Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes

Helger Lipmaa

Institute of Computer Science, University of Tartu, Estonia

Abstract. Gennaro, Gentry, Parno and Raykova proposed an efficient NIZK argument for CIRCUIT-SAT, based on non-standard tools like conscientious and quadratic span programs. We propose a new linear PCP for the CIRCUIT-SAT, based on a combination of *standard* span programs (that verify the correctness of every individual gate) and high-distance linear error-correcting codes (that check the consistency of wire assignments). This allows us to simplify all steps of the argument, which results in significantly improved efficiency. We then construct an NIZK CIRCUIT-SAT argument based on existing techniques.

Keywords: Circuit-SAT, linear error-correcting codes, linear PCP, non-interactive zero knowledge, polynomial algebra, quadratic span program, span program, verifiable computation

1 Introduction

By using non-interactive zero knowledge (NIZK, [3]), the prover can create a proof π, s.t. any verifier can later, given access to a common reference string, the statement, and π, verify the truth of the intended statement without learning any side information. Since a single proof might get transferred and verified many times, one often requires sublinear communication and verifier's computation. (Unless stated explicitly, we measure the communication in group elements, and the computation in group operations.) While succinct NIZK proofs are important in many cryptographic applications, there are only a few different generic methodologies to construct them efficiently.

Groth [16] proposed the first sublinear-communication NIZK argument (computationally-sound proof, [4]) for an **NP**-complete language. His construction was improved by Lipmaa [19]. Their CIRCUIT-SAT argument consists of efficient arguments for more primitive tasks like Hadamard sum, Hadamard product and permutation. The CIRCUIT-SAT arguments of [16,19] have constant communication, quadratic prover's computation, and linear verifier's computation in s (the circuit size). In [16], the CRS length is $\Theta(s^2)$, and in [19], it is $\Theta(r_3^{-1}(s)) = o(s2^{2\sqrt{2\log_2 s}})$, where $r_3(N) = \Omega(N \log^{1/4} N / 2^{2\sqrt{2\log_2 N}})$ [9] is the cardinality of the largest progression-free subset of $[N]$. Because of the quadratic prover's computation, the arguments of Groth and Lipmaa are not applicable in practice, unless s is really small. Very recently, Fauzi, Lipmaa and Zhang [10] constructed arguments for **NP**-complete languages Set Partition, Subset Sum and Decision Knapsack with the CRS length $\Theta(r_3^{-1}(s))$ and prover's computation $\Theta(r_3^{-1}(s) \log s)$. They did not propose a similar argument for the Circuit-SAT.

Gennaro, Gentry, Parno and Raykova [15] constructed a Circuit-SAT NIZK argument based on efficient (quadratic) span programs. Their argument consists of two steps. The first step is an information-theoretic reduction from the Circuit-SAT to QSP-SAT [2], the satisfaction problem of quadratic span programs (QSPs, [15]). The second step consists of cryptographic tools that allow one to succinctly verify the satisfiability of a QSP.

Intuitively, a span program consists of vectors $\boldsymbol{u_i}$ for $i > 0$, a target vector $\boldsymbol{u_0}$, and a labelling of every vector $\boldsymbol{u_i}$ by a literal $x_\iota = x_\iota^1$ or $\bar{x}_\iota = x_\iota^0$ or by $\perp$. A span program accepts an input $\boldsymbol{w}$ iff $\boldsymbol{u_0}$ belongs to the span of the vectors $\boldsymbol{u_i}$ that are labelled by literals $x_\iota^{w_\iota}$ (or by $\perp$) that are consistent with the assignment $\boldsymbol{w} = (w_\iota)$ to the input $\boldsymbol{x} = (x_\iota)$. I.e., $\boldsymbol{u_0} = \sum_{i>0} a_i \boldsymbol{u_i}$, where $a_i \neq 0$ if the labelling of $\boldsymbol{u_i}$ is not consistent with $\boldsymbol{w}$. (See Sect. 3 for more background.)

Briefly, the first step constructs span programs (which satisfy a non-standard conscientiousness property) that verify the correct evaluation of every individual gate. Conscientiousness means that the span program accepts only if all inputs to the span program were actually used (in the case of Circuit-SAT, this means that the prover has set some value to every input and output wire of the gate, and that exactly the same value can be uniquely extracted from the argument). The gate checkers are aggregated to obtain a single large conscientious span program that verifies the operation of every individual gate in parallel. They then construct a *weak wire checker* that verifies consistency, i.e., that all individual gate checkers work on an unequivocally defined set of wire values. The weak wire checker of [15] guarantees consistency only if all gate checkers are conscientious. They define quadratic span programs (QSPs, see [15]) and construct a QSP that implements both the aggregate gate checker and the weak wire checker.

In the second step, Gennaro et al. construct a non-adaptively sound NIZK argument that verifies the QSP, with a linear CRS length, $\Theta(s \log^2 s)$ prover's computation, and linear-in-input size verifier's computation. It can be made adaptively sound by using universal circuits [25], see [15] for more information.

The construction of [15] is quite monolithic and while containing many new ideas, they are not sufficiently clarified in [15]. Bitansky et al [2] simplified the second step of the construction from [15], by first constructing a linear PCP [2], then a linear interactive proof, and finally a NIZK argument for Circuit-SAT. Their more modular approach makes the ideas behind the second step more accessible. Unfortunately, [2] is slightly less efficient than [15], and uses a (presumably) stronger security assumption.

We improve the construction of [15] in several aspects. Some improvements are conceptual (e.g., we provide cleaner definitions, that allow us to offer more efficient constructions) and some of the improvements are technical (with special emphasis on concrete efficiency). More precisely, we modularize — thus making its ideas more clear and accessible — the first step of [15] to construct a succinct non-adaptive 3-query linear PCP [2] for Circuit-SAT. Then we use the techniques of [2], together with several new techniques, to modularize the second step of [15]. Importantly and contrarily to [2], by doing so we both improve on the efficiency of both steps and relax the security assumptions. We outline our construction below, and sketch the differences compared to [15].

The main body of the current work consists of a cleaner and more efficient reduction from Circuit-SAT to QSP-SAT (another **NP**-complete language, defined later). Given a circuit C, we construct an efficient circuit checker, a QSP that is satisfiable iff C is satisfiable.

To verify whether circuit C accepts an input, we use a small *standard* (i.e., not necessarily conscientious) span program to verify an individual gate. For example, a NAND checker is a span program that accepts if the gate implements NAND correctly. We construct efficient span programs for gate checkers, needed for the Circuit-SAT argument. E.g., we construct a size 6 and dimension 3 NAND checker; this can be compared to size 12 and dimension 9 conscientious NAND checker from [15]. By using the AND composition of span programs, we construct a single large span program that verifies every gate in parallel.

Unfortunately, simple AND composition of the gate checkers is not secure, because it allows "double-assignments". More precisely, some vectors of several adjacent gate checkers are labelled by the variable corresponