')$-bit witnesses, and let $f(u, w) = 1$ iff $(u, w) \in R$. Let $Q_f = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}, \mathcal{I} = \mathcal{I}_{free} \cup \mathcal{I}_{labeled})$ be a strong QSP for f over F of some size m and degree d.

In our SNARK for the relation R, the polynomials (e.g., $\{v_k(x)\}$) are represented by encodings of these polynomials evaluated at some secret point (e.g., $\{E(v_k(s))\}$), where E is the encoding function of an additively homomorphic encoding scheme $\mathcal{E}$ for F-elements. Our preferred encoding is exponentiation within a bilinear group ($E(v_k(s)) = g^{v_k(s)}$), but one may also use an additively homomorphic encryption scheme such as Paillier ($E(v_k(s)) = Enc_{pk}(v_k(s))$). In the latter case, the verifier needs a secret key sk to remove the encoding, and hence the SNARK is designated-verifier. In the former case, $sk = \perp$ and a public verifier can use the bilinear map to verify.

CRS generation Gen**:** On input security parameter κ and upper bound d on the degree of the strong QSP for the functions f that will be computed, run $(pk, sk) \leftarrow \mathcal{E}.\mathtt{Setup}$, generate uniformly at random $\alpha, s \leftarrow F^*$ and output $\mathsf{priv} = sk$, $\mathsf{crs} = (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]})$.

Function specific CRS generation Regen**:** On input crs, a function f with a strong QSP Q_f as above, and n' as above, let $\mathcal{I}_{in} = \cup_{i=1}^{n'} \mathcal{I}_{ij}$ and $\mathcal{I}_{mid} = \mathcal{I} \setminus \mathcal{I}_{in}$, and generate uniformly at random $\beta_v, \beta_w, \gamma \leftarrow F^*$, and output:

$$\begin{aligned}
\mathsf{crs}_f &= (\mathsf{crs}, Q_f, n', \{E(v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(w_k(s))\}_{k\in\mathcal{I}}, \{E(s^i)\}_{i\in[0,d]}, \\
&\quad \{E(\alpha v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\alpha w_k(s))\}_{k\in\mathcal{I}}, \{E(\alpha s^i)\}_{i\in[0,d]}, \\
&\quad \{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in\mathcal{I}}). \\
\mathsf{shortcrs}_f &= (\mathsf{priv}, E(1), E(\alpha), E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma), \{E(v_k(s))\}_{k\in\{0\}\cup\mathcal{I}_{in}}, E(w_0(s)), E(t(s))).
\end{aligned}$$

Prove P**:** On input crs_f, 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

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

For $v_{mid}(x) = \sum_{k\in\mathcal{I}_{mid}} a_k \cdot v_k(x)$ and $w(x) = \sum_{k\in\mathcal{I}} b_k \cdot w_k(x)$, P outputs:

$$\pi = (\ \ E(v_{mid}(s)), E(w(s)), E(h(s)),\ \ E(\alpha v_{mid}(s)), E(\alpha w(s)), E(\alpha h(s)),\ \ E(\beta_v v_{mid}(s) + \beta_w w(s))\ \ ).$$

Verify V**:** On input $\mathsf{shortcrs}_f$, sk, 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 $E(v_{in}(s))$ of $v_{in}(s) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(s)$, using the fact that, in a strong QSP, $\{a_k : k \in \mathcal{I}_{in}\}$ can be computed efficiently and deterministically from u. V confirms that the following equations hold: $(v_0(s) + v_{in}(s) + V_{mid}) \cdot (w_0(s) + W) = H \cdot t(s)$, $V'_{mid} = \alpha V_{mid}$, $W' = \alpha W$, $H' = \alpha H$, and $\gamma Y = (\beta_v\gamma)V_{mid} + (\beta_w\gamma)W$.

3.2 Making the SNARK Statistical Zero-Knowledge (NIZKs)

To facilitate randomization in our ZK construction, we add $E(t(s))$, $E(\alpha t(s))$, $E(\beta_v t(s))$, $E(\beta_w t(s))$, $E(v_0(s))$, $E(\alpha v_0(s))$, $E(w_0(s))$ and $E(\alpha w_0(s))$ to crs_f. To randomize its original SNARK from above, the prover picks random $\delta_{v_{mid}}, \delta_w \leftarrow F$. For $v_{mid}(x)$, $w(x)$, $h(x)$ as above, the proof is:

$$E(v'_{mid}(s)), E(w'(s)), E(h'(s)),\quad E(\alpha v'_{mid}(s)), E(\alpha w'(s)), E(\alpha h'(s)),\quad E(\beta_v v'_{mid}(s) + \beta_w w'(s)),$$

where $v'_{mid}(x) = v_{mid}(x) + \delta_{v_{mid}} t(x)$ and $w'(x) = w(x) + \delta_w t(x)$. The new value of the quotient polynomial is $h'(x) = h(x) + \delta_{v_{mid}} w^\dagger(x) + \delta_w v^\dagger(x) + \delta_{v_{mid}}\delta_w t(x)$, for $v^\dagger(x) = v_0(x) + \sum_{k\in\mathcal{I}} a_k v_k(x)$ and $w^\dagger(x) = w_0(x) + w(x)$. All of these new values can be computed efficiently from crs_f. Theorem 6 in Section 4.4 proves that the new SNARK, still only 7 elements, is statistical ZK for suitable encoding schemes, such as the bilinear-group encoding scheme.

3.3 Performance and Optimizations

Our NIZK consists of only 7 group elements. Prover computation requires a linear number of group operations, aside from the computation of $h(x)$. As detailed in Section 2.4, $v_0(x) + \sum a_k v_k(x)$ and $w_0(x) + \sum b_k w_k(x)$ can be computed with a linear number of F-ops due to the structure of the polynomials in $\mathcal{V}, \mathcal{W}$. Then, the quotient $h(x) = (v_0(x) + \sum a_k v_k(x)) \cdot (w_0(x) + \sum b_k w_k(x))/t(x)$ can be computed in $O(s \cdot \mathrm{polylog}(s))$ time via multipoint evaluation and interpolation. (In [32, 38], prover computation is quadratic.)

Verifier computation is linear in the size of the statement (not witness). In particular, V's main cost is computing an encoding $E(v_{in}(s))$ of $v_{in}(s) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(s)$. We can reduce this cost. In particular, we can make V's computation constant (independent of $|u|$), except that V must compute $u' = H(u)$, where H is an ordinary (not necessarily extractable) collision-resistant hash function, like SHA-256. In effect, u' becomes the new statement, and u becomes part of the witness (see Section 4.5.1). We can also reduce verifier computation by using PCPs in a targeted way: only to prove that the prover computed $E(v_{in}(s))$ correctly (Section 4.5.2).

3.4 Security

In an earlier version of our scheme, for random $\alpha_v, \alpha_w, \alpha_h$, a SNARK was just 6 elements:

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

We modified the scheme to base security on assumptions comparable to (actually slightly weaker than) Groth's [32]. We first consider how the security proof for this earlier version would go.

For the earlier version, 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 the prover *knows* a representation $\{b_k : k \in \mathcal{I}\}$ of W such that $W = \sum b_k w_k(s)$. Knowledge of exponent assumptions (KEAs)[5] 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 that outputs the representation (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 $(v_0(s) + v(s)) \cdot (w_0(s) + w(s)) = h(s) \cdot t(s)$ for $v(x) = v_{mid}(x) + \sum_{k \in \mathcal{I}_{in}} v_k(x)$. If indeed $(v_0(x) + v(x)) \cdot (w_0(x) + w(x)) = h(x) \cdot t(x)$ as polynomials, the soundness of our QSP implies that we have extracted a *true* proof. Otherwise, $(v_0(x) + v(x)) \cdot (w_0(x) + w(x)) - h(x) \cdot t(x)$ is a nonzero polynomial having s as a root, which allows the simulator to solve a hard problem.

In our current scheme, the terms $E(\alpha v_{mid}(s))$, $E(\alpha w(s))$, $E(\alpha h(s))$ are used only to extract representations of the encoded terms with respect to the power basis $\{x^i\}$, not with respect (e.g.) to $\{v_k(x) : k \in \mathcal{I}_{mid}\}$. This extraction does not guarantee that $v_{mid}(x)$ and $w(x)$ are in their proper spans. Instead, we ensure this via the final term $E(\beta_v v_{mid}(s) + \beta_w w(s))$, from which the simulator can solve a hard problem if $v_{mid}(x)$ or $w(x)$ lies outside its proper span.

4 Cryptographic Constructions from Strong QSPs

4.1 Definitions: SNARGs, SNARKs, Verifiable Computation, NIZKs

We follow the definition of succinct non-interactive arguments (SNARGs) given by Gentry and Wichs [27].[6]

A SNARG system Π consists of three efficient machines $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$. The generation algorithm $(\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa)$ produces a common reference string crs along with some private verification state priv. The prover algorithm $\pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)$ produces a proof π for a statement u using a witness w. The verification algorithm $\mathsf{V}(\mathsf{priv}, u, \pi)$ decides if π is a valid proof for u, using the private verification state priv.

[5] KEAs [6, 19, 28, 35] exist for Paillier/RSA [20, 28], bilinear groups [32, 38], and even lattices [39].

[6] This differs from the re-definition of SNARGs by Bitansky et al. [8], which we would call SNARGs with fast verification.

4.1.1 Succinct Non-Interactive Arguments

Definition 6 (SNARG)**.** *We say that* $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *is a* succinct non-interactive argument *(*SNARG*) for an* NP *language* L *with a corresponding* NP *relation* R*, if it satisfies the following three properties:*

Completeness: *For all* $(u, w) \in R$,
$$\Pr\left[\mathsf{V}(\mathsf{priv}, u, \pi) = 0 \;\middle|\; \begin{array}{c}(\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa) \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)\end{array}\right] = \mathtt{negl}(\kappa).$$

Soundness: *For all efficient* $\overline{\mathsf{P}}$,
$$\Pr\left[\begin{array}{c}\mathsf{V}(\mathsf{priv}, u, \pi) = 1 \\ \wedge \quad u \notin L\end{array}\;\middle|\; \begin{array}{c}(\mathsf{crs}, \mathsf{priv}) \leftarrow \mathsf{Gen}(1^\kappa) \\ (u, \pi) \leftarrow \overline{\mathsf{P}}(1^\kappa, \mathsf{crs})\end{array}\right] = \mathtt{negl}(\kappa).$$

Succinctness: *The length of a proof is given by* $|\pi| = \mathrm{poly}(\kappa)\mathrm{polylog}(|u| + |w|)$.

Public vs. Designated Verifiability. We say that a SNARG is *publicly verifiable* if the private verification state is just $\mathsf{priv} = \mathsf{crs}$. In that case, proofs can be verified by all parties. Otherwise, we call it a *designated-verifier* SNARG, in which case only the party that knows priv can verify proofs.

Definition 7 (SNARK)**.** *A* succinct non-interactive argument of knowledge *(*SNARK*) is a* SNARG *that comes together with an extractor* $\mathcal{E}$*. In particular, for any statement* u*, we require that there be a polynomial-time extractor* $\mathcal{E}_u$ *such that, for any* $\pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)$, $w \leftarrow \mathcal{E}_u(\mathsf{priv}, \pi)$.

Definition 8 (Zero-Knowledge SNARK)**.** *A SNARK for an* NP *language* L *with a corresponding* NP *relation* R *is zero-knowledge, if there exists a simulator* (S_1, S_2) *such that* S_1 *outputs a simulated CRS* crs *and a trapdoor* τ*,* S_2 *takes as input* crs*, a statement* x *and* τ *and outputs a simulated proof* π*. Now for all adversaries* $\mathcal{A}$ *it holds that*

$$Pr[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\kappa);\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow P(\mathsf{crs}, u, w);\ :\ (x, w) \in R \text{ and } \mathcal{A}(\pi) = 1] \approx$$
$$Pr[(\mathsf{crs}, \tau) \leftarrow S_1(1^\kappa);\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow S_2(\mathsf{crs}, u, \tau);\ :\ (x, w) \in R \text{ and } \mathcal{A}(\pi) = 1]$$

SNARKs with Fast Verification. We say that a SNARG or SNARK has *fast verification*, or is *unsubtle*, if the running time of $\mathsf{V}(\mathsf{priv}, u, \pi)$ is $\mathrm{poly}(\kappa)\mathrm{polylog}(|u| + |w|)$. Otherwise, it is a *subtle* SNARK.

4.1.2 Verifiable Computation

Verifiable computation (VC) [24], similar to a delegation scheme [30], addresses the setting where a computationally limited client outsources the evaluation of a function F on input u to another party, a worker. The goal here is to enable the client to verify the correctness of the returned result $F(u)$ performing less work than required for the function evaluation itself. A VC scheme provides public verifiability [43] when anyone can verify the worker's output using only public information.

Definition 9 (Verifiable Computation)**.** *A* verifiable computation scheme (with preprocessing) $\mathcal{VC}$ *is a four-tuple of polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *which work as follows:*

- $(SK, PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\kappa)$*: The randomized* key generation *algorithm takes as input a security parameter* κ *and the function* F*, and outputs a secret key* SK*, a public key* PK_F *and an evaluation key* EK_F*.*
- $(\sigma_u, VK_u) \leftarrow \mathsf{ProbGen}(PK_F, u)$*: The randomized* problem generation *algorithm uses the public key* PK_F *to encode an input* u *into an input encoding* σ_u*, which is given to the worker and a private verification key* VK_u*.*

- $\sigma_y \leftarrow \mathsf{Compute}(EK_F, \sigma_u)$*: The deterministic* worker *algorithm uses the evaluation key* EK_F *together with the value* σ_u *to compute a value* σ_y*.*
- $y \leftarrow \mathsf{Verify}_{SK}(VK_u, \sigma_y)$*: The deterministic* verification *algorithm uses the verification key* VK_u *and the worker's output* σ_y *to compute a string* $y \in \{0,1\}^* \cup \{\perp\}$*. Here, the special symbol* $\perp$ *signifies that the verification algorithm rejects the worker's answer* σ_y*.*

Public vs. Designated Verifiability. We refer to the above definition of verifiable computation as designated verifier (DV) since the verification key VK_u is kept private and provided only to the designated verifier. A VC scheme is called *publicly verifiable* if the verification key VK_u can be made public and the secret key SK is empty.

A verifiable computation scheme needs to satisfy three properties: *correctness, security and efficiency.* The correctness property guarantees that if the worker performs the evaluation honestly, the verification check will hold. The efficiency requirement states that the complexity of the delegation $\mathsf{ProbGen}$ and the verification Verify algorithms together is less than the computation required for the evaluation of F. Finally, the security of a VC scheme is defined as follows:

Definition 10 (Security)**.** *Let* $\mathcal{VC}$ *be a verifiable computation scheme for a class of functions* $\mathcal{F}$*, and let* $A = (A_1, A_2)$ *be any pair of probabilistic polynomial time machines. Consider the experiment* $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, F, \kappa]$ *for any* $F \in \mathcal{F}$ *below:*

Experiment $\mathbf{Exp}_A^{Verif}[\mathcal{VC}, F, \kappa]$
$(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\kappa)$*;*
For $i = 1, \ldots, l = poly(\kappa)$
$(u_i, \mathsf{priv}) \leftarrow A_1(PK_F, EK_F)$*;*
$(\sigma_{u_i}, VK_{u_i}) \leftarrow \mathsf{ProbGen}(PK_F, u_i)$*;*
$\sigma_{\mathsf{out}_i} \leftarrow A_2(\mathsf{priv}, \sigma_{u_i})$*;*
$y_i \leftarrow \mathsf{Verify}(VK_{u_i}, \sigma_{\mathsf{out}_i})$
If $\exists j \in [1, l]$ *where* $y_j \neq \perp$ *and* $y_j \neq F(u_j)$*, output '1', else output '0';*

A verifiable computation scheme $\mathcal{VC}$ *is* secure *for a class of functions* $\mathcal{F}$*, if for every function* $F \in \mathcal{F}$ *and every p.p.t. adversary* $A = (A_1, A_2)$*:*

$$Pr[\mathbf{Exp}_A^{Verif}[\mathcal{VC}, F, \kappa] = 1] \leq \mathtt{negl}(\kappa). \tag{4}$$

where $\mathtt{negl}(\cdot)$ *denotes a negligible function of its input.*

The security definition for a publicly verifiable computation scheme is the same as the one above except for the fact that the algorithm A_2 is given as input also VK_{u_i}, and it suffices to consider the delegation of a single input in the security experiment.

4.1.3 Non-Interactive Zero Knowledge

Let R be a binary relation which consists of pairs (u, w), where u is a statement and w is a witness. Let f be a function such that $f(u, w) = 1$ iff $(u, w) \in R$. Let L be the language that consists of statements with valid witnesses for R. A non-interactive zero knowledge argument for the relation R, or the function f, consists of the triple of polynomial time algorithms (K, P, V):

- K takes a security parameter κ as well as the maximum size of a statement n and outputs a common reference string crs;

- P takes as input the CRS crs, a statement u and a witness w, and outputs an argument π;
- V takes as input the CRS crs, a statement u and its proof π, and either accepts or rejects the proof.

A non-interactive zero-knowledge argument for R satisfies the following three properties:

Completeness: This property refers to the fact that a prover can always provide a valid proof for a statement with witness. Formally for every adversary $\mathcal{A}$ the following holds

$$Pr[\mathsf{crs} \leftarrow K(1^\kappa, n);\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow P(\mathsf{crs}, u, w);\ :\ V(\mathsf{crs}, u, \pi) = 1 \text{ if } (u, w) \in R] = 1.$$

Soundness: This property guarantees that the prover can give a proof that verifies for a false statement only with negligible probability. Formally for every adversary $\mathcal{A}$ the following holds

$$Pr[\mathsf{crs} \leftarrow K(1^\kappa, n);\ (u, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\ : V(\mathsf{crs}, u, \pi) = 1 \text{ and } u \notin L] < \mathtt{negl}(\kappa).$$

Zero-knowledge: This property guarantees that the proof reveals nothing more than the correctness of the statement. For this purpose we show that it is possible to simulate the proof for a valid statement without knowing a witness. Formally there exists a simulator (S_1, S_2) such that S_1 output a simulated CRS crs and trapdoor τ, S_2 takes as input crs, a statement u and τ and output a simulated proof π. Now for all adversaries $\mathcal{A}$ it holds that

$$Pr[\mathsf{crs} \leftarrow K(1^k, n);\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow P(\mathsf{crs}, u, w);\ :\ (u, w) \in R \text{ and } \mathcal{A}(\pi) = 1] \approx$$
$$Pr[(\mathsf{crs}, \tau) \leftarrow S_1(1^k, n);\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs});\ \pi \leftarrow S_2(\mathsf{crs}, u, \tau);\ :\ (u, w) \in R \text{ and } \mathcal{A}(\pi) = 1]$$

4.2 Verifiable Computation from Designated-Verifier SNARKs

The following construction shows how to instantiate a verifiable computation scheme from a designated verifier SNARK with fast verification.

Construction 1. *Let* $\Pi = (\mathsf{Gen}, \mathsf{Regen}_f, \mathsf{P}, \mathsf{V})$ *be a designated verifier SNARK.*

Key Generation KeyGen**:** *On input a function* $F : \{0,1\}^n \to \{0,1\}^m$ *construct a function* $f(x, w)$ *for a relation* R *such that* $f(x, w) = 1$ *if and only if* $F(x_{[1,n]}) = x_{[n+1,n+m]}$ *where* x *has* $n+m$ *bits,* $x_{[1,n]}$ *denotes the first* n *bits of* x *and* $x_{[n+1,n+m]}$ *denotes the last* m *bits of* x*. Run the SNARK algorithms* Gen *and* Regen_f *and output:*

- priv *a secret key* SK*;*
- crs *and* crs_f *as a public evaluation key* EK_F*;*
- $\mathsf{shortcrs}$ *as a public key* PK_F*.*

Delegation ProbGen**:** *On input* $u \in \{0,1\}^n$ *to* F*, the delegator sends* u *to the worker.*

Computation Compute**:** *The worker evaluates* $y = F(u)$ *and computes corresponding witness* w*. He runs* P *from the SNARK construction with input* $x = u||y$ *and witness* w *to compute a proof* π *and returns* y *and* π *to the delegator.*

Verification Verify_{SK}**:** *The delegator runs the SNARK verification* $\mathsf{V}(\mathsf{priv}, x, \pi)$ *to accept the output.*

If we start with a publicly verifiable SNARK where priv is empty, we obtain a publicly verifiable VC scheme since we do not need to have secret key SK. The scheme uses a *per-function verification key* PK_F, rather than a *per-input verification key*, as in Parno et al. [43].

Falsifiable vs. Non-falsifiable Assumptions. Gentry and Wichs showed that a designated-verifier SNARG/SNARK for NP languages cannot be based (via black box reductions) on falsifiable assumptions [27]. Moreover, they observe that the assumption "this SNARG construction is secure" is itself non-falsifiable [27], since in order for a challenger to decide whether the adversary successfully produces a proof of a false statement, the challenger needs to decide whether a NP statement is true or false, which may not be efficiently decidable. However, for languages in P, the assumption that "this SNARG construction is secure" is, in fact, falsifiable: a challenger can efficiently distinguish whether the adversary forged a proof of a false statement. Verifiable computation schemes are only required to work for languages in P. Therefore, one can transform any SNARG with fast verification (ours or, say, Micali's PCP-based construction [40]) into a robust verifiable computation schemes based on a falsifiable assumption. However, more work is needed to base robust verifiable computation on *natural* falsifiable assumptions.

4.3 Construction of Designated-Verifier/Public-Verifier SNARKs from Strong QSPs

In this section, we describe how to construct SNARKs from strong QSPs. The SNARKs may be designated-verifier or public-verifier, depending on what sort of "encoding" scheme the construction uses. From these SNARKs, we obtain verifiable computation via the general transformation described in Section 4.2 and NIZKs via the transformation described in Section 4.4.

We present our construction using schemes for encoding elements of the field/ring F over which the QSPs are defined. We will assume here that $|F|$ has size exponential in the security parameter. Two example encoding schemes to keep in mind are Paillier (or, more generally, additively homomorphic) encryption, and exponentiation within a bilinear group. We periodically elaborate on how to instantiate our abstract constructions with concrete encodings.

An encoding scheme $\mathcal{E}$ has two algorithms (Setup, E). The Setup function generates some public parameters or a public key pk, and may also generate some secret state sk. $\mathcal{E}$ has also a (possibly probabilistic) encoding function $E : F \times R \to S$ (where R is a space of random bits) mapping F-elements to the set S of encoded F-elements. For each $s \in S$, it must be well-defined which element it encodes – that is, the sets $\{\{E(a)\} : a \in F\}$, where $\{E(a)\} = \{E(a, r) : r \in R\}$, partition S. For convenience, we will often write $E(a)$ with the understanding that this is some element $E(a, r) \in S$ for some $r \in R$. The encoding scheme $\mathcal{E}$ has the following additional properties.

- Additively homomorphic: Given pk, and any $E(a_1) \in \{E(a_1)\}$ and $E(a_2) \in \{E(a_2)\}$, one can efficiently compute some $E(a_1 + a_2) \in \{E(a_1 + a_2)\}$.
- Quadratic root detection: Given $param$, $E(a_1), \ldots, E(a_t)$, and the quadratic polynomial $Q(x_1, \ldots, x_t) \in F[x_1, \ldots, x_t]$, one can distinguish whether $Q(a_1, \ldots, a_t) = 0$.
- Image verification: Given $param$, one can distinguish whether a term c is in the set $\{\{E(a)\} : a \in F\}$, i.e. whether it validly encodes some element of F.

The parameters *param* needed for quadratic root detection and image verification may include just the public key pk, or it may also include a secret key sk. In the former case, the resulting construction will be public-verifier; in the latter, it will be designated-verifier. The quadratic root detection property is stated more generally than we need: in our QSP-based constructions, we only require root detection for particular quadratic polynomials. The image verification property has also been needed in previous applications of extractable primitives [6, 14, 35], but not always [8].

Next we describe two possible instantiations for the encoding scheme.

Example 1. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of the same prime order p, and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. Let g be a generator of $\mathbb{G}$. We implement the encoding scheme as: $E(a, r) = g^a$. This encoding has the desired properties:*

- *Additively homomorphic: Compute $E(a_1 + a_2)$ as $E(a_1)E(a_2) = g^{a_1+a_2}$.*
- *Quadratic root detection: Given $E(a_1), \ldots, E(a_t) = g^{a_1}, \ldots, g^{a_t}$, use the bilinear map to compute $e(g, g)^{Q(a_1,\ldots,a_t)}$ and check whether it equals the identity element in $\mathbb{G}_T$.*
- *Image verification: Typically, it is straightforward to determine whether an element is in the group $\mathbb{G}$, and all elements of $\mathbb{G}$ are valid encodings.*

Note that none of these properties requires secret state sk. Note also that this encoding scheme is deterministic.

Example 2. *Let (Gen, Enc, Dec) be a randomized additively homomorphic encryption system, such as Paillier. For* Setup*, run Gen to produce (pk, sk). Implement the encoding scheme as: $E(a, r) = Enc(pk, a, r)$. This encoding has the desired properties:*

- *Additively homomorphic: By definition.*
- *Quadratic root detection: Use sk to decrypt the $E(a_i)$ values; check whether $Q(a_1, \ldots, a_t) = 0$.*
- *Image verification: For Paillier, every element modulo N^2 is a valid ciphertext, except elements whose GCD with N is nontrivial, which are easy to detect.*

Note that quadratic root detection requires sk.

Technically, the plaintext space of Paillier is a ring, not a field. Our constructions and proofs apply only where the encoding space is a field. However, we believe that it would be easy to extend our results to Paillier, using the fact (a fact used, for example, by the elliptic curve factoring algorithm) that one is unlikely to encounter encodings of nontrivial zero divisors in $\mathbb{Z}_N$ unless one is able to factor N.

We now define our SNARK system $\Pi = (\mathsf{Gen}, \mathsf{Regen}_f, \mathsf{P}, \mathsf{V})$ for the function $f : \{0,1\}^n \to \{0,1\}$, which enables succinct arguments, that for specified u, there exists w such that $f(u, w) = 1$. We will refer to u as the "statement" and w as the "witness". In the description of our SNARK, the indices $i \in [1, n']$ will correspond to the statement u, and the positions $[n' + 1, n]$ will correspond to the witness w.

We split the generation algorithm into two parts, Gen and Regen, where the former is (almost) independent of particular function f – i.e., Gen only needs an upper bound on the circuit size of f.

CRS generation Gen**:** On input the security parameter κ and an upper bound, d, on the degree of the strong QSP for the functions f that will be computed, run $(pk, sk) \leftarrow \mathcal{E}.\mathtt{Setup}$, generate uniformly at random $\alpha, s \leftarrow F^*$ and output $\mathsf{priv} = sk$, $\mathsf{crs} = (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]})$.

Function specific CRS generation Regen**:** On input crs, a function f with a strong QSP

$$Q_f = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij}).$$

of size m and degree at most d, and value $n' \in [1, n]$, let $\mathcal{I}_{in} = \cup_{i\in[1,n'],j\in[0,1]}\mathcal{I}_{ij}$, let $\mathcal{I}_{mid}$ be all of the remaining indices $\{1, \ldots, m\} \setminus \mathcal{I}_{in}$, generate uniformly at random $\beta_v, \beta_w, \gamma \leftarrow F^*$, and output:

- Public common reference string for f, n':

$$\begin{aligned}\mathsf{crs}_f = (\mathsf{crs}, Q_f, n', &\{E(v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(w_k(s))\}_{k\in[m]}, \{E(s^i)\}_{i\in[0,d]},\\ &\{E(\alpha v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\alpha w_k(s))\}_{k\in[m]}, \{E(\alpha s^i)\}_{i\in[0,d]},\\ &\{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in[m]}).\end{aligned}$$

- A short CRS that will be used for verification

$$\mathsf{shortcrs}_f = (\mathsf{priv}, E(1), E(\alpha), E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma), \{E(v_k(s))\}_{k\in\{0\}\cup\mathcal{I}_{in}}, E(w_0(s)), E(t(s))).$$

Prove P**:** On input crs_f, statement $u \in \{0,1\}^{n'}$, and witness w, P will prove that $(u, w) \in R$ – i.e., that $f(u, w) = 1$. The prover evaluates Q_f to obtain $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_m)$ such that

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

For $v_{mid}(x) = \sum_{k\in\mathcal{I}_{mid}} a_k \cdot v_k(x)$ and $w(x) = \sum_{k\in[m]} b_k \cdot w_k(x)$, the prover outputs the following proof π:

$$E(v_{mid}(s)), E(w(s)), E(h(s)), \quad E(\alpha v_{mid}(s)), E(\alpha w(s)), E(\alpha h(s)), \quad E(\beta_v v_{mid}(s) + \beta_w w(s)).$$

Verify V**:** On input $\mathsf{shortcrs}_f$, sk, u, and proof $\pi = (\pi_{v_{mid}}, \pi_w, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \pi_y)$, 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 $E(v_{in}(s))$ of $v_{in}(s) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(s)$, using the fact that, in a strong QSP, $\{a_k : k \in \mathcal{I}_{in}\}$ can be computed efficiently and deterministically from u. Using quadratic root detection, V confirms that the following equations hold: $(v_0(s) + v_{in}(s) + V_{mid}) \cdot (w_0(s) + W) - H \cdot t(s) = 0$, $V'_{mid} - \alpha V_{mid} = 0$, $W' - \alpha W = 0$, $H' - \alpha H = 0$, and $\gamma Y - (\beta_v\gamma)V_{mid} - (\beta_w\gamma)W = 0$.

Public vs. Designated Verifiability. Whether we achieve a designated verifier SNARK or a public verifier SNARK in the above construction depends on the instantiation of the encoding scheme. Specifically it depends on whether we require access to the encoding's secret key for quadratic root detection. If we instantiate the encoding with pairings, we can perform quadratic root detection without any secret information, using the pairing operation, and thus we obtain a public-verifier (PV) SNARK. If we instantiate the encoding with additive homomorphic encryption, then we need the secret key to detect quadratic roots, so the resulting SNARK construction is designated-verifier (DV).

Our use of the term designated-verifier in this paper will be informal – in particular, we make no requirement that a DV SNARK be hard to verify for parties other than the designated verifier, since this property is not of interest to us here. In our parlance, a PV SNARK can be viewed as a DV SNARK. Typically, however, we will use the term designated verifier to emphasize that the verification algorithm V takes some private state sk that is not included in pk.

Verifier Computation. Regarding verifier computation, note that in our canonical strong QSP we make sure that each input is associated to only one index k. Therefore, the verifier's work in computing $E(v_{in}(s))$ is proportional only to the size $|u|$ of the statement. (This work is otherwise independent of the witness size and circuit size.) Of course, the verifier's remaining work after computing $E(v_{in}(s))$ is constant.

4.4 Zero-Knowledge SNARKs (including NIZKs) from Strong QSPs: How to Randomize Our SNARKs

To make our SNARKs zero-knowledge (ZK), the idea is simply to randomize the elements of the SNARK proof to make them statistically indistinguishable from a set of random (encoded) elements that satisfy the quadratic checks.

To facilitate the randomization in our ZK construction, we include additional terms $E(t(s))$, $E(\alpha t(s))$, $E(\beta_v t(s))$, $E(\beta_w t(s))$, $E(v_0(s))$, $E(\alpha v_0(s))$, $E(w_0(s))$ and $E(\alpha w_0(s))$ in crs_f. The prover proceeds as before, except that it additively perturbs $E(v_{mid}(s))$ and $E(w(s))$ by random multiples of $E(t(s))$, and modifies the other values accordingly.

More specifically, the values of $v_{mid}(x)$, $w(x)$, and $h(x)$ are as before. The prover picks random $\delta_{v_{mid}}, \delta_w \leftarrow F$. Its proof is:

$$E(v'_{mid}(s)), E(w'(s)), E(h'(s)), \quad E(\alpha v'_{mid}(s)), E(\alpha w'(s)), E(\alpha h'(s)), \quad E(\beta_v v'_{mid}(s) + \beta_w w'(s)),$$

where $v'_{mid}(x) = v_{mid}(x) + \delta_{v_{mid}} t(x)$ and $w'(x) = w(x) + \delta_w t(x)$. Regarding the value of $h'(x)$, let $v'(x) = v_0(x) + v_{in}(x) + v'_{mid}(x) = v^\dagger(x) + \delta_{v_{mid}} t(x)$, where $v^\dagger(x) = v_0(x) + v_{in}(x) + v_{mid}(x)$. Define $w'(x)$ and $w^\dagger(x)$ similarly. We have:

$$\begin{aligned} h'(x) &= v'(x) \cdot w'(x)/t(x) \\ &= (v^\dagger(x) + \delta_{v_{mid}} t(x)) \cdot (w^\dagger(x) + \delta_w t(x))/t(x) \\ &= h(x) + \delta_{v_{mid}} w^\dagger(x) + \delta_w v^\dagger(x) + \delta_{v_{mid}} \delta_w t(x). \end{aligned}$$

Notice that the encodings of all of the values in this new proof can be computed efficiently from crs_f (the same asymptotic efficiency as before).

We want to argue that this scheme is statistical zero-knowledge, but for this we require an additional property of the encoding scheme $\mathcal{E}$:

- Re-randomizable: Given pk and any $e_1 = E(a)$, one can efficiently output a statistically uniform encoding e_2 from the set of encodings of a.

The prover re-randomizes the terms of its proof, so that, information-theoretically, the terms of its proof reveal nothing more than what they encode. Notice that this re-randomizability property holds trivially when the encoding is deterministic, as when we use the suggested encoding over a bilinear group.

Theorem 6. *The ZK SNARK construction above is statistically ZK when instantiated with our canonical QSP.*

Proof. We claim three things. First, for fixed crs_f and statement $u \in \{0,1\}^{n'}$, once the elements V_{mid} and W that are encoded in the proof are fixed, they determine all of the other elements H, V'_{mid}, W', H', and Y that are encoded in the proof, via the verification constraints $V \cdot W^\dagger - H \cdot t(s) = 0$, $V'_{mid} - \alpha V_{mid} = 0$, $W' - \alpha W = 0$, $H' - \alpha H = 0$, and $\gamma Y - (\beta_v \gamma) V_{mid} - (\beta_w \gamma) W = 0$, where $V = v_0(s) + V_{in} + V_{mid}$, $V_{in} = v_{in}(s)$, $v_{in}(x) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(x)$ and $W^\dagger = w_0(s) + W$. Second, in

the ZK construction, the elements V_{mid}, W that are encoded in the proof are statistically uniform. Third, there is a simulator (S_1, S_2) such that S_1 outputs a simulated CRS crs_f and a trapdoor τ, S_2 takes as input crs_f, a statement u and τ and outputs a simulated proof π – in particular, without knowing any witness w for u, and with encodings of appropriately uniform V_{mid} and W (and what is encoded in the remaining terms is dictated by the verification constraints). The theorem follows from these claims, since the simulator can use re-randomization to ensure that its actual encodings (not just what is encoded) is appropriately uniform.

Regarding the first claim, fixing V_{mid} and W clearly fixes V'_{mid}, W', and Y. It also fixes $V = v_0(s) + V_{in} + V_{mid}$ and W, since the coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are determined by u in a strong QSP. Consequently, it fixes $H = V \cdot W/t(s)$ and also H'.

Regarding the second claim, $t(s)$ is in F^* with overwhelming probability. Since the final step of generating V_{mid} and W involves adding $\delta_{v_{mid}} t(s)$ and $\delta_w t(s)$, respectively, for uniform values of $\delta_{v_{mid}}$ and δ_w, the values of V_{mid} and W are statistically close to uniform.

Regarding the third claim, S_1 generates a regular crs_f and sets the trapdoor τ to be s, α, β_v, β_w, γ. Given the trapdoor τ, S_2 picks random $v(x)$, $w^\dagger(x)$ such that $t(x)$ divides $v(x)w^\dagger(x)$, sets $h(x)$ be the quotient polynomial, and sets $v_{mid}(x) = v(x) - v_0(x) - v_{in}(x)$ and $w(x) = w^\dagger(x) - w_0(x)$. Since S_2 knows these polynomials and s, α, β_v, β_w, γ, it can compute encodings of $V_{mid} = v_{mid}(s)$ and $W = w(s)$, as well as the other elements that need to be encoded in the proof. Moreover, as required, the values $V_{mid} = v_{mid}(s)$ and $W = w(s)$ are statistically uniform. ■

Notice that our SNARKs can be re-randomized by anyone, not just the prover. Re-randomizable and malleable NIZKs have found many applications. See [16] and references therein for applications of malleable proof systems.

As also observed by Groth [32] and Lipmaa [38], our result immediately yields a sub-linear size 2-move publicly verifiable witness-indistinguishable argument where the verifier's first message can be reused many times, a so-called Zap [21], as follows: The verifier generates a common reference string, which the prover checks for correctness. Now the prover can make arbitrarily many Zaps using our succinct SNARK with the verifier initial message as CRS.

4.5 Optimizations

4.5.1 Reducing the Verifier's Work *without* PCPs

We have described how to construct a SNARK for the relation $f(u, w) = 1$, where the statement u is required by the verifier, and the remaining portion of the input w is a witness that the prover may set freely. In one extreme case, the verifier may fix all of f's input. For example, in the verifiable computation setting, the verifier may want to outsource to a worker the computation of a predicate $f(u)$ that determines whether the verifier's input u has some property. In this case, to verify, the verifier must first compute on its own an encoding $E(v_{in}(s))$ of the value $V_{in} = v_{in}(s) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(s)$. The verifier's computation here may be quite expensive, depending on how large $|u|$ (which is proportional to $|\mathcal{I}_{in}|$) is. In the other extreme case, the verifier might not make any constraints on the prover's input – i.e., u may be zero bits. For example, the verifier can set up a CRS for a function f that determines whether or not w is a proof of the Riemann Hypothesis. In this case, the prover's SNARK can be verified in constant time, since the most expensive component of verification has gone away; there is no value V_{in} whose encoding needs to be computed.

Suppose that the verifier wants to specify a long statement u, but wants to minimize its verification work. What can we do, besides use PCPs?

Here is a simple approach that may not reduce the verifier's work asymptotically, but which should significantly reduce the verifier's work in practice: use a collision-resistant hash function H. More specifically, rather than using a strong QSP for the original function $f(u, w)$, we use a strong QSP for a related function $g(u', w')$, where w' is parsed as (u, w), where $g(u', w')$ outputs 1 iff $u' = H(u)$ and $f(u, w) = 1$. In the function g, only the (short) hash output u' is fixed by the verifier, but this hash output implicitly fixes (via collision resistance) the statement u that the verifier wants to fix. In effect, u' becomes the new statement, and u becomes part of the witness. To verify the SNARK, the verifier computes $u' = H(u)$, and then uses u' to verify the SNARK for g in the usual way, but now $|\mathcal{I}_{in}| = O(|u'|)$ is potentially much smaller than before. Consequently, the second component of this computation, verifying the SNARK for g, is not very expensive, since u' is so short. In particular, the verifier's computation is now constant, independent of $|u|$, aside from the computation of $u' = H(u)$!

While this may not alter the verifier's computation asymptotically, in practice this will reduce the verifier's computation dramatically. In practice, computing a hash function is very fast, even when the data to be hashed is huge.

On the other hand, this approach increases the prover computation (as the additional computation of $H(u)$ needs to be verified). Accordingly, to find the right tradeoff between prover and verifier computation, one may consider hashing a portion of the input u.

We note that this trick is similar in spirit to a trick by Applebaum et al. [2]. In the context of verifiable computation of a function $f(x)$, they observe that the worker's output can be more succinct – in particular, the size of the worker's output can be made independent of the output length of f – if the VC scheme is run on the function $\mathsf{MAC}_k(f(x))$ rather than $f(x)$. Their trick also makes the client's verification less expensive – in particular, it makes the client's verification work independent of the output length of f, except that the verifier needs to compute a MAC on the output. However, in their setting, the client still needs to perform expensive cryptographic operations (such as FHE encryption) to prepare the *input* to the function. In our setting, our hash trick has an even more dramatic impact on verification efficiency. The verifier's work is completely independent of the input/output length, except for the application of an ordinary (*not "extractable"*) collision-resistant hash function (an *inexpensive* cryptographic primitive).

4.5.2 Reducing the Verifier's Work *with* PCPs

As explained above, the verifier's computation is constant, aside from the computation of an encoding $E(v_{in}(s))$ of $v_{in}(s) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(s)$, which corresponds to the statement u. Why not let the *prover* compute this encoding $E(v_{in}(s))$, and then use PCPs to enable the verifier to quickly check that $E(v_{in}(s))$ was computed correctly?

This "limited" use of PCPs seems more practical than using PCPs to verifier the prover's *entire* computation. Here, we are only using PCPs to verify the computation of $E(v_{in}(s))$, and the complexity of this computation is only proportional to the size $|u|$ of the statement, not to the size of the witness or the entire circuit needed to compute f.

4.5.3 Reducing the Verifier's *Preprocessing*: Combining Universal Circuits with the Hash Trick or PCPs

In our SNARK, someone needs to generate a CRS. Although this pre-processing step is only one-time, it takes computation linear in the circuit size. If the verifier is weak computationally, too weak to compute the function by itself even one time, the verifier may not be able to set up the CRS itself. Fortunately, in our NIZK, crs is a global parameter that does not necessarily need to be generated by the weak verifier individually. On the other hand, crs_f is dependent on a function f that potentially only the verifier is interested in outsourcing. How can a weak verifier outsource f without expensive setup?

One solution is to use a universal circuit. The global setup procedure generates not just crs, but also $\{\mathsf{crs}_{U_k}\}$ for universal circuits $\{U_k\}$ where U_k is able to handle functions whose circuits have between 2^{k-1} and 2^k gates. U_k takes input of the form (f, u, w), where f is a function, where (f, u) acts as a "statement", w is the witness, and it outputs 1 iff $f(u, w) = 1$.

The drawback is that the new statement (f, u) is longer than the old statement u, and the verifier's work (during the actual verification procedure) increases accordingly. But often $|f| + |u|$ will be much shorter than the circuit size, and therefore still a significant improvement. A weak verifier may be able to bear the additional computational complexity.

However, if even $O(|f| + |u|)$ cryptographic group operations is too much for the verifier, we can further reduce the verifier computation by using the hash trick or PCPs, as described in Sections 4.5.1 and 4.5.2. For example, using the hash trick, verification time is actually constant, independent of $|f|+|u|$, except that the verifier needs to hash (f, u) with an ordinary cryptographic hash function, such as SHA-256.

4.5.4 Even Shorter Proofs in the Designated-Verifier Setting

A SNARK proof contains the terms (or similar terms in the ZK setting):

$$E(v_{mid}(s)), E(w(s)), E(h(s)), \quad E(\alpha v_{mid}(s)), E(\alpha w(s)), E(\alpha h(s)), \quad E(\beta_v v_{mid}(s) + \beta_w w(s)).$$

Suppose that, in the designated-verifier setting, the verifier knows α, β_v, β_w. Then, given the proof terms $E(v_{mid}(s)), E(w(s)), E(h(s))$, the verifier can generate the remaining terms of the proof itself. This raises the question: does the prover actually need to include these terms in its proof at all? Is there a more succinct way that the prover can convince the verifier that it knows these terms without actually sending them?

In fact, we can use an extractable collision-resistant hash (ECRH) function here. The prover applies the hash function H to the terms $E(\alpha v_{mid}(s))$, $E(\alpha w(s))$, $E(\alpha h(s))$, $E(\beta_v v_{mid}(s)+\beta_w w(s))$ to produce a short hash digest $\mathcal{D}$, and sends $\pi = (E(v_{mid}(s)), E(w(s)), E(h(s)), \mathcal{D})$. To verify π, the verifier generates the terms $E(\alpha v_{mid}(s))$, $E(\alpha w(s))$, $E(\alpha h(s))$, $E(\beta_v v_{mid}(s) + \beta_w w(s))$ itself, confirms that $\mathcal{D}$ is the correct hash output, and then verifies the de-compressed SNARK in the usual way. In [8, 31], the ECRHs are designated verifier, which is compatible with our designated verifier setting here. In the security proof, the extractability property of the ECRH ensures that the simulator can unpack the full proof from the truncated proof and hash digest, and then proceed as it would in the simulation of the ordinary SNARK scheme.

5 Security of Our Public-Verifier SNARK and NIZK

Here we prove the security of our NIZK construction, which implicitly also proves the security of our non-ZK public-verifier SNARK. We address the security of our designated-verifier construction in Section 6.

We chose to handle the public-verifier case separately in this section, because (1) the assumptions and proof are simpler, and (2) we want to make it clear that our public-verifier construction using bilinear maps is based on the same assumptions that Groth [32] uses for his NIZK construction. Actually, our assumptions are slightly weaker than Groth's.

5.1 Assumptions

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 [32]. (Our q-PDH assumption is actually weaker than his q-CPDH assumption, and our q-PKE assumption is identical to Groth's and Lipmaa's [38], except that we extend the assumption to handle auxiliary inputs.) 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 [10, 11].

The q-PDH assumption is a "conventional" falsifiable assumption, though still somewhat unconventional 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 $q = \text{poly}(\kappa)$. 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*

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

More concretely, suppose $(p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa)$ outputs a description of a cyclic bilinear group of order p, a κ-bit prime, where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is the usual pairing (bilinear map) function. The q-PDH over a bilinear group assumption becomes:

Assumption 2 (q-PDH over a bilinear group)**.** *Let κ be a security parameter, and $q = \text{poly}(\kappa)$. The q-power Diffie-Hellman (q-PDH) assumption holds for $\mathcal{G}$ if for all non-uniform probabilistic polynomial time adversaries $\mathcal{A}$ we have*

$$Pr\left[\begin{array}{c} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa)\ ;\ g \leftarrow \mathbb{G}\backslash\{1\}\ ;\ s \leftarrow \mathbb{Z}_p^*\ ; \\ \sigma \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, \ldots, g^{s^q}, g^{s^{q+2}}, \ldots, g^{s^{2q}})\ ; \\ y \leftarrow \mathcal{A}(\sigma)\ :\ y = g^{s^{q+1}} \end{array}\right] = \mathtt{negl}(\kappa).$$

For the same q, our q-PDH over a bilinear group assumption is almost exactly the same as, but weaker than, Groth's q-CPDH assumption [32]. It is also quite similar to, but weaker than,

Boneh at al.'s [10] bilinear Diffie-Hellman Exponent (BDHE) assumption, which was invoked to prove security of a hierarchical identity-based encryption scheme [10] and a broadcast encryption scheme [11] with constant-sized ciphertexts. We require smaller q than Groth, only linear in the circuit size (versus quadratic), which also makes our assumption weaker.

The q-PKE assumption is a non-falsifiable "knowledge" assumption, similar in spirit to (but of course more complicated than) early knowledge-of-exponent assumptions (KEAs) [6, 19].

Assumption 3 (q-PKE)**.** *Let κ be a security parameter, and $q = \text{poly}(\kappa)$. 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*

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

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

More concretely, over a bilinear group, the q-PKE assumption becomes:

Assumption 4 (q-PKE over a bilinear group)**.** *Let κ be a security parameter, and $q = \text{poly}(\kappa)$. The q-power knowledge of exponent assumption holds for $\mathcal{G}$ 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*

$$Pr[\quad (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa)\ ;\ g \leftarrow \mathbb{G}\backslash\{1\}\ ;\ \alpha, s \leftarrow \mathbb{Z}_p^*\ ;$$
$$\sigma \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, \ldots, g^{s^q}, g^\alpha, g^{\alpha s}, \ldots, g^{\alpha s^q})\ ;$$
$$(c, \hat{c}\ ;\ a_0, \ldots, a_q) \leftarrow (\mathcal{A}\ \|\ \chi_{\mathcal{A}})(\sigma, z)\ :\ \hat{c} = c^\alpha \wedge c \neq \textstyle\prod_{i=0}^{n} g^{a_i s^i}\quad] = \mathtt{negl}(\kappa)$$

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

The q-PKE assumption, and knowledge extraction assumptions in general, while plausible, raise a number of delicate issues. Yao et al. [49] observed, with respect to previous protocols that used knowledge of exponent assumptions (KEAs), that an adversary can get auxiliary inputs from other instances of the protocol, instances run by different parties. In particular, from these other transcripts, the adversary may collect encodings of pairs (r, r^α) for which it does not know a representation of r in terms of the elements given in the KEA problem. So, it is not reasonable to assume that the adversary *knows* such a representation, or that the extractor can *extract* such a representation. Stinson and Wu [48] suggested a way to handle this problem: simply give the extractor access to the internal state of the other parties that are running the protocol as well, or (alternatively) just view the original adversary and these other parties as being part of a single "aggregate" adversary. Although no single party "knows" a representation of r, all of the parties together – the aggregate adversary – "know" the representation of r. Of course, knowledge assumptions start to look rather ridiculous when one imagines such an all-powerful extractor that omnisciently peers into the internal state of all machines (and brains) that are running the protocol in question. Instead, one should view the assumption as saying something like: even with all of the computational power in the world, it is infeasible to generate a pair (r, r^α) except in the obvious way, via a representation over the elements given in the KEA problem.

The q-PKE assumption seems plausible even if the extractor is prohibited from viewing some computations, as long as those computations are not dependent on α. In our statement of the assumption, we capture this notion by providing $\mathcal{A}$ and $\chi_{\mathcal{A}}$ with auxiliary information $z \in \{0,1\}^{\text{poly}(k)}$. $\mathcal{A}$ and $\chi_{\mathcal{A}}$ do not know how z was generated, but they are given the assurance that is not dependent on α. This assurance seems necessary: e.g., if z included an encoded pair (r, r^α) whose representation with respect to $\{s^i\}$ is unknown, then we could not hope for an efficient extractor $\chi_{\mathcal{A}}$, assuming problems such as discrete-log that underlie the encoding scheme are hard. Note that we do not require that z be independent of s. Even if z includes the actual value s, q-PKE still seems plausible.

5.2 Lemmas

To simplify the proof of our main security theorem, we extract out a couple of technical lemmas.

Recall that our SNARK π for statement u and witness w has the form:

$$E(v_{mid}(s)), E(w(s)), E(h(s)), \quad E(\alpha v_{mid}(s)), E(\alpha w(s)), E(\alpha h(s)), \quad E(\beta_v v_{mid}(s) + \beta_w w(s)).$$

Our first lemma basically says that we would be completely convinced by the prover's proof if he gave us the actual *polynomials* $v_{mid}(x)$, $w(x)$, $h(x)$.

Lemma 9. *Let $Q_f = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n],j\in\{0,1\}}\mathcal{I}_{ij})$ be a QSP of size m for function $f : \{0,1\}^n \to \{0,1\}$. For $n' \leq n$, let $u \in \{0,1\}^{n'}$ be a statement, $\mathcal{I}_{in} = \cup_{i\in[n'],j\in[0,1]}\mathcal{I}_{ij}$, and $\mathcal{I}_{mid} = \{1,\ldots,m\} \setminus \mathcal{I}_{in}$. Let $\pi = (v_{mid}(x), w(x), h(x))$. Suppose the following properties hold:*

1. *$(v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) = h(x) \cdot t(x)$ for $v_{in}(x) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(x)$ (where coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are deterministically derived from u in the usual way),*
2. *$v_{mid}(x)$ is in the linear span of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$,*
3. *$w(x)$ is in the linear span of $\{w_k(x) : k \in [m]\}$.*

Then, π is an actual proof *that u is a true statement. That is, π implies unconditionally that there exists witness w with $f(u,w) = 1$, and such w can be extracted from π.*

Proof. Let $\{a_k\}$, $\{b_k\}$ be linear combinations such that $v_{in}(x) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(x)$, $v_{mid}(x) = \sum_{k\in\mathcal{I}_{mid}} a_k v_k(x)$, and $w(x) = \sum_{k\in[m]} b_k w_k(x)$. (This is well-defined, since $\mathcal{I}_{in}$ and $\mathcal{I}_{mid}$ are disjoint.) Let $v(x) = v_{in}(x) + v_{mid}(x)$. Since $v(x)$ and $w(x)$ are in the spans of $\{v_k(x) : k \in [m]\}$ and $\{w_k(x) : k \in [m]\}$, respectively, and since $(v_0(x)+v(x))\cdot(w_0(x)+w(x)) = h(x)\cdot t(x)$, the soundness of the strong QSP implies that the linear combinations represent an unequivocal assignment to the n input bits of f – more formally, there is an assignment $B \in \{0,1\}^n$ such that, for all $i \in [n]$, there exist $k_1, k_2 \in \mathcal{I}_{iB_i}$ with a_{k_1}, b_{k_2} both nonzero, but $a_{k_1} = b_{k_2} = 0$ for all $k_1, k_2 \in \mathcal{I}_{i\bar{B}_i}$. This B can be extracted efficiently from the linear combinations $\{a_k\}$, $\{b_k\}$. By the soundness of the QSP, we have $f(B) = 1$. Parse B as (u', w') with $u' \in \{0,1\}^{n'}$. We have $u = u'$, since $\{a_k\}$ restricted to the indices in $\mathcal{I}_{in}$ corresponds to u. We also have that w' is a witness for u. ■

The second lemma says that if you give me a set of polynomials $\{v_k(x)\}$ of degree d, such as the polynomials in a QSP, I can pick a polynomial $a(x)$ of degree $d+1$ such that $a(x) \cdot v_k(x)$ has a zero coefficient for x^{d+1} for all k. Moreover, if I keep $a(x)$ secret from you, but assure you that $a(x)$ has the aforementioned property and also give you the value $a(s)$ at known s, then, unconditionally with overwhelming probability, you will not be able to output a degree-d polynomial $u(x)$ that is not in the span of the $v_k(x)$'s such that $a(x) \cdot u(x)$ has a zero coefficient for x^{d+1}. One may view

the polynomial $a(x)$ as a "guard" that only lets through polynomials that are in the span of the $v_k(x)$'s. The simulator in the security proof will use this guard to ensure that, if the adversary outputs a false proof that passes verification that implicitly uses some $v_{mid}(x)$ that is not in the span of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$, then the simulator will be able to use that false proof to solve q-PDH.

Lemma 10. *Let $F[x]^{(k)}$ denote polynomials over $F[x]$ of degree at most k. Let $F[x]^{(\neg k)}$ denote polynomials over $F[x]$ that have a zero coefficient for x^k. For some d, let $\mathcal{U} = \{u_k(x)\} \subset F[x]^d$, and let $\mathrm{span}(\mathcal{U})$ denote the set of polynomials that can be generated as F-linear combinations of the polynomials in $\mathcal{U}$. Let $a(x) \in F[x]^{(d+1)}$ be generated uniformly at random subject to the constraint that $\{a(x) \cdot u_k(x) : u_k(x) \in \mathcal{U}\} \subset F[x]^{(\neg(d+1))}$. Let $s \in F^*$. Then, for all algorithms $\mathcal{A}$,*

$$\Pr[u(x) \leftarrow \mathcal{A}(\mathcal{U}, s, a(s)) : u(x) \in F[x]^d \ \wedge \ u(x) \notin \mathrm{span}(\mathcal{U}) \ \wedge \ a(x) \cdot u(x) \in F[x]^{(\neg(d+1))}] \leq 1/|F|.$$

Proof. Assume that $u(x) \in F[x]^d$ and $u(x) \notin \mathrm{span}(\mathcal{U})$. Then, the coefficient vector of $u(x)$ – namely, $(u_0, u_1, \ldots, u_d, 0)$ (padded with a zero to have $d+2$ coefficients) – is not in the span of the coefficient vectors of $\{u_k(x)\}$ and the vector $(s^{d+1}, \ldots, 1)$. (By assumption, $u(x) \notin \mathrm{span}(\mathcal{U})$, and the additional vector $(s^{d+1}, \ldots, 1)$ does not help since it is the only vector with a nonzero coefficient in the rightmost position.)

On the other hand, the only information that $\mathcal{A}$ has about $a(x)$ is the dot product of the *reversal* $(a_{d+1}, \ldots, a_0)$ of $a(x)$'s coefficient vector with the vectors in $\mathcal{U}$ and $(s^{d+1}, \ldots, 1)$. (Observe that the condition that $a(x) \cdot u(x) \in F[x]^{(\neg(d+1))}$ is equivalent to $\sum_{i+j=d+1} a_i \cdot u_j = 0$ – i.e., that the dot product of $u(x)$'s coefficient vector and the reversal of $a(x)$'s coefficient vector is 0.)

Since $a(x)$ appears uniformly random to $\mathcal{A}$ subject to dot product constraints wrt the coefficient vectors of $\{u_k(x)\}$ and the vector $(s^{d+1}, \ldots, 1)$, and since $u(x)$'s coefficient vector is not in the span these vectors, the dot product of the reversal of $a(x)$'s coefficient vector with $u(x)$'s coefficient vector – i.e., the coefficient of x^{d+1} in $a(x) \cdot u(x)$ – appears uniformly random to $\mathcal{A}$. ■

5.3 Security Theorem

Here we prove the main security theorem.

Theorem 7. *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 4, instantiated with a QSP of degree d, is secure under Definition 7, with soundness error $1/|F|$.*

Before proceeding with the formal proof, we provide an informal sketch of the main ideas. The CRS for the scheme contains encodings of $\{v_k(s)\}_{k \in \mathcal{I}_{mid}}$, $\{w_k(s)\}_{k \in [m]}$, and $\{s^i\}_{i \in [0,d]}$, as well as these terms multiplied by α, and the scheme requires the prover to present encodings of V_{mid}, W, and H, and these terms multiplied by α. The reason that we require the prover to duplicate its effort wrt α is so that the simulator in the security proof can extract representations of V_{mid}, W, and H as degree-d polynomials $v_{mid}(x)$, $w(x)$, and $h(x)$ such that $v_{mid} = v_{mid}(s)$, $w = w(s)$ and $h = h(s)$. The d-PKE assumption implies that this extraction is efficient.

Suppose an adversary manages to forge a SNARK of a false statement that nonetheless passes the verification test. Then, by Lemma 9, the soundness of our strong QSP implies that, for the extracted polynomials, one of the following must be true:

1. $(v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) \neq h(x) \cdot t(x)$, for $v_{in}(x) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(x)$ (where coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are deterministically derived from u in the usual way),
2. $v_{mid}(x)$ is not in the linear span of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$,
3. $w(x)$ is not in the linear span of $\{w_k(x) : k \in [m]\}$.

If the first case, then $p(x) = (v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) - h(x) \cdot t(x) = \sum_{i=0}^{k} p_i x^i$ is a nonzero polynomial of degree some $k \leq 2d$ that has s as a root (since the verification test implies $(v_0(s) + v_{in}(s) + v_{mid}(s)) \cdot (w_0(s) + w(s)) = h(s) \cdot t(s)$). If $q + 1 \geq 2d$, the simulator can use $p(x)$ to solve q-PDH by using the fact that $E(0) = E(s^{q+1-k} p(s))$ and subtracting off encodings of lower powers of s to get $E(p_k s^{q+1})$ and then $E(s^{q+1})$. To handle the other cases – i.e., to ensure that $v_{mid}(x)$ is in the linear span of the $v_k(x)$'s with $k \in \mathcal{I}_{mid}$ (and similarly for $w(x)$) – we use two more scalars β_v, β_w, supplement the CRS with more terms $\{\beta_v v_k(s)\}$ and $\{\beta_w w_k(s)\}$, and require the prover to present (encoded) $\beta_v v_{mid}(s) + \beta_w w(s)$ in its proof. In particular, to (implicitly) set β_v, the simulator chooses a polynomial $a^{(v)}(x)$ as in Lemma 10 wrt the set of polynomials $\{v_k(x) : k \in \mathcal{I}_{mid}\}$ and implicitly sets $\beta_v = s^{q-d} a^{(v)}(s)$. We invoke Lemma 10 to argue that if $v_{mid}(x)$ is not in its proper span, then the simulator can, with probability $1 - 1/|F|$, use the encoding of $\beta_v v_{mid}(s) + \beta_w w(s) = s^{q-d} a^{(v)}(s) v_{mid}(s) + s^{q-d} a^{(w)}(s) w(s)$ to obtain an encoding of s^{q+1} and thus solve q-PDH.

Proof. (Theorem 7) Regarding completeness, the properties of the QSP guarantee that the prover can always provide a proof for a satisfying input.

To prove soundness, we assume that there exists an adversary $\mathcal{A}_{snark}$ that returns a cheating proof for the SNARK construction, and we show how we construct an adversary $\mathcal{B}_{pdh}$, which interacts with $\mathcal{A}_{snark}$ and breaks the q-PDH assumption.

Suppose that $\mathcal{B}_{pdh}$ receives the q-PDH challenge $\sigma = (pk, E(1), E(s), \ldots, E(s^q), E(s^{q+2}), \ldots, E(s^{2q}))$. The adversary $\mathcal{A}_{snark}$ generates a function $f : \{0,1\}^n \to \{0,1\}$ that has a QSP of size m and degree d with $q \geq \max\{2d - 1, d + 2\}$, defined as follows:

$$Q_f = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i \in [n], j \in \{0,1\}} \mathcal{I}_{ij}).$$

(It will be convenient notationally, recalling that we are proving the security of the ZK construction, to view $\mathcal{V}$ and $\mathcal{W}$ as also including $t(x)$: $t(x) = v_m(x) = w_m(x)$.) $\mathcal{A}_{snark}$ also selects a number $n' \in [n]$, where the first n' input bits are associated to the statement, and the remaining $n - n'$ input bits are associated to the witness. Let $\mathcal{I}_{in} = \cup_{i \in [n'], j \in [0,1]} \mathcal{I}_{ij}$ and let $\mathcal{I}_{mid}$ be the rest of the indices – i.e., $\{1, \ldots, m\} \setminus \mathcal{I}_{in}$.

Next, $\mathcal{B}_{pdh}$ provides a CRS of the correct form to $\mathcal{A}_{snark}$. The CRS is:

$$\mathsf{crs} = (pk, \{E(s^i)\}_{i \in [0,d]}, \{E(\alpha s^i)\}_{i \in [0,d]})$$

$$\begin{aligned}\mathsf{crs}_f = (\mathsf{crs}, Q_f, n', &\{E(v_k(s))\}_{k \in \mathcal{I}_{mid}}, \{E(w_k(s))\}_{k \in [m]}, \{E(s^i)\}_{i \in [0,d]}, \\ &\{E(\alpha v_k(s))\}_{k \in \mathcal{I}_{mid}}, \{E(\alpha w_k(s))\}_{k \in [m]}, \{E(\alpha s^i)\}_{i \in [0,d]}, \\ &\{E(\beta_v v_k(s))\}_{k \in \mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k \in [m]})\end{aligned}$$

$$\mathsf{shortcrs}_f = (E(1), E(\alpha), E(\gamma), E(\beta_v \gamma), E(\beta_w \gamma), \{E(v_k(s))\}_{k \in \{0\} \cup \mathcal{I}_{in}}, E(w_0(s)), E(t(s))).$$

However, there are some subtleties in how $\mathcal{B}_{pdh}$ generates the values β_v, β_w, and γ. (α, on the other hand, $\mathcal{B}_{pdh}$ just generates uniformly at random from F.) In particular, $\mathcal{B}_{pdh}$ does not know these values explicitly, but rather knows these values in terms of their representations over the power basis $\{s^i\}$. We now describe explicitly how $\mathcal{B}_{pdh}$ generates these values, and observe how $\mathcal{B}_{pdh}$ can generate the CRS despite only knowing these values implicitly.

$\mathcal{B}_{pdh}$ generates β_v as follows. $\mathcal{B}_{pdh}$ generates a uniformly random polynomial $a^{(v)}(x)$ of degree $d+1$ subject to the constraint that all of the polynomials $\{x^{q-d}a^{(v)}(x) \cdot v_k(x) : k \in \mathcal{I}_{mid}\}$ have a zero coefficient for x^{q+1}. $\mathcal{B}_{pdh}$ implicitly sets $\beta_v = s^{q-d}a^{(v)}(s)$. The generation of β_w is analogous: $\beta_w = s^{q-d}a^{(w)}(s)$ for $a^{(w)}(x)$ where all polynomials $\{x^{q-d}a^{(w)}(x) \cdot w_k(x) : k \in [m]\}$ have a zero coefficient for x^{q+1}. Notice that $\mathcal{B}_{pdh}$ can generate encodings of $\{\beta_v v_k(s)\}$ and $\{\beta_w w_k(s)\}$ for the CRS. This is because the polynomial $\beta_v v_k(s) = s^{q-d}a^{(v)}(s)v_k(s)$ has degree at most $q-d+d+1+d = q+1+d \leq 2q$ over s, and because this polynomial has a zero coefficient for the s^{q+1} term; similarly for β_w. Observe also that β_v and β_w generated in this way have appropriately uniform distribution.

To verify a proof, it would suffice for $\mathsf{shortcrs}_f$ to contain only $E(\beta_v)$ and $E(\beta_w)$ (and other terms), rather than (the more complicated) $E(\gamma)$, $E(\beta_v\gamma)$, and $E(\beta_w\gamma)$, but unfortunately $\beta_v = s^{q-d}a^{(v)}(s)$ likely has a nonzero coefficient for s^{q+1}, making it impossible for $\mathcal{B}_{pdh}$ to generate an encoding of it. However, suppose $\mathcal{B}_{pdh}$ generates γ' uniformly from F and implicitly sets $\gamma = \gamma' s^{d+2}$. Then, γ has an appropriately uniform distribution, and $\mathcal{B}_{pdh}$ can generate an encoding of γ from the q-PDH instance since $q \geq d+2$. Moreover $\mathcal{B}_{pdh}$ can generate encodings of $\beta_v\gamma = \gamma' s^{q+2}a^{(v)}(s)$ and $\beta_v\gamma = \gamma' s^{q+2}a^{(w)}(s)$, since these two terms have a zero coefficient for s^{q+1} and are of degree at most $q+2+d+1 = q+d+2 \leq 2q$.

Once $\mathcal{A}_{snark}$ obtains the CRS, it can generate proofs on its own. Since the scheme is public-verifier, we do not need to worry about simulating a verification oracle for $\mathcal{A}_{snark}$'s proofs; such an oracle exists automatically.

Now, suppose that $\mathcal{A}_{snark}$ sends a claimed proof π of a false statement u that passes the verification test. From the verification equations and the fact that the image of the encoding is verifiable, the proof must have the form:

$$E(V_{mid}), E(W), E(H), \quad E(\alpha V_{mid}), E(\alpha W), E(\alpha H), \quad E(\beta_v V_{mid} + \beta_w W) \ .$$

$\mathcal{B}_{pdh}$ applies an extractor to π. Though it may not be immediately recognizable as such, the CRS received by $\mathcal{A}_{snark}$ is in fact a valid input (σ, z) in the d-PKE assumption: it consists of

$$\begin{aligned} \sigma &= (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]}) \\ z &= (Q_f, n', \{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in[m]}, E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma)) \ , \end{aligned}$$

where the auxiliary information z is independent of α, and other terms in the CRS can be generated efficiently from σ, z.

Via extraction through the d-PKE assumption, $\mathcal{B}_{pdh}$ obtains a degree-d polynomial $v_{mid}(x)$ such that $V_{mid} = v_{mid}(s)$. Similarly, it obtains degree-d polynomials $w(x)$ and $h(x)$. Since π verifies, we have that

- $(v_0(s) + v_{in}(s) + v_{mid}(s)) \cdot (w_0(s) + w(s)) = h(s) \cdot t(s)$ for $v_{in}(x) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(x)$ (where coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are deterministically derived from u in the usual way),
- The last term of the proof properly encodes $\beta_v v_{mid}(s) + \beta_w w(s)$.

But since π is a proof for a false statement, Lemma 9 implies that one of the following two cases must hold:

- Case 1: $(v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) \neq h(x) \cdot t(x)$.
- Case 2: Either $v_{mid}(x)$ or $w(x)$ is not in its proper span.

In either case, $\mathcal{B}_{pdh}$ can solve the q-PDH problem.

Suppose Case 1 holds. Then $p(x) = (v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) - h(x) \cdot t(x)$ is a nonzero polynomial of degree $2d$ having s as a root. Suppose $p(x)$'s highest nonzero coefficient is p_k

for some $k \leq 2d$. Write $p^\dagger(x) = x^k - p_k^{-1} \cdot p(x) = \sum_{i=0}^{k-1} p_i^\dagger x^i$. Since s is a root of $x^k - p^\dagger(x)$, it is a root of $x^{q+1} - x^{q+1-k}p^\dagger(x)$, which is a polynomial with all nonnegative powers of x since $q \geq 2d-1$. Thus, we have $s^{q+1} = s^{q+1-k}p^\dagger(s)$, and $\mathcal{B}_{pdh}$ solves q-PDH by computing $E(s^{q+1}) = E(s^{q+1-k}p^\dagger(s))$, where the latter is a known linear combination of encodings $E(1), \ldots, E(s^q)$, which are available from the q-PDH instance.

Suppose Case 2 holds. Wlog, suppose first that $v_{mid}(x)$ cannot be expressed as a linear combination of $\{v_k : k \in \mathcal{I}_{mid}\}$. Recall that $\mathcal{B}_{pdh}$ chose a polynomial of $a^{(v)}(x)$ of degree $d+1$ subject to the constraint that all of the polynomials $\{x^{q-d}a^{(v)}(x) \cdot v_k(x) : k \in \mathcal{I}_{mid}\}$ have a zero coefficient for x^{q+1}. However, information-theoretically, the only information that the extractor χ (which output $v(x)$) has about $a^{(v)}(x)$ is the above constraint, and the value $a^{(v)}(s)$, which it gets via $\beta_v = s^{q-d}a^{(v)}(s)$. Thus, by Lemma 10, since $v_{mid}(x)$ is not in its proper span, the coefficient for x^{q+1} in the polynomial $x^{q-d}a^{(v)}(x)v_{mid}(x)$, and hence the coefficient of s^{q+1} in $\beta_v v_{mid}(s)$, appears unpredictably uniform to χ. Thus, the coefficient of s^{q+1} in $\beta_v v_{mid}(s) + \beta_w w(s)$ also appears unpredictably uniform to χ, regardless of what $w(x)$ is. With probability $1 - 1/|F|$, this coefficient is nonzero. If it is nonzero, the final term of the proof π encodes an element $\beta_v v_{mid}(s) + \beta_w w(s) = s^{q-d}a^{(v)}(s)v_{mid}(s) + s^{q-d}a^{(w)}(s)w(s)$ that is a polynomial of degree $\leq 2q$ over s with a nonzero coefficient for s^{q+1}, and we can subtract off encodings of multiples of the other powers of s (given in the q-PDH instance) to obtain an encoding of a nonzero multiple of s^{q+1}, from which $\mathcal{B}_{pdh}$ can obtain an encoding of s^{q+1}, solving the q-PDH problem.

Extraction: Recall that Definition 7 requires extraction – i.e., for any statement u, there is a polynomial-time extractor $\mathcal{E}_u$ such that, for any $\pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)$, $w \leftarrow \mathcal{E}_u(\mathsf{priv}, \pi)$. Such extraction follows from the extraction of the polynomials $v_{mid}(x)$, $w(x)$ and $h(x)$, as described above, and Lemma 9. ■

6 Security: the Designated-Verifier Case

The security proof for our SNARK construction becomes more complicated in the designated-verifier setting, where sk may be nontrivial and the encoding may even be a semantically secure encryption scheme such as Paillier. In particular, unless the encoding scheme is deterministic (i.e., there is exactly one encoding for each element), it becomes more difficult for the simulator in the security proof to simulate the verification oracle – to provide the adversary with an accurate bit indicating whether a SNARK provided by the adversary verifies successfully. Extraction is a very powerful tool in a security proof, but still we need to slightly strengthen our extraction assumption to enable the simulator to handle semantically secure encoding.

This section is somewhat dependent on Section 5, which discusses the security of the public-verifier SNARK and NIZK. We build on the discussion of assumptions in Section 5.1, and use the lemmas from Section 5.2. Otherwise, the security proof given in Section 6.2 is self-contained.

6.1 Assumptions

We base security on three assumptions: the same q-PDH assumption described in Section 5.1 except that $\mathcal{E}.\mathsf{Setup}(1^k)$ also outputs sk, a version of the q-PKE assumption from Section 5.1 that is slightly extended to address the designated-verifier setting, and one additional knowledge extraction assumption that we call q-PKEQ.

Here is the version of q-PKE, extended to the designated-verifier setting.

Assumption 5 (Augmented q-PKE)**.** *Let κ be a security parameter, and $q = \text{poly}(\kappa)$. The augmented q-power knowledge of exponent (Augmented 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*

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

for any auxiliary information $z \in \{0,1\}^{\text{poly}(\kappa)}$ that is generated independently of α, and depends on sk only to the extent that it can be efficiently generated from $(pk, E(1), E(s), \ldots, E(s^q))$.

As discussed in Section 5.1, we require that z be generated independently of α, since if z includes a pair $(E(r), E(\alpha r))$ "from the outside" (not included in or generated from σ), then $\mathcal{A}$ can easily defeat the extractor by using it to construct $(E(c), E(\hat{c}))$.

The rationale behind the second requirement in Augmented q-PKE, that z is not generated via some super-polynomial computation on $(pk, E(1), E(s), \ldots, E(s^q))$, is that it is unclear whether the assumption would remain true if z were some nontrivial function of sk. If z actually included sk, the assumption is still true in the sense that the extractor $\chi_{\mathcal{A}}$ can "decrypt" $E(c)$ to recover c, decrypt $E(s)$ to recover s, and then easily output some representation of c as a polynomial evaluated at s. But it remains conceivable that z could encode some clever function of sk that enables $\mathcal{A}$ to generate suitable $(E(c), E(\hat{c}))$ without enabling $\chi_{\mathcal{A}}$ to extract a representation of c in terms of the powers $\{s^i\}$. In the terminology of [8], we therefore require the auxiliary information z to have a "benign distribution" with respect to sk. We make this "benign distribution" more concrete by specifying that z depend on sk only to the extent that it can be efficiently generated from $(pk, E(1), E(s), \ldots, E(s^q))$.

Finally, in the designated verifier setting, we use the following additional extraction assumption.

Assumption 6 (q-PKEQ)**.** *Let κ be a security parameter, and $q = \text{poly}(\kappa)$. The q-power knowledge of equality (q-PKEQ) 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*

$$\Pr[\quad (pk, sk) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\kappa) \; ; \; s \leftarrow F^* \; ; \\ \sigma \leftarrow (pk, E(1), E(s), \ldots, E(s^q), E(s^{q+2}), \ldots, E(s^{2q})) \; ; \\ (E(c), e \; ; \; b) \leftarrow (\mathcal{A} \; || \; \chi_{\mathcal{A}})(\sigma) \; : (e \in \{E(c)\} \wedge b = 0) \vee (e \notin \{E(c)\} \wedge b = 1) \quad] = \mathtt{negl}(\kappa).$$

The q-PKEQ assumption basically says that if $\mathcal{A}$ outputs a valid encoding $E(c)$ of some c, and another term e, $\chi_{\mathcal{A}}$ can distinguish whether e also encodes c. This assumption is designed to make up for a shortcoming of augmented q-PKE – namely, it says nothing about what $\chi_{\mathcal{A}}$ does if $\mathcal{A}$ outputs something other than well-formed encodings of some r and $\alpha \cdot r$. For example, even if the encodings are not well-formed, the extractor could output polynomials $v_{mid}(x)$, $w(x)$, $h(x)$ that (as far as the simulator knows) plausibly correspond to the terms in $\mathcal{A}$'s SNARK, leading the simulator to accept a SNARK that a legitimate verifier (with sk) would reject. The q-PKEQ extractor, when combined with other techniques, enables the simulator to avoid this pitfall.

As a reality check, let us consider the plausibility of q-PKEQ when the encoding scheme is Paillier encryption. Paillier encryption [42] is a permutation over $\mathbb{Z}_{N^2}^*$. In particular, image verification in Paillier is easy: every element in $\mathbb{Z}_{N^2}^*$ legitimately encodes some value. Therefore, q-PKEQ boils

down to the question of whether $\mathcal{A}$ can output $(E(c_1), E(c_2))$ where $\chi_{\mathcal{A}}$ cannot distinguish whether $c_1 = c_2$. It is difficult to see how $\mathcal{A}$ itself could know the plaintexts without $\chi_{\mathcal{A}}$ also knowing them. Certainly, $\mathcal{A}$ can generate values $e \in \mathbb{Z}_{N^2}^*$ for which it does not know the corresponding plaintext. It can even generate a second encoding $e' \leftarrow e^{kN+1}$ (for any integer k) that encodes the same unknown value as e. However, despite not know what is encoded, $\mathcal{A}$ (and $\chi_{\mathcal{A}}$) know that these terms encode the same value. $\mathcal{A}$ can generate encodings (e_1, e_2) such that, from the perspective of $\mathcal{A}$ and $\chi_{\mathcal{A}}$, they definitely encode the same value, or have a negligible chance of encoding the same value, but it seems difficult to generate encodings where the probability of equality is anything in between.

Before ending our discussion of the new assumptions for the designated-verifier setting, we remind the reader that in Section 5 we proved the security of a public-verifier SNARK under more natural assumptions, assumptions for a bilinear group that are actually slightly weaker than Groth's [32]. So, a reader who finds the assumptions in this section difficult to accept can ignore this section, and use our PV SNARK in the DV setting. The main advantage of our DV construction is that it permits encodings that do not employ bilinear groups.

6.2 Security Theorem

Here we prove the main security theorem.

Theorem 8. *If the q-PDH and d-PKE assumptions hold for some* $q \geq \max\{2d-1, d+2\}$ *and the encoding scheme is deterministic, or if the q-PDH, d-PKE and q-PKEQ assumptions hold for some* $q \geq \max\{2d-1, d+2\}$*, then the designated-verifier SNARK scheme defined in Section 4, instantiated with a QSP of degree d, is secure under Definition 7, with soundness error* $1/|F|$.

Before proceeding with the formal proof, we refer to the beginning of Section 5 for an informal sketch of the main ideas of the proof. Here, before the proof, we informally highlight the main difference in the designated-verifier setting – namely, the simulation of the verification oracle.

As before, the simulator uses the (augmented) d-PKE extractor χ_{PKE} to extract representations of the proof terms $\pi_{v_{mid}}$, π_w, and π_h as degree-d polynomials $v_{mid}(x)$, $w(x)$, and $h(x)$ such that (allegedly) the proof terms encode $v_{mid} = v_{mid}(s)$, $w = w(s)$ and $h = h(s)$, respectively. At this point, the simulator does not know whether the proof terms actually encode these values – or encode anything at all – since the d-PKE assumption does not say anything about what χ_{PKE} does when the adversary outputs garbage. So, if $v_{mid}(x)$, $w(x)$, and $h(x)$ could correspond to a legitimate SNARK in the sense of satisfying the QSP, the simulator generates encodings of $v_{mid}(s)$, $w(s)$, $h(s)$ and $\beta_v v_{mid}(s) + \beta_w w(s)$ on its own. Then, it basically asks the q-PKEQ χ_{PKEQ} to confirm that the proof terms validly encode the same things. (The simulator does not need q-PKEQ's help if the encoding scheme is deterministic.)

If χ_{PKEQ} says yes, then of course a normal verifier would deem the SNARK to be valid (with overwhelming probability, depending on the accuracy of χ_{PKEQ}). If χ_{PKEQ} says no, then it must be the case (with overwhelming probability) that a normal verifier would reject the proof. To see that this is the case, notice that, certainly, via image verification, the normal verifier would reject the proof if one of the proof terms was not a proper encoding of anything. So suppose that the proof terms $\pi_{v_{mid}}, \pi_w, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \pi_y$ all encode something, but something different than the encodings that the simulator computed. If the pairs $(\pi_{v_{mid}}, \pi_{v'_{mid}})$, $(\pi_w, \pi_{w'})$, and $(\pi_h, \pi_{h'})$ all have the correct relationship with respect to α, then the augmented q-PKE assumption guarantees that χ_{PKE} outputs correct representations of what is encoded by $\pi_{v_{mid}}, \pi_w, \pi_h$. So, given that the proof terms encode something different than the simulator's encodings, it must be the case that

the above pairs do not have the correct relationship with respect to α, or that the π_y term encodes something other than $\beta_v v_{mid}(s) + \beta_w w(s)$. A normal verifier would detect either of these situations. Therefore, the simulator can follow χ_{PKEQ}'s advice and simulate the verification oracle correctly with overwhelming probability.

Proof. (Theorem 8) Regarding completeness, the properties of the QSP guarantee that the prover can always provide a proof for a satisfying input.

To prove soundness, we assume that there exists an adversary $\mathcal{A}_{snark}$ that returns a cheating proof for the SNARK construction, and we show how we construct an adversary $\mathcal{B}_{pdh}$, which interacts with $\mathcal{A}_{snark}$ and breaks the q-PDH assumption.

Suppose that $\mathcal{B}_{pdh}$ receives the q-PDH challenge $\sigma = (pk, E(1), E(s), \ldots, E(s^q), E(s^{q+2}), \ldots, E(s^{2q}))$. The adversary $\mathcal{A}_{snark}$ generates a function $f : \{0,1\}^n \to \{0,1\}$ that has a QSP of size m and degree d with $q \geq \max\{2d-1, d+2\}$, defined as follows:

$$Q_f = (\mathcal{V}, \mathcal{W}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n], j\in\{0,1\}} \mathcal{I}_{ij}).$$

(It will be convenient notationally, recalling that we are proving the security of the ZK construction, to view $\mathcal{V}$ and $\mathcal{W}$ as also including $t(x)$: $t(x) = v_m(x) = w_m(x)$.) $\mathcal{A}_{snark}$ also selects a number $n' \in [n]$, where the first n' input bits are associated to the statement, and the remaining $n - n'$ input bits are associated to the witness.

Next, $\mathcal{B}_{pdh}$ provides a CRS of the correct form to $\mathcal{A}_{snark}$. The CRS is:

$$\mathsf{crs} = (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]})$$

$$\begin{aligned}\mathsf{crs}_f = (\mathsf{crs}, Q_f, n', &\{E(v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(w_k(s))\}_{k\in[m]}, \{E(s^i)\}_{i\in[0,d]},\\ &\{E(\alpha v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\alpha w_k(s))\}_{k\in[m]}, \{E(\alpha s^i)\}_{i\in[0,d]},\\ &\{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in[m]})\end{aligned}$$

$$\mathsf{shortcrs}_f = (\mathsf{priv}, E(1), E(\alpha), E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma), \{E(v_k(s))\}_{k\in\{0\}\cup\mathcal{I}_{in}}, E(w_0(s)), E(t(s))).$$

However, there are some subtleties in how $\mathcal{B}_{pdh}$ generates the values β_v, β_w, and γ. (α, on the other hand, $\mathcal{B}_{pdh}$ just generates uniformly at random from F.) In particular, $\mathcal{B}_{pdh}$ does not know these values explicitly, but rather knows these values in terms of their representations over the power basis $\{s^i\}$. We now describe explicitly how $\mathcal{B}_{pdh}$ generates these values, and observe how $\mathcal{B}_{pdh}$ can generate the CRS despite only knowing these values implicitly.

$\mathcal{B}_{pdh}$ generates β_v as follows. $\mathcal{B}_{pdh}$ generates a uniformly random polynomial $a^{(v)}(x)$ of degree $d+1$ subject to the constraint that all of the polynomials $\{x^{q-d}a^{(v)}(x) \cdot v_k(x) : k \in \mathcal{I}_{mid}\}$ have a zero coefficient for x^{q+1}. (Such a polynomial exists by Lemma 10.) $\mathcal{B}_{pdh}$ implicitly sets $\beta_v = s^{q-d}a^{(v)}(s)$. The generation of β_w is analogous: $\beta_w = s^{q-d}a^{(w)}(s)$ for $a^{(w)}(x)$ where all polynomials $\{x^{q-d}a^{(w)}(x) \cdot w_k(x) : k \in [m]\}$ have a zero coefficient for x^{q+1}. Notice that $\mathcal{B}_{pdh}$ can generate encodings of $\{\beta_v v_k(s)\}$ and $\{\beta_w w_k(s)\}$ for the CRS. This is because the polynomial $\beta_v v_k(s) = s^{q-d}a^{(v)}(s)v_k(s)$ has degree at most $q - d + d + 1 + d = q + 1 + d \leq 2q$ over s, and because this polynomial has a zero coefficient for the s^{q+1} term; similarly for β_w. Observe also that β_v and β_w generated in this way have appropriately uniform distribution.

To verify a proof, it would suffice for $\mathsf{shortcrs}_f$ to contain only $E(\beta_v)$ and $E(\beta_w)$ (and other terms), rather than (the more complicated) $E(\gamma)$, $E(\beta_v\gamma)$, and $E(\beta_w\gamma)$, but unfortunately $\beta_v =$

$s^{q-d}a^{(v)}(s)$ likely has a nonzero coefficient for s^{q+1}, making it impossible for $\mathcal{B}_{pdh}$ to generate an encoding of it. However, suppose $\mathcal{B}_{pdh}$ generates γ' uniformly from F and implicitly sets $\gamma = \gamma' s^{d+2}$. Then, γ has an appropriately uniform distribution, and $\mathcal{B}_{pdh}$ can generate an encoding of γ from the q-PDH instance since $q \geq d+2$. Moreover $\mathcal{B}_{pdh}$ can generate encodings of $\beta_v\gamma = \gamma' s^{q+2}a^{(v)}(s)$ and $\beta_v\gamma = \gamma' s^{q+2}a^{(w)}(s)$, since these two terms have a zero coefficient for s^{q+1} and are of degree at most $q+2+d+1 = q+d+2 \leq 2q$.

Once $\mathcal{A}_{snark}$ obtains the CRS, it can generate proofs on its own. In the designated-verifier setting, we give $\mathcal{A}_{snark}$ access to an oracle that indicates whether its proof verified. (In the public-verifier setting, no such oracle is necessary.) The oracle works by extraction: $\mathcal{B}_{pdh}$ recovers the linear combinations used by the prover, and thereby also extracts a witness for the truth of the statement being proved. (Recall that witness extraction is required by Definition 7 of SNARKs.) We now explain how this extraction works.

Now, when $\mathcal{A}_{snark}$ sends a claimed proof π, it should have the form:

$$E(V_{mid}), E(W), E(H), \quad E(\alpha V_{mid}), E(\alpha W), E(\alpha H), \quad E(\beta_v V_{mid} + \beta_w W) \ ,$$

and it should satisfy various verification equations. But let us not make any presuppositions about the form of $\mathcal{A}_{snark}$'s proof; it sends:

$$\pi_{v_{mid}}, \pi_w, \pi_h, \quad \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \quad \pi_y \ .$$

There may be trivial ways by which $\mathcal{B}_{pdh}$ can recognize π immediately as an invalid proof – e.g., if image verification is a public algorithm and one of the proof terms is not a valid encoding of any element. If so, $\mathcal{B}_{pdh}$ rejects π.

Otherwise, if $(\pi_{v_{mid}}, \pi_{v'_{mid}})$ are of the form $(E(V_{mid}), E(\alpha V_{mid}))$, then the d-PKE assumption implies that there is an efficient extractor χ that outputs a linear combination $a_0, \ldots, a_d$ such that $V_{mid} = \sum a_i \cdot s^i$. Though it may not be immediately recognizable as such, the CRS received by $\mathcal{A}_{snark}$ is in fact a valid input (σ, z) in the d-PKE assumption: it consists of

$$\begin{aligned} \sigma &= (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]}) \\ z &= (Q_f, n', \{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in[m]}, E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma)) \ , \end{aligned}$$

where the auxiliary information z is independent of α, and other terms in the CRS can be generated efficiently from σ, z. Accordingly, $\mathcal{B}_{pdh}$ obtains a d-degree polynomial $v_{mid}(x)$ such that, allegedly, $V_{mid} = v_{mid}(s)$. Similarly, it obtains degree-d polynomials $w(x)$ and $h(x)$. (If it does not obtain such polynomials from the d-PKE extractor, $\mathcal{B}_{pdh}$ rejects π.)

Next, $\mathcal{B}_{pdh}$ checks that $(v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) = h(x) \cdot t(x)$, for $v_{in}(x) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(x)$ (where coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are deterministically derived from u in the usual way). It also checks that $v_{mid}(x)$ is in the span of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$ and that $w(x)$ is in the span of $\{w_k(x) : w \in [m]\}$. If any of these checks fails, $\mathcal{B}_{pdh}$ rejects the proof.

Next, $\mathcal{B}_{pdh}$ generates on its own encodings of $v_{mid}(s)$, $w(s)$, $h(s)$, $\alpha v_{mid}(s)$, $\alpha w(s)$, $\alpha h(s)$, and $\beta_v v_{mid}(s) + \beta_w w(s)$ using the encodings from the q-PDH instance and its knowledge of α. (Although it does not know β_v, β_w explicitly, $\mathcal{B}_{pdh}$ can generate an encoding of $\beta_v v_{mid}(s) + \beta_w w(s)$ as long as $v_{mid}(x)$ and $w(x)$ are in their proper spans, which $\mathcal{B}_{pdh}$ has already checked.) If the encoding scheme is deterministic, $\mathcal{B}_{pdh}$ simply compares its encodings with $\mathcal{A}_{snark}$'s proof terms, accepts if the terms match, and rejects otherwise.

If the encoding scheme is not deterministic, then it invokes the q-PKEQ extractor χ_{PKEQ}. Specifically, for σ from the q-PDH instance, the q-PKEQ assumption implies that if we run $(\mathcal{A}, \chi_{\mathcal{A}})$

on σ, then if $\mathcal{A}$ outputs a valid encoding $E(c)$ and a term d, then $\chi_{\mathcal{A}}$ can distinguish whether d also encodes c. In our setting, $\mathcal{A}$'s computation actually consists of a combination of some of $\mathcal{B}_{pdh}$'s computation and some of $\mathcal{A}_{snark}$'s computation – in particular, it consists of the $\mathcal{B}_{pdh}$'s computation of the CRS from σ, and then $\mathcal{A}_{snark}$'s computation over the CRS. χ_{PKEQ} is run on the various pairs, such as $(E(v_{mid}(s)), \pi_{v_{mid}})$, where the first term was computed by $\mathcal{B}_{pdh}$, and the latter is a term from π. If χ_{PKEQ} outputs '1' on all such pairs (indicating matches), then a normal verifier would accept π, and therefore $\mathcal{B}_{pdh}$ does also. If χ_{PKEQ} outputs '0' on any such pair, then we claim that a normal verifier would reject π, and therefore $\mathcal{B}_{pdh}$ does also. To see that this is the case, notice that, certainly, via image verification, the normal verifier would reject the proof if one of the proof terms was not a proper encoding of anything. So suppose that the proof terms $\pi_{v_{mid}}, \pi_w, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \pi_y$ all encode something, but something different than the encodings that $\mathcal{B}_{pdh}$ computed. If the pairs $(\pi_{v_{mid}}, \pi_{v'_{mid}})$, $(\pi_w, \pi_{w'})$, and $(\pi_h, \pi_{h'})$ all have the correct relationship with respect to α, then the augmented q-PKE assumption guarantees that χ_{PKE} outputs correct representations of what is encoded by $\pi_{v_{mid}}, \pi_w, \pi_h$. So, given that the proof terms encode something different than $\mathcal{B}_{pdh}$'s encodings, it must be the case that the above pairs do not have the correct relationship with respect to α, or that the π_y term encodes something other than $\beta_v v_{mid}(s) + \beta_w w(s)$. A normal verifier would detect either of these situations. Therefore, $\mathcal{B}_{pdh}$ can follow χ_{PKEQ}'s advice and simulate the verification oracle correctly with overwhelming probability.

$\mathcal{B}_{pdh}$ accepts or rejects exactly as a normal verifier would, except that there are two situations in which $\mathcal{B}_{pdh}$ would reject π even though a normal verifier might not:

- Case 1: $(v_0(x)+v_{in}(x)+v_{mid}(x)) \cdot (w_0(x)+w(x)) \neq h(x) \cdot t(x)$, but $(v_0(s)+v_{in}(s)+v_{mid}(s)) \cdot (w_0(s)+w(s)) = h(s) \cdot t(s)$.
- Case 2: Either $v_{mid}(x)$ or $w(x)$ is not in its proper span, but nonetheless the last term of the proof properly encodes $\beta_v v_{mid}(s) + \beta_w w(s)$.

These deviations from the normal verifier are fine, since, in either case, $\mathcal{B}_{pdh}$ can solve the q-PDH problem.

Suppose Case 1 holds. Then $p(x) = (v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) - h(x) \cdot t(x)$ is a nonzero polynomial of degree $2d$ having s as a root. Suppose $p(x)$'s highest nonzero coefficient is p_k for some $k \leq 2d$. Write $p^\dagger(x) = x^k - p_k^{-1} \cdot p(x) = \sum_{i=0}^{k-1} p_i^\dagger x^i$. Since s is a root of $x^k - p^\dagger(x)$, it is a root of $x^{q+1} - x^{q+1-k} p^\dagger(x)$, which is a polynomial with all nonnegative powers of x since $q \geq 2d-1$. Thus, we have $s^{q+1} = s^{q+1-k} p^\dagger(s)$, and $\mathcal{B}_{pdh}$ solves q-PDH by computing $E(s^{q+1}) = E(s^{q+1-k} p^\dagger(s))$, where the latter is a known linear combination of encodings $E(1), \ldots, E(s^q)$, which are available from the q-PDH instance.

Suppose Case 2 holds. Wlog, suppose first that $v_{mid}(x)$ cannot be expressed as a linear combination of $\{v_k : k \in \mathcal{I}_{mid}\}$. Recall that $\mathcal{B}_{pdh}$ chose a polynomial of $a^{(v)}(x)$ of degree $d+1$ subject to the constraint that all of the polynomials $\{x^{q-d} a^{(v)}(x) \cdot v_k(x) : k \in \mathcal{I}_{mid}\}$ have a zero coefficient for x^{q+1}. However, information-theoretically, the only information that the extractor χ (which output $v(x)$) has about $a^{(v)}(x)$ is the above constraint, and the value $a^{(v)}(s)$, which it gets via $\beta_v = s^{q-d} a^{(v)}(s)$. Thus, by Lemma 10, since $v_{mid}(x)$ is not in its proper span, the coefficient for x^{q+1} in the polynomial $x^{q-d} a^{(v)}(x) v_{mid}(x)$, and hence the coefficient of s^{q+1} in $\beta_v v_{mid}(s)$, appears unpredictably uniform to χ. Thus, the coefficient of s^{q+1} in $\beta_v v_{mid}(s) + \beta_w w(s)$ also appears unpredictably uniform to χ, regardless of what $w(x)$ is. With probability $1 - 1/|F|$, this coefficient is nonzero. If it is nonzero, the final term of the proof π encodes an element $\beta_v v_{mid}(s) + \beta_w w(s) = s^{q-d} a^{(v)}(s) v_{mid}(s) + s^{q-d} a^{(w)}(s) w(s)$ that is a polynomial of degree $\leq 2q$ over

s with a nonzero coefficient for s^{q+1}, and we can subtract off encodings of multiples of the other powers of s (given in the q-PDH instance) to obtain an encoding of a nonzero multiple of s^{q+1}, from which $\mathcal{B}_{pdh}$ can obtain an encoding of s^{q+1}, solving the q-PDH problem.

Extraction: Recall that Definition 7 requires extraction – i.e., for any statement u, there is a polynomial-time extractor $\mathcal{E}_u$ such that, for any $\pi \leftarrow \mathsf{P}(\mathsf{crs}, u, w)$, $w \leftarrow \mathcal{E}_u(\mathsf{priv}, \pi)$. Such extraction follows from the extraction of the polynomials $v_{mid}(x)$, $w(x)$ and $h(x)$, as described above, and Lemma 9. ■

7 Quadratic Programs for Arithmetic Circuits

Our quadratic span programs (QSPs) consisted of only two sets of polynomials. This fact allowed us to construct protocols – VC, SNARKs, NIZKs – with very succinct, constant size, arguments. If our QSPs had required, say, three sets of polynomials, our QSP-based protocols would have been slightly less efficient – in particular, the arguments still would have been constant size, but for a slightly larger constant.

A disadvantage of QSPs, however, is that they compute boolean circuits. In many cases, it is more natural, and more efficient, to compute arithmetic circuits – i.e., circuits composed of additions and multiplications modulo p, where p is an appropriate modulus. For example, one appealing feature of Groth-Sahai proofs is that they can be adapted to prove relationships among group elements, which are most naturally expressed as equations modulo the group order. Proving such relationships via a boolean circuit would be much less efficient.

In this section, we present quadratic arithmetic programs (QAPs), which are analogous to QSPs, but which "naturally" compute arithmetic circuits. QAPs require three sets of polynomials, and therefore they lead to slightly longer, constant sized, arguments. However, for many computations, they may be much more efficient for the prover, since the prover performs "cryptographic" operations – for example, a small constant number of group multiplications – only for each mod-p gate, not for each boolean gate.

7.1 Definitions: QAP and Strong QAP

Definition 11 (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 target polynomial $t(x)$, all from $F[x]$.*

Let f 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

$$t(x) \quad \text{divides} \quad \left(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} a_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} a_k \cdot y_k(x)\right) \ .$$

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

Note that we can assume that all of the $v_k(x)$'s, $w_k(x)$'s, and $y_k(x)$'s have degree at most $\deg(t(x)) - 1$, since they can all be reduced modulo $t(x)$ without affecting the divisibility check (the check whether $t(x)$ divides the expression).

For the protocols, we will also need a slightly stronger definition, of a "strong QAP", which rules out the perverse possibility that different linear combinations are used for the $v_k(x)$'s, $w_k(x)$'s, and $y_k(x)$'s.

Definition 12 (Strong QAP)**.** *A strong QAP is a QAP with one additional property: for any* $(a_1, \ldots, a_m, b_1, \ldots, b_m, c_1, \ldots, c_m) \in F^{3m}$ *that satisfies*

$$t(x) \quad \text{divides} \quad \left(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} b_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x)\right)$$

it must be the case that $(a_1, \ldots, a_m) = (b_1, \ldots, b_m) = (c_1, \ldots, c_m)$.

From QAP to Strong QAP.

Given a QAP for f, it is straightforward to construct a strong QAP for f. Suppose we are given a QAP Q that computes f and consists of polynomial sets $\mathcal{V} = \{v_k(x) : k \in \{0, \ldots, m\}\}$, $\mathcal{W} = \{w_k(x) : k \in \{0, \ldots, m\}\}$, and $\mathcal{Y} = \{y_k(x) : k \in \{0, \ldots, m\}\}$, and a target polynomial $t(x)$ of degree d, all from $F[x]$. From Q, we construct a strong QAP Q' that computes f and consists of polynomial sets $\mathcal{V}' = \{v'_k(x) : k \in \{0, \ldots, m\}\}$, $\mathcal{W}' = \{w'_k(x) : k \in \{0, \ldots, m\}\}$, and $\mathcal{Y}' = \{y'_k(x) : k \in \{0, \ldots, m\}\}$, and a target polynomial $t'(x)$ of degree $d + 2m$, as follows.

Choose $2m$ elements $r_1, \ldots, r_m, s_1, \ldots, s_m \in F^{2m}$ that are not roots of $t(x)$. Set $T(x) = \prod_{i=1}^{m}(x - r_i)(x - s_i)$ and the new target polynomial $t'(x) = t(x) \cdot T(x)$.

For all $k \in \{0, \ldots, m\}$, we set $v'_k(x)$, $w'_k(x)$, $y'_k(x)$ to satisfy:

$$v'_k(x) = v_k(x) \bmod t(x), \quad w'_k(x) = w_k(x) \bmod t(x), \quad y'_k(x) = y_k(x) \bmod t(x).$$

For all $k \in \{0, \ldots, m\}$, we set $v'_k(x)$, $w'_k(x)$, $y'_k(x)$ to evaluate to 0 at all of the r_i's and s_i's, subject to the following exceptions:

- For all $i \in [m]$, $v'_0(s_i) = 1$ and $w'_0(r_i) = 1$.
- For all $k \in [m]$, $v'_k(r_k) = 1$, $w'_k(s_k) = 1$, and $y'_k(r_k) = y'_k(s_k) = 1$.

By the Chinese Remainder theorem, we can set the polynomials to meet these conditions. Clearly, $t'(x)$ has degree $d + 2m$, and the rest of the polynomials can be made to have degree at most $d + 2m - 1$ (as they can be reduced modulo $t'(x)$). Theorem 9 says that Q' is a strong QAP that computes f.

Theorem 9. *Suppose that there is a QAP of size* m *and degree* d *that computes* f*. Then, there is a strong QAP of size* m *and degree* $d + 2m$ *that computes* f*.*

Proof. Let Q and Q' be defined as above. Since Q is a QAP, and since all of the polynomials in Q' are congruent to their counterparts in Q modulo $t(x)$, the following statement (*) 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

$$t(x) \quad \text{divides} \quad \left(v'_0(x) + \sum_{k=1}^{m} a_k \cdot v'_k(x)\right) \cdot \left(w'_0(x) + \sum_{k=1}^{m} a_k \cdot w'_k(x)\right) - \left(y'_0(x) + \sum_{k=1}^{m} a_k \cdot y'_k(x)\right) .$$

To finish the proof, it suffices to prove the following statement (**):

$$T(x) \quad \text{divides} \quad \left(v'_0(x) + \sum_{k=1}^{m} a_k \cdot v'_k(x)\right) \cdot \left(w'_0(x) + \sum_{k=1}^{m} b_k \cdot w'_k(x)\right) - \left(y'_0(x) + \sum_{k=1}^{m} c_k \cdot y'_k(x)\right)$$
$$\iff \quad (a_1, \ldots, a_m) = (b_1, \ldots, b_m) = (c_1, \ldots, c_m) .$$

The reason it suffices is that, by the $\Leftarrow$ direction and statement (*) above, we conclude that the following statement 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

$$t'(x) \quad \text{divides} \quad \left(v_0'(x) + \sum_{k=1}^{m} a_k \cdot v_k'(x)\right) \cdot \left(w_0'(x) + \sum_{k=1}^{m} a_k \cdot w_k'(x)\right) - \left(y_0'(x) + \sum_{k=1}^{m} a_k \cdot y_k'(x)\right) .$$

Therefore, Q' is a QAP that computes f. By the $\Rightarrow$ direction, Q' is a strong QAP.

It remains to prove statement (**). For $i \in [m]$, the polynomial $v_0'(x) + \sum_{k=1}^{m} a_k \cdot v_k'(x)$ evaluates to a_i at r_i, and to 1 at s_i. Analogously, $w_0'(x) + \sum_{k=1}^{m} b_k \cdot w_k'(x)$ evaluates to b_i at s_i, and to 1 at r_i. Finally, $y_0'(x) + \sum_{k=1}^{m} c_k \cdot y_k'(x)$ evaluates to c_i at both r_i and s_i. Thus, the expression above is divisible by $T(x)$ – i.e., evaluates to 0 at all r_i and s_i – iff for all $i \in [m]$, we have $a_1 \cdot 1 - c_i = 0$ and $1 \cdot b_i - c_i = 0$, which is equivalent to $(a_1, \ldots, a_m) = (b_1, \ldots, b_m) = (c_1, \ldots, c_m)$. ■

7.2 QAPs for Arithmetic Circuits with One Multiplication Gate

Any arithmetic circuit may be viewed as being composed of addition, multiplication-by-scalar, and multiplication gates. In this subsection, we focus on a special type of function, which we call a multiplication subcircuit. A multiplication subcircuit has a single output wire, which is the output wire of a multiplication gate. Connected to the inputs of the single (only) multiplication gate, the multiplication subcircuit may have an arbitrary number of addition and multiplication-by-scalar gates. That is, if the inputs to the multiplication subcircuit are $(X_1, \ldots, X_t)$, the function computed by the multiplication subcircuit can be expressed as $\rho_1(X_1, \ldots, X_t) \cdot \rho_2(X_1, \ldots, X_t)$, where $\rho_1(X_1, \ldots, X_t)$ and $\rho_2(X_1, \ldots, X_t)$ are linear polynomials.

It should be clear that any arithmetic circuit can be expressed as a "circuit" composed of multiplication subcircuits that are stitched together.[7] In the next subsection, we will explain how to compose QAPs for functions which share common input/output variables. This will allow us to construct QAPs for general arithmetic circuits from our QAPs for multiplication subcircuits.

The main result of this subsection is the following theorem.

Theorem 10. *Let C be an arithmetic multiplication subcircuit with $m-1$ inputs (and 1 output). There is a QAP of size m and degree 1 that computes C.*

Since the QAP has degree 1 – i.e., the target polynomial $t(x)$ is linear, and the rest of the polynomials have degree 0 (they are constants) – the QAP looks a bit bizarre. In particular, most of the polynomials are not really "polynomials", in the usual sense. The degree-1 QAPs will look more natural when we compose them. To compose them, we will use the Chinese Remainder Theorem modulo the various (linear) target polynomials $t_1(x), t_2(x), \ldots$ to get $v_k(x)$'s, $w_k(x)$'s, and $y_k(x)$'s that have higher degree, and thus look more like legitimate polynomials.

Proof. Set $t(x) = x - r$ for some $r \in F$ to be the target polynomial.

Let $\rho_1(X_1, \ldots, X_{m-1}) = c_0 + \sum_{i=1}^{m-1} c_i \cdot X_i$ and $\rho_2(X_1, \ldots, X_{m-1}) = d_0 + \sum_{i=1}^{m-1} d_i \cdot X_i$ be the linear polynomials corresponding to the induced values of the left input wire and right input wire, respectively, of the multiplication gate in C. For $k \in \{0, \ldots, m-1\}$, set $v_k(x) = c_k$, $w_k(x) = d_k$ and $y_k(x) = 0$. Set $v_m(x) = w_m(x) = 0$ and $y_m(x) = 1$.

[7] There is one minor complication: the output gates of the arithmetic circuit need to be multiplication gates. However, the fix is simple: create one more input variable, which is always required to be assigned '1', and create multiplication gates at the top of the circuit, which multiply this new variable with the "old" outputs that are not outputs of multiplication gates.

We claim that this is a QAP for C. Suppose $a_1, \ldots, a_m \in F^m$ is a valid assignment to the input/output of C. Then,

$$\left(v_0(x) + \sum_{k=1}^{m} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} a_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} a_k \cdot y_k(x)\right)$$
$$= \rho_1(a_1, \ldots, a_{m-1}) \cdot \rho_2(a_1, \ldots, a_{m-1}) - a_m = 0$$

is (trivially) divisible by $t(x)$. Conversely, if the polynomial expression above is divisible by $t(x)$, it must equal 0, and $a_1, \ldots, a_m \in F^m$ is a valid assignment to the input/output of C. ■

To distill the strategy for computing multiplication subcircuits in words, the $v_k(x)$ polynomials "handle" the left input to the multiplication gate, and the $w_k(x)$ polynomials handle the right input, and the $y_k(x)$ polynomials (really, only $y_m(x)$) handle the output. When we compose multiplication subcircuits to form general arithmetic circuits, the strategy will be the same – the $v_k(x)$'s will "handle" all of the left inputs to multiplication gates, the $w_k(x)$'s all of the right inputs, and the $y_k(x)$'s all of the outputs – but these values will be handled in parallel via the Chinese Remainder Theorem.

7.3 Composition of QAPs, and QAPs for General Arithmetic Circuits

Here we describe how to compose QAPs for different functions that may share common input/output variables. The composition technique takes two functions f_1 and f_2, where some of f_1's output variables may become some of f_2's input variables, and their respective QAPs, and then composes their QAPs via a natural invocation of the Chinese Remainder Theorem, assuming their target polynomials are relatively prime. If f_1 and f_2 do not overlap at all, the size of the new QAP is just the sum of the sizes of the original QAPs, but if they do overlap, the size may be much smaller. The degree of the new QAP is just the sum of the degrees of the component QAPs. Indeed the target polynomial of the composed QAP is simply the product of the original target polynomials. By composing multiplication subcircuits, we can construct a QAP for any arithmetic circuit.[8]

Composition of QAPs.

The composition works as follows. For $i \in \{1, 2\}$, QAP Q_i consists of polynomial sets $\mathcal{V}^{(i)} = \{v_k^{(i)}(x) : k \in \mathcal{I}_i\}$, $\mathcal{W}^{(i)} = \{w_k^{(i)}(x) : k \in \mathcal{I}_i\}$, $\mathcal{Y}^{(i)} = \{y_k^{(i)}(x) : k \in \mathcal{I}_i\}$, and target polynomial $t^{(i)}(x)$, where each $\mathcal{I}_i$ is a set of indices that includes indices for all of the input/output variables of circuit f_i, and where $\mathcal{I}_1 \cap \mathcal{I}_2$ is the set of ℓ indices of the variables that are outputs of f_1 and inputs to f_2. For indices in $k \in \mathcal{I}_1 \setminus \mathcal{I}_2$, we say $v_k^{(2)}(x) = w_k^{(2)}(x) = y_k^{(2)}(x) = 0$. Similarly, for indices in $k \in \mathcal{I}_2 \setminus \mathcal{I}_1$, we say $v_k^{(1)}(x) = w_k^{(1)}(x) = y_k^{(1)}(x) = 0$.

The composed QAP Q consists of polynomial sets $\mathcal{V} = \{v_k(x) : k \in \mathcal{I}_1 \cup \mathcal{I}_2\}$, $\mathcal{W} = \{w_k(x) : k \in \mathcal{I}_i \cup \mathcal{I}_2\}$, $\mathcal{Y} = \{y_k(x) : k \in \mathcal{I}_i \cup \mathcal{I}_2\}$, and target polynomial $t(x)$, as follows. We set $t(x) = t^{(1)}(x) \cdot t^{(2)}(x)$. For all $k \in \mathcal{I}_1 \cup \mathcal{I}_2$, $i \in \{1, 2\}$, we set $v_k(x) = v_k^{(i)}(x) \bmod t^{(i)}(x)$, $w_k(x) = w_k^{(i)}(x) \bmod t^{(i)}(x)$, and $y_k(x) = y_k^{(i)}(x) \bmod t^{(i)}(x)$. Since $t^{(1)}(x)$ and $t^{(2)}(x)$ are relatively prime, we can find polynomials to satisfy these equations by the Chinese Remainder Theorem.

Theorem 11 says that this composition works as intended.

[8] ...subject to the previous footnote.

Theorem 11. *Let Q_1 and Q_2 be two QAPs whose respective target polynomials are relatively prime and which compute the arithmetic circuits f_1 and f_2, where f_2's input variables may include some (say ℓ) of f_1's output variables (but f_1 and f_2 do not otherwise intersect). Let f be the composed function formed by stitching f_1 and f_2 together at their common variables. There is a QAP Q with* $\text{size}(Q) = \text{size}(Q_1) + \text{size}(Q_2) - \ell$ *and* $\deg(Q) = \deg(Q_1) + \deg(Q_2)$ *that computes f. Q's target polynomial is the product of the target polynomials for Q_1 and Q_2.*

Proof. Define Q_1, Q_2, Q, as above. Clearly, $\text{size}(Q) = |\mathcal{I}_1 \cup \mathcal{I}_2| = \text{size}(Q_1) + \text{size}(Q_2) - \ell$, and $\deg(Q) = \deg(Q_1) + \deg(Q_2)$. It remains to show that Q computes f.

Let $\mathcal{I}_{inout}$, $\mathcal{I}_{1,inout}$, and $\mathcal{I}_{2,inout}$ be the indices of the input/output variables of f, f_1, and f_2, respectively. We have $\mathcal{I}_{inout} \subseteq \mathcal{I}_{1,inout} \cup \mathcal{I}_{2,inout}$. Suppose $\{a_k : k \in \mathcal{I}_{inout}\}$ is a valid assignment of these variables. This assignment must extend to a valid assignment $\{a_k : k \in \mathcal{I}_{1,inout} \cup \mathcal{I}_{2,inout}\}$ of the union of the input/output variables for f_1 and f_2. Since Q_1 is a QAP, there exists $\{b_k : k \in \mathcal{I}_1\}$ that is consistent with the valid assignment to $\mathcal{I}_{1,inout}$ such that

$$t^{(1)}(x) \text{ divides } \left(v_0^{(1)}(x) + \sum_{k \in \mathcal{I}_1} b_k \cdot v_k^{(1)}(x)\right) \cdot \left(w_0^{(1)}(x) + \sum_{k \in \mathcal{I}_1} b_k \cdot w_k^{(1)}(x)\right) - \left(y_0^{(1)}(x) + \sum_{k \in \mathcal{I}_1} b_k \cdot y_k^{(1)}(x)\right) .$$

Similarly, since Q_2 is a QAP, there exists $\{c_k : k \in \mathcal{I}_2\}$ that is consistent with the valid assignment to $\mathcal{I}_{2,inout}$ such that

$$t^{(2)}(x) \text{ divides } \left(v_0^{(2)}(x) + \sum_{k \in \mathcal{I}_2} c_k \cdot v_k^{(2)}(x)\right) \cdot \left(w_0^{(2)}(x) + \sum_{k \in \mathcal{I}_2} c_k \cdot w_k^{(2)}(x)\right) - \left(y_0^{(2)}(x) + \sum_{k \in \mathcal{I}_2} c_k \cdot y_k^{(2)}(x)\right) .$$

These linear combinations $\{b_k : k \in \mathcal{I}_1\}$ and $\{c_k : k \in \mathcal{I}_2\}$ must be consistent on the indices in the intersection $\mathcal{I}_1 \cap \mathcal{I}_2$, which correspond to output values in f_1 and input values in f_2 which have already been fixed by the valid assignment. Therefore, we can merge the two linear combinations above, setting $a_k = b_k$ for all $k \in \mathcal{I}_1$ and $a_k = c_k$ for all $k \in \mathcal{I}_2$. Since $v_k(x) = v_k^{(1)}(x) \bmod t^{(1)}(x)$, $w_k(x) = w_k^{(1)}(x) \bmod t^{(1)}(x)$, and $y_k(x) = y_k^{(1)}(x) \bmod t^{(1)}(x)$ for all k, and since $v_k(x) = w_k(x) = y_k(x) = 0 \bmod t^{(1)}(x)$ for $k \in \mathcal{I}_2 \setminus \mathcal{I}_1$, we obtain

$$t^{(1)}(x) \text{ divides } \left(v_0(x) + \sum_{k \in \mathcal{I}_1 \cup \mathcal{I}_2} a_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k \in \mathcal{I}_1 \cup \mathcal{I}_2} a_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k \in \mathcal{I}_1 \cup \mathcal{I}_2} a_k \cdot y_k(x)\right) .$$

Similarly, we obtain that $t^{(2)}(x)$, and hence $t(x)$ divides the expression above.

Conversely, if the divisibility check just above holds, it can be decomposed into the separate divisibility checks wrt the polynomials for the respective QAPs Q_1 and Q_2, which implies (since Q_1 and Q_2 are QAPs) that $\{a_k : k \in \mathcal{I}_1 \cup \mathcal{I}_2\}$ contains valid assignments to the input/output variables of f_1 and f_2, which is therefore a valid assignment to the subset of these variables that are in $\mathcal{I}_{inout}$. Thus, Q is a QAP that computes f. ■

Theorem 12. *Let C be an arithmetic circuit with input from F^n that has s multiplication gates, each with fan-in 2, and whose output gates are all multiplication gates. There is a QAP with size $n + s$ and degree s that computes C.*

Proof. The proof simply combines Theorems 10 and 11 recursively. We use the fact that we can compose s multiplication subcircuits to construct C, including them into the aggregate circuit one at a time, invoking Theorem 11 each time. As long as $s < |F|$, we can ensure that the target polynomials of the QAPs for the multiplication subcircuits are relatively prime, so as to permit the composition. The degree of Q is s. Inductively, we see that the size of C equals the number of input variables for C, plus the number of variables in C that are the output of a multiplication gate; hence $n + s$. ■

As mentioned above, the output wires of an arithmetic circuit are not, in general, outputs exclusively of multiplication gates. Consequently, to apply Theorem 12, we may need to modify our arithmetic circuit slightly. In particular, we create one more input variable, which is always required to be assigned '1', and create multiplication gates at the top of the circuit, which multiply this new variable with the "old" outputs that are not already outputs of multiplication gates.

When we assign input values $a_1, \ldots, a_n$ to C, this induces assignments to all of the wires of C. In our QAP for C, one can verify that it has the following nice property: for $m = n + s$, $(\sum_{k=1}^{m} a_k \cdot v_k(x)) \cdot (\sum_{k=1}^{m} a_k \cdot w_k(x)) - (\sum_{k=1}^{m} a_k \cdot y_k(x))$ is divisible by $t(x)$ precisely when $(a_1, \ldots, a_m)$ is the induced set of values on a "special" subset of wires – namely, the wires corresponding to inputs to C and to output wires of multiplication gates. In other words, there is an immediate and natural mapping between the linear combination $(a_1, \ldots, a_m)$ used in the QAP, and the actual values on the wires of the arithmetic circuit.

7.4 Illustration of a QAP for a Simple Arithmetic Circuit

Constructing the QAP for a circuit by composition may make the construction somewhat confusing. Thus, we provide a redundant informal explanation that does not rely on the composition theorem so heavily, and uses a concrete simple arithmetic circuit as an example.

Consider the arithmetic circuit shown in Figure 1. As shown in the Figure 1, we express each input value to a multiplication gate as a linear function of the values lower in the circuit, where each of these values is either an output of a lower multiplication gate (for which there is a no-multiplication-gate path from this lower multiplication gate to the original multiplication gate), or is an input value to the circuit. (For convenience, we will refer to "special values" or "special wires" as the values or wires that are outputs of multiplication gates, or are input values or wires to the circuit.)

More formally, for each multiplication gate g, we have an equation of the form

$$a_g = \left(\sum_{k \in \mathcal{I}_{g,L}} c_{g,L,k} \cdot a_k \right) \cdot \left(\sum_{k \in \mathcal{I}_{g,R}} c_{g,R,k} \cdot a_k \right) .$$

In the equation, a_g denotes the value of the output of multiplication gate g, and a_k denotes some other "special" value, which itself may be the output of a lower multiplication gate, or an input value. The notation $\mathcal{I}_{g,L}$ denotes the subset of special wires that are *indirect left inputs* to g – i.e., special values that feed into the left wire of g (perhaps indirectly, but without going through another multiplication gate). Finally, $c_{g,L,k}$ is the scalar applied to wire k's value as it feeds into g's left input wire. For example, in Figure 1, we have that $a_5 = (a_1 + 7a_2) \cdot (a_2 - 2a_3)$. Values a_1 and a_2 are indirect left inputs to gate 5, while values a_2 and a_3 are indirect right inputs.

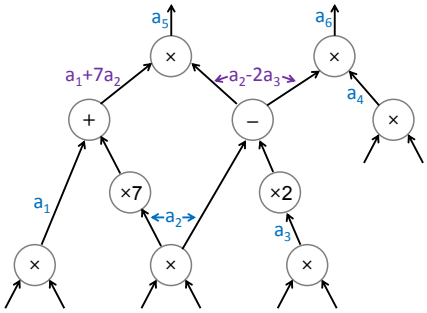

Figure 1: **An arithmetic circuit.** *The value at each output wire of a multiplication gate is expressed in terms of the values of output wires of lower multiplication gates (or of input wires).*

Let $\mathcal{M}$ be the indices of the multiplication gates (viewed as being the same as the indices associated to their output values). For a circuit with s multiplication gates, we would choose distinct $\{r_i \in F : i \in \mathcal{M}\}$, associate each multiplication gate $g \in \mathcal{M}$ to r_g, and set the target polynomial $t(x) = \prod_{g \in \mathcal{M}}(x - r_g)$.

Returning to Figure 1, let us assume for simplicity that wires 1 through 4 are input wires. To generate the target polynomial for out QAP, we pick distinct values $r_5, r_6 \in F$, associated to values 5 and 6, which are the output values of gates 5 and 6. We set the target polynomial $t(x) = (x - r_5) \cdot (x - r_6)$.

In general, we set the $\{v_k(x)\}$, $\{w_k(x)\}$, and $\{y_k(x)\}$ polynomials as follows. For $k \in \mathcal{M}$, we set $y_k(x)$ so that $y_k(r_k) = 1$ and $y_k(r_j) = 0$ for all $j \neq k$. Since the $y_k(x)$'s correspond to "output values", these polynomials are set to 0 for all of the other indices, which correspond to the input values to the circuit (rather than to output values of multiplication gates). We set $v_k(x)$ and $w_k(x)$ to be polynomials such that:

$$\begin{aligned} v_k(r_g) &= c_{g,L,k} \quad \text{for all} \quad k \in \mathcal{I}_{g,L}. \qquad \text{Otherwise} \quad v_k(r_g) = 0. \\ w_k(r_g) &= c_{g,R,k} \quad \text{for all} \quad k \in \mathcal{I}_{g,R}. \qquad \text{Otherwise} \quad w_k(r_g) = 0. \end{aligned}$$

Notice that we have

$$v(r_g) = \sum_{k=1}^{m} a_k \cdot v_k(r_g) = \sum_{k \in \mathcal{I}_{g,L}} a_k \cdot c_{g,L,k} \quad \text{and} \quad w(r_g) = \sum_{k=1}^{m} a_k \cdot w_k(r_g) = \sum_{k \in \mathcal{I}_{g,R}} a_k \cdot c_{g,R,k} \ ,$$

and therefore, using the multiplication equations for the gates, we have

$$\left(\sum_{k=1}^{m} a_k \cdot v_k(r_g)\right) \cdot \left(\sum_{k=1}^{m} a_k \cdot w_k(r_g)\right) - \left(\sum_{k=1}^{m} a_k \cdot y_k(r_g)\right) = 0$$

for all g iff $(a_1, \ldots, a_m)$ is a valid assignment of the special wires, as desired.

Returning again to Figure 1, the polynomials $\{v_k(x)\}$, $\{w_k(x)\}$, and $\{y_k(x)\}$, for $k \in [6]$, will all have degree at most 1. It will be convenient to express these polynomials as ordered pairs $(\cdot, \cdot)$, representing their evaluations at r_5 and r_6. The y polynomials are all zero, except $y_5(x) = (1, 0)$ and $y_6(x) = (0, 1)$. The polynomials $v_5(x), v_6(x), w_5(x), w_6(x)$ are all zero, since the 5-th and 6-th wires are not left or right inputs to any gate. The remaining polynomials are: $v_1(x) = (1, 0)$, $w_1(x) = (0, 0)$, $v_2(x) = (7, 1)$, $w_2(x) = (1, 0)$, $v_3(x) = (0, -2)$, $w_3(x) = (-2, 0)$, $v_4(x) = (0, 0)$, $w_4(x) = (0, 1)$. For example, we have that $v_2(x) = (7, 1)$, since wire 2 is an indirect left input (with magnitude 7) to gate 5 and an indirect left input (with magnitude 1) to gate 6. We have $w_2(x) = (1, 0)$, since wire 2 is an indirect right input to gate 5 (with magnitude 1).

Now, suppose that we have a candidate assignment $(a_1, \ldots, a_6)$ to the special wires of the circuit in Figure 1. We compute the following:

$$\sum_{k=1}^{6} a_k \cdot v_k(r_5) = a_1 + 7a_2, \quad \sum_{k=1}^{6} a_k \cdot w_k(r_5) = a_2 - 2a_3, \quad \sum_{k=1}^{6} a_k \cdot y_k(r_5) = a_5,$$
$$\sum_{k=1}^{6} a_k \cdot v_k(r_6) = a_2 - 2a_3, \quad \sum_{k=1}^{6} a_k \cdot w_k(r_6) = a_4, \quad \sum_{k=1}^{6} a_k \cdot y_k(r_6) = a_6.$$

We have that $t(x)$ divides $\left(\sum_{k=1}^{6} a_k \cdot v_k(x)\right) \cdot \left(\sum_{k=1}^{6} a_k \cdot w_k(x)\right) - \left(\sum_{k=1}^{6} a_k \cdot y_k(x)\right)$ iff $a_5 = (a_1 + 7a_2) \cdot (a_2 - 2a_3)$ and $a_6 = (a_2 - 2a_3) \cdot a_4$ – that is, iff $(a_1, \ldots, a_6)$ is a valid assignment of the special wires.

7.5 QAP Efficiency Considerations

The main efficiency advantage of QAPs over QSPs is that QAPs handle arithmetic circuits "naturally" – that is, without converting them into boolean circuits. Otherwise, the efficiency analysis for QAPs is similar to that for QSPs (see Section 2.4). We briefly summarize some differences.

Like the QSP polynomials, the QAP polynomials are quite structured in that, typically, $v_k(r_i) = 0$ for "most" pairs (i, k), since this value can be nonzero only when indices k and i have a "local" relationship in the circuit. For example, the wire corresponding to k must be an indirect left input to gate i. For $v_k(r_i)$ or $w_k(r_i)$ to be nonzero, k must be an input to i's multiplication subcircuit. We can exploit this structure to compute and write these polynomials in a compressed form, as described in Section 2.4.

One complication is that we have not bounded the number of inputs to each multiplication subcircuit. This means that, in fact, the number of nonzero values $v_k(r_i)$ may be super-linear in the circuit size. Recall that our performance optimizations from Section 2.4 relied heavily on the vectors $\{v_k(r_i) : i \in [d]\}$ being extremely sparse (having only constant support).

For such perverse cases, our solution is to insert dummy multiplication gates into the arithmetic circuit (where a native wire value is multiplied with a dummy wire value that is set to '1') so as to "break up" each overly large multiplication subcircuit into constant size multiplication subcircuits.

In more detail, suppose that the arithmetic circuit has a gate with more than two indirect left inputs. That is, for some gate g, the gate equation is of the form

$$a_g = \left(\sum_{k \in \mathcal{I}_{g,L}} c_{g,L,k} \cdot a_k \right) \cdot \left(\sum_{k \in \mathcal{I}_{g,R}} c_{g,R,k} \cdot a_k \right) \ ,$$

where $|\mathcal{I}_{g,L}| > 2$. Then, we modify the arithmetic circuit by grouping the indices in $\mathcal{I}_{g,L}$ into pairs: for $j \in [\lceil |\mathcal{I}_{g,L}|/2 \rceil]$, $\mathcal{I}_{g,L} = \cup_j \mathcal{I}_{g,L,j}$. We create dummy multiplication gates $\{g_j\}$, with gate equations:

$$a_{g_j} = \left(\sum_{k \in \mathcal{I}_{g,L,j}} c_{g,L,k} \cdot a_k \right) \cdot (1) \ ,$$

whose right input is a dummy wire that is set to '1'. At this point, if we set the new gate equation to be

$$a_g = \left(\sum_j a_{g_j} \right) \cdot \left(\sum_{k \in \mathcal{I}_{g,R}} c_{g,R,k} \cdot a_k \right) \ ,$$

the linear equation for the left side now has half as many variables. We continue breaking up the left side into pairs, in a binary tree fashion, until the left and right inputs for a_g take only two variables.

Alternatively, one may modify the arithmetic circuit in the following way (which is basically equivalent). First, modify the circuit so that addition gates have fan-in 2. Then, on top of each addition gate, place a dummy multiplication gate – that is, a multiplication gate that takes two inputs: the output of the addition gate, and a dummy input that is always set to '1'.

If the original arithmetic circuit already had addition gates with fan-in 2, then the modification at most doubles the number of gates. In any case, it multiplies the number of wires in the circuit by only a constant factor. Of course, the modification may increase the number of multiplication gates, and hence the degree of the QAP, by a super-linear factor.

With this modification, all of the vectors $\{v_k(r_i) : i \in [d]\}$ associated to the modified arithmetic circuit have constant support, and the optimizations from Section 2.4 may be applied.

8 SNARK Construction from QAP

We now define our SNARK system $\Pi = (\mathsf{Gen}, \mathsf{Regen}_f, \mathsf{P}, \mathsf{V})$ for QAPs. Let $R = \{(u, w)\}$ be a set of relations over F^n. For convenience, we will refer to u as the "statement" and w as the "witness". In the description of our SNARK, the input indices $i \in [1, n']$ will correspond to the statement u, and the positions $[n' + 1, n]$ will correspond to the witness w. We assume that R is a relation that can be verified with an arithmetic circuit over F, which we take to mean that there is a efficient arithmetic function $f(u, w) = 1$ iff $(u, w) \in R$.

Let us pause to consider how we can construct such a relation-checker over a large field. Suppose that we are interested in the language L where $(x, w) \in R$ is and only if $f(x) = w$. We modify f to construct an arithmetic circuit ϕ which takes (u, w) as input, outputs '1' if $(u, w) \in R$ and '0' otherwise, and run the SNARK using a QAP for ϕ. ϕ runs f on input u (by incorporating f's

arithmetic circuit) to obtain f's output values $w'_{n'+1}, \ldots, w'_n$. It then computes $(b_{n'+1}, \ldots, b_n) \leftarrow (w'_{n'+1} - w_{n'+1}, \ldots, w'_n - w_n)$, which should be all zeros if ϕ's input is satisfying. While there may be simpler approaches, ϕ may then compute $(b_{n'+1}^{|F|-1}, \ldots, b_n^{|F|-1})$, where $|F|-1$ is the order of the multiplicative subgroup of the field (each exponentiation requires $\log(|F|-1)$ multiplication gates), which is a vector that is still all-zero if ϕ's input is satisfying, but otherwise has one or more '1's in it. Using standard techniques, ϕ may then finish off the computation so that the output is '1' if the input is satisfying, and '0' otherwise. Depending on the application, one may alternatively choose to run a SNARK-like protocol that does not verify a relation *per se*, but instead uses the more natural form of QAPs as in Definition 11 to check that (u, w) is a valid input/output pair.

CRS generation Gen**:** On input the security parameter κ and an upper bound, d, on the degree of the strong QAP for the functions f that will be computed, run $(pk, sk) \leftarrow \mathcal{E}.\mathtt{Setup}$, generate uniformly at random $\alpha, s \leftarrow F^*$ and output $\mathsf{priv} = sk$, $\mathsf{crs} = (pk, \{E(s^i)\}_{i\in[0,d]}, \{E(\alpha s^i)\}_{i\in[0,d]})$.

Function specific CRS generation Regen**:** On input crs, a function f with a strong QAP

$$Q_f = (\mathcal{V}, \mathcal{W}, \mathcal{Y}, t(x), \mathcal{I}_{free}, \mathcal{I}_{labeled} = \cup_{i\in[n], j\in\{0,1\}}\mathcal{I}_{ij}).$$

of size m and degree at most d, and value $n' \in [1, n]$, let $\mathcal{I}_{in} = \cup_{i\in[1,n'], j\in[0,1]}\mathcal{I}_{ij}$, let $\mathcal{I}_{mid}$ be all of the remaining indices $\{1, \ldots, m\} \setminus \mathcal{I}_{in}$, generate uniformly at random $\beta_v, \beta_w, \beta_y, \gamma \leftarrow F^*$, and output:

- Public common reference string for f, n':

$$\begin{aligned}\mathsf{crs}_f = (\mathsf{crs}, Q_f, n', &\{E(v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(w_k(s))\}_{k\in[m]}, \{E(y_k(s))\}_{k\in[m]}, \{E(s^i)\}_{i\in[0,d]},\\ &\{E(\alpha v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\alpha w_k(s))\}_{k\in[m]}, \{E(\alpha y_k(s))\}_{k\in[m]}, \{E(\alpha s^i)\}_{i\in[0,d]},\\ &\{E(\beta_v v_k(s))\}_{k\in\mathcal{I}_{mid}}, \{E(\beta_w w_k(s))\}_{k\in[m]}, \{E(\beta_y y_k(s))\}_{k\in[m]}).\end{aligned}$$

- A short CRS that will be used for verification

$$\mathsf{shortcrs}_f = (\mathsf{priv}, E(1), E(\alpha), E(\gamma), E(\beta_v\gamma), E(\beta_w\gamma), E(\beta_y\gamma), \{E(v_k(s))\}_{k\in\mathcal{I}_{in}}, E(w_0(s)), E(t(s))).$$

Prove P**:** On input crs_f, statement $u \in \{0,1\}^{n'}$, and witness w, P will prove that $(u, w) \in R$ – i.e., that $f(u, w) = 1$. The prover evaluates the span program Q_f to obtain $(a_1, \ldots, a_m)$ such that

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

For $v_{mid}(x) = \sum_{k\in\mathcal{I}_{mid}} a_k \cdot v_k(x)$, $w(x) = \sum_{k\in[m]} a_k \cdot w_k(x)$, and $y(x) = \sum_{k\in[m]} a_k \cdot y_k(x)$, the prover outputs the following proof π:

$$\begin{aligned}\{&E(v_{mid}(s)), E(w(s)), E(y(s)), E(h(s)),\\ &E(\alpha v_{mid}(s)), E(\alpha w(s)), E(\alpha y(s)), E(\alpha h(s)),\\ &E(\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s))\}.\end{aligned}$$

Verify V: On input $\mathsf{shortcrs}_f$, u, and proof $\pi = (\pi_{v_{mid}}, \pi_w, \pi_y, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{y'}, \pi_{h'}, \pi_z)$, V confirms that the terms are in the support of validly encoded elements using image verification. Let V_{mid}, W, Y, H, V'_{mid}, W', Y', H', and Z denote what is encoded in the respective terms. V computes an encoding $E(v_{in}(s))$ of $v_{in}(s) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(s)$. Using the quadratic root detection, V confirms that $(v_0(s) + v_{in}(s) + V_{mid}) \cdot (w_0(s) + W) - (y_0(s) + Y)) - H \cdot t(s) = 0$, $V'_{mid} - \alpha V_{mid} = 0$, $W' - \alpha W = 0$, $Y' - \alpha Y = 0$, $H' - \alpha H = 0$, and $\gamma Z - (\beta_v \gamma) V_{mid} - (\beta_w \gamma) W - (\beta_y \gamma) Y = 0$.

8.1 Zero-Knowledge SNARKs (including NIZKs) from QAP

The idea for randomization of the SNARK construction from QAP is the same as in the case of SNARKs from QSPs. We outline the construction next.

To facilitate the randomization in our ZK construction, we include additional terms $E(t(s))$, $E(\alpha t(s))$, $E(\beta_v t(s))$, $E(\beta_w t(s))$, $E(\beta_y t(s))$, $E(v_0(s))$, $E(\alpha v_0(s))$, $E(w_0(s))$, $E(\alpha w_0(s))$, $E(y_0(s))$ and $E(\alpha y_0(s))$ in crs_f. The prover proceeds as before, except that it additively perturbs $E(v_{mid}(s))$, $E(w(s))$ and $E(y(s))$ by random multiples of $E(t(s))$, and modifies the other values accordingly.

More specifically, the values of $\{a_k\}_{k \in \mathcal{I}_{in}}$, $v_{in}(x)$, $v_{mid}(x)$, $v(x) = v_0(x) + v_{in}(x) + v_{mid}(x)$, $w(x)$, $y(x)$ and $h(x)$ are as before. The verifier picks random $\delta_{v_{mid}}, \delta_w, \delta_y \leftarrow F$. Its proof is:

$$E(v'_{mid}(s)), E(w'(s)), E(y'(s)), E(h'(s)), \quad E(\alpha v'_{mid}(s)), E(\alpha w'(s)), E(\alpha y'(s)), E(\alpha h'(s)),$$
$$E(\beta_v v'_{mid}(s) + \beta_w w'(s) + \beta_y y'(s)),$$

where $v'_{mid}(x) = v_{mid}(x) + \delta_{v_{mid}} t(x)$, $w'(x) = w(x) + \delta_w t(x)$ and $y'(x) = y(x) + \delta_y t(x)$. Regarding the value of $h'(x)$, let $v'(x) = v_0(x) + v_{in}(x) + v'_{mid}(x) = v(x) + \delta_{v_{mid}} t(x) = v^\dagger(x) + \delta_{v_{mid}} t(x)$, where $v^\dagger(x) = v_0(x) + v_{in}(x) + v_{mid}(x)$. Define $w'(x)$, $w^\dagger(x)$ and $y'(x)$, $y^\dagger(x)$ similarly. We have:

$$\begin{aligned} h'(x) &= (v'(x) \cdot w'(x) - y'(x))/t(x) \\ &= ((v^\dagger(x) + \delta_{v_{mid}} t(x)) \cdot (w^\dagger(x) + \delta_w t(x)) - (y^\dagger(x) + \delta_y t(x)))/t(x) \\ &= h(x) + \delta_{v_{mid}} w^\dagger(x) + \delta_w v^\dagger(x) + \delta_{v_{mid}} \delta_w t(x) - \delta_y. \end{aligned}$$

Notice that all of the values in this new proof can be computed efficiently from crs_f. As in the QSP construction we require re-randomizability for the encoding scheme, which is used by the prover to re-randomize the terms in the proof, so that, information-theoretically, they do not reveal more than what they encode.

Theorem 13. *The ZK SNARK construction above is statistically ZK.*

Proof. We claim three things. First, for fixed crs_f and statement $u \in \{0,1\}^{n'}$, once the elements V_{mid}, W, and Y that are encoded in the proof are fixed, they determine all of the other elements H, V'_{mid}, W', Y', H', and Z that are encoded in the proof, via the verification constraints $V \cdot W^\dagger - Y^\dagger - H \cdot t(s) = 0$, $V'_{mid} - \alpha V_{mid} = 0$, $W' - \alpha W = 0$, $Y' - \alpha Y = 0$, $H' - \alpha H = 0$, and $\gamma Z - (\beta_v \gamma) V_{mid} - (\beta_w \gamma) W - (\beta_y \gamma) Y = 0$, where $V = v_0(s) + V_{in} + V_{mid}$, $V_{in} = v_{in}(s)$, $v_{in}(x) = \sum_{k \in \mathcal{I}_{in}} a_k \cdot v_k(x)$, $W^\dagger = w_0(s) + W$ and $Y^\dagger = y_0(s) + Y$. Second, in the ZK construction, the elements V_{mid}, W, Y that are encoded in the proof are statistically uniform. Third, there is a simulator (S_1, S_2) such that S_1 outputs a simulated CRS crs_f and a trapdoor τ, S_2 takes as input crs_f, a statement u and τ and outputs a simulated proof π – in particular, without knowing any witness w for u, and with encodings of appropriately uniform V_{mid}, W and Y (and what is encoded

in the remaining terms is dictated by the verification constraints). The theorem follows from these claims, since the simulator can use re-randomization to ensure that its actual encodings (not just what is encoded) is appropriately uniform.

Regarding the first claim, fixing V_{mid}, W and Y clearly fixes V'_{mid}, W', Y' and Z. It also fixes $V = v_0(s) + V_{in} + V_{mid}$, W and Y, since the coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are determined by u. Consequently, it fixes $H = (V \cdot W^\dagger - Y^\dagger)/t(s)$ and also H'.

Regarding the second claim, $t(s)$ is in F^* with overwhelming probability. Since the final step of generating V_{mid}, W and Y involves adding $\delta_{v_{mid}} t(s)$, $\delta_w t(s)$ and $\delta_y t(s)$, respectively, for uniform values of $\delta_{v_{mid}}$, δ_w and δ_y, the values of V_{mid}, W and Y are statistically close to uniform.

Regarding the third claim, S_1 generates a regular crs_f and sets the trapdoor τ to be s, α, β_v, β_w, β_y, γ. Given the trapdoor τ, S_2 picks random $v(x)$, $w^\dagger(x)$, $y^\dagger(x)$ such that $t(x)$ divides $v(x)w^\dagger(x) - y^\dagger$, sets $h(x)$ be the quotient polynomial, and sets $v_{mid}(x) = v(x) - v_0(x) - v_{in}(x)$, $w(x) = w^\dagger(x) - w_0(x)$ and $y(x) = y^\dagger(x) - y_0(x)$. Since S_2 knows these polynomials and s, α, β_v, β_w, β_y, γ, it can compute encodings of $V_{mid} = v_{mid}(s)$, $W = w(s)$ and $Y = y(s)$, as well as the other elements that need to be encoded in the proof. Moreover, as required, the values $V_{mid} = v_{mid}(s)$, $W = w(s)$ and $Y = y(s)$ are statistically uniform. ■

Like the QSP-based NIZK, the QAP-based NIZK can be re-randomized by anyone, not just the original prover. See [16] and references therein for applications of re-randomizable NIZKs and malleable proof systems.

Theorem 14. *If the q-PDH and d-PKE assumptions hold for some $q \geq \max\{2d-1, d+2\}$, then the NIZK scheme defined above, instantiated with a QAP of degree d, is secure under Definition 7, with soundness error $1/|F|$.*

Proof Idea: Since the proof of the theorem is the same as the proof of ZK SNARK construction from Section 5.3 except for the additional set of polynomials $y_k(x)$, we provide only a sketch for the proof idea. The CRS for the scheme contains encodings of $\{v_k(s)\}_{k\in\mathcal{I}_{mid}}$, $\{w_k(s)\}_{k\in[m]}$, $\{y_k(s)\}_{k\in[m]}$ and $\{s^i\}_{i\in[0,d]}$, as well as these terms multiplied by α, and the scheme requires the prover to present encodings of V_{mid}, W, Y and H, and these terms multiplied by α. This would enable the simulator in the security proof to extract representations of V_{mid}, W, Y and H as degree-d polynomials $v_{mid}(x)$, $w(x)$, $y(x)$ and $h(x)$ such that $v_{mid} = v_{mid}(s)$, $w = w(s)$, $y = y(s)$ and $h = h(s)$. The d-PKE assumption implies that this extraction is efficient.

If an adversary manages to forge a SNARK of a false statement that nonetheless passes the verification test, then, by Lemma 9, the soundness of the strong QAP implies that, for the extracted polynomials, one of the following must be true:

1. $(v_0(x)+v_{in}(x)+v_{mid}(x))\cdot(w_0(x)+w(x))-(y_0(x)+y(x)) \neq h(x)\cdot t(x)$, for $v_{in}(x) = \sum_{k\in\mathcal{I}_{in}} a_k \cdot v_k(x)$, where coefficients $\{a_k : k \in \mathcal{I}_{in}\}$ are deterministically derived from u in the usual way,
2. $v_{mid}(x)$ is not in the linear span of $\{v_k(x) : k \in \mathcal{I}_{mid}\}$,
3. $w(x)$ is not in the linear span of $\{w_k(x) : k \in [m]\}$,
4. $y(x)$ is not in the linear span of $\{y_k(x) : k \in [m]\}$.

If the first case, set $p(x) = (v_0(x) + v_{in}(x) + v_{mid}(x)) \cdot (w_0(x) + w(x)) - (y_0(x) + y(x)) - h(x) \cdot t(x)$. Then, $p(x)$ is a nonzero polynomial of degree some $k \leq 2d$ that has s as a root. If $q + 1 \geq 2d$, the simulator can use $p(x)$ to solve q-PDH by using the fact that $E(0) = E(s^{q+1-k}p(s))$ and subtracting off encodings of lower powers of s to get $E(p_k s^{q+1})$ and then $E(s^{q+1})$ (see the proof of Theorem 7 for details). To handle the other cases – i.e., to ensure that $v_{mid}(x)$ is in the linear span of the $v_k(x)$'s with $k \in \mathcal{I}_{mid}$ (and similarly for $w(x)$ and y(x)) – we use three more

scalars $\beta_v, \beta_w, \beta_y$, and include in the CRS the following terms $\{\beta_v v_k(s)\}$, $\{\beta_w w_k(s)\}$ and $\{\beta_y y_k(s)\}$. Then we require the prover to present (encoded) $\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)$ in its proof. The simulator implicitly sets β_v choosing a polynomial $a^{(v)}(x)$ as in Lemma 10 wrt the set of polynomials $\{v_k(x) : k \in \mathcal{I}_{mid}\}$ and setting $\beta_v = s^{q-d} a^{(v)}(s)$. We invoke Lemma 10 to argue that if $v_{mid}(x)$ is not in its proper span, then the simulator can, with probability $1 - 1/|F|$, use the encoding of $\beta_v v_{mid}(s) + \beta_w w(s) - \beta_y y(s) = s^{q-d} a^{(v)}(s) v_{mid}(s) + s^{q-d} a^{(w)}(s) w(s) - s^{q-d} a^{(y)}(s) y(s)$ to obtain an encoding of s^{q+1} and thus solve q-PDH.

The technical details of the proof follow the proof in Theorem 7 with the difference that the polynomials $y_k(x)$ are handled in the same way as $w_k(x)$.

8.2 Designated-Verifier SNARK from QAP

Just as in the case of the construction of a designated-verifier SNARK from QSP, we would need to resort to the augmented assumptions from Section 6.1 in order to prove the security of the DV SNARK construction from QAP. (We remind the reader that the augmented assumption is necessary only in the case of randomized encodings such a Paillier encryption, and not for the encoding instantiation from bilinear groups.) Since the proof follows closely the proof of Theorem 8, we present only the intuition sketch.

Theorem 15. *If the q-PDH and d-PKE assumptions hold for some* $q \geq \max\{2d-1, d+2\}$ *and the encoding scheme is deterministic, or if the q-PDH, d-PKE and q-PKEQ assumptions hold for some* $q \geq \max\{2d-1, d+2\}$*, then the designated-verifier SNARK scheme defined in Section 4, instantiated with a QAP of degree d, is secure under Definition 7, with soundness error* $1/|F|$.

Proof Idea: The proof for the designated verifier has the same general idea as in the case of the PV SNARK with the following differences. As before, the simulator uses the (augmented) d-PKE extractor χ_{PKE} to extract representations of the proof terms $\pi_{v_{mid}}$, π_w, π_y and π_h as degree-d polynomials $v_{mid}(x)$, $w(x)$, $y(x)$ and $h(x)$ such that (allegedly) the proof terms encode $v_{mid} = v_{mid}(s)$, $w = w(s)$, $y = y(s)$ and $h = h(s)$, respectively. At this point, the simulator does not know whether the proof terms actually encode these values – or encode anything at all – since the d-PKE assumption does not say anything about what χ_{PKE} does when the adversary outputs garbage. So, if $v_{mid}(x)$, $w(x)$, $y(x)$ and $h(x)$ could correspond to a legitimate SNARK in the sense of satisfying the QAP, the simulator generates encodings of $v_{mid}(s)$, $w(s)$, $y(s)$, $h(s)$ and $\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)$ on its own. Then, it basically asks the q-PKEQ χ_{PKEQ} to confirm that the proof terms validly encode the same things. (The simulator does not need q-PKEQ's help if the encoding scheme is deterministic.)

If χ_{PKEQ} says yes, then of course a normal verifier would deem the SNARK to be valid (with overwhelming probability, depending on the accuracy of χ_{PKEQ}). If χ_{PKEQ} says no, then it must be the case (with overwhelming probability) that a normal verifier would reject the proof. To see that this is the case, notice that, certainly, via image verification, the normal verifier would reject the proof if one of the proof terms was not a proper encoding of anything. So suppose that the proof terms $\pi_{v_{mid}}, \pi_w, \pi_y, \pi_h, \pi_{v'_{mid}}, \pi_{w'}, \pi_{h'}, \pi_{y'}, \pi_z$ all encode something, but something different than the encodings that the simulator computed. If the pairs $(\pi_{v_{mid}}, \pi_{v'_{mid}})$, $(\pi_w, \pi_{w'})$, $(\pi_y, \pi_{y'})$, and $(\pi_h, \pi_{h'})$ all have the correct relationship with respect to α, then the augmented q-PKE assumption guarantees that χ_{PKE} outputs correct representations of what is encoded by $\pi_{v_{mid}}, \pi_w, \pi_y, \pi_h$. So, given that the proof terms encode something different than the simulator's encodings, it must be the case that the above pairs do not have the correct relationship with respect to α, or that the π_z term encodes something other than $\beta_v v_{mid}(s) + \beta_w w(s) + \beta_y y(s)$. A normal verifier would detect

either of these situations. Therefore, the simulator can follow χ_{PKEQ}'s advice and simulate the verification oracle correctly with overwhelming probability.

For the technical details of the proof we refer the reader to the proof of Theorem 8. The only difference here will be the terms related to the $y_k(x)$ polynomials, which will be handled in the same way as those related to the $w_k(x)$ polynomials.

9 Conclusions and Open Questions

We introduced Quadratic Span Programs (QSPs) as a new characterization of NP that lends itself to efficient cryptographic applications. Specifically, we use QSPs to construct non-interactive zero-knowledge (NIZK) arguments and hence succinct non-interactive arguments of knowledge (SNARKs) and verifiable computation schemes, all without the need for Probablistically Checkable Proofs (PCPs). The NIZK arguments require only 7 group elements, and the common reference string is linear in the circuit size. The prover performs only a linear number of cryptographic operations, though the prover must also perform a quasi-linear, non-cryptographic step to compute the quotient polynomial for the QSP. Computing it in purely linear time, by exploiting the structure of the QSP's polynomials, remains an interesting open problem. Finally, we introduce Quadratic Arithmetic Programs (QAPs) as a variant of QSPs that more naturally computes arithmetic circuits, and hence may be more efficient for many applications. Although not all of our schemes use pairings (bilinear maps), QSPs appear to be a very useful model for capturing the power of pairings in cryptography: using QSPs, we get much more powerful pairing-based schemes than existed previously, within a conceptually simple framework.

Acknowledgement

The research of the first 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.

References

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

[2] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010.

[3] 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.

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

[5] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 21–31. ACM, 1991.

[6] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2004.

[7] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[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 Goldwasser [29], pages 326–349.

[9] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012. http://eprint.iacr.org/2012/095.

[10] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[11] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In Goldwasser [29], pages 350–366.

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

[14] Ran Canetti and Ronny Ramzi Dakdouk. Towards a theory of extractable functions. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 595–613. Springer, 2009.

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

[16] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. *IACR Cryptology ePrint Archive*, 2012:12, 2012. To appear in Eurocrypt 2012.

[17] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2010.

[18] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 2008.

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

[20] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 54–74. Springer, 2012.

[21] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM J. Comput.*, 36(6):1513–1543, 2007.

[22] 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.

[23] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[24] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[25] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[26] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

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

[28] Kristian Gjøsteen. *Subgroup membership problems and public key cryptosystems*. PhD thesis, Norwegian University of Science and Technology, 2004. `urn.kb.se/resolve?urn=urn:nbn:no:ntnu:diva-128`.

[29] Shafi Goldwasser, editor. *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. ACM, 2012.

[30] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *STOC*, pages 113–122. ACM, 2008.

[31] 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.

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

[33] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, 2006.

[34] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.

[35] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1998.

[36] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, pages 278–291. IEEE Computer Society, 2007.

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

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

[39] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2011.

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

[41] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[42] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[43] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *IACR Theory of Cryptography Conference (TCC)*, 2012.

[44] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[45] Guy N. Rothblum and Salil P. Vadhan. Are PCPs inherent in efficient arguments? *Computational Complexity*, 19(2):265–304, 2010.

[46] Srinath Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the ISOC Symposium on Network and Distributed System Security (NDSS)*, 2012.

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

[48] Jiang Wu and Douglas R. Stinson. An efficient identification protocol and the knowledge-of-exponent assumption. *IACR Cryptology ePrint Archive*, 2007:479, 2007.

[49] Andrew Chi-Chih Yao, Frances F. Yao, Yunlei Zhao, and Bin Zhu. Deniable internet key-exchange. *IACR Cryptology ePrint Archive*, 2007:191, 2007.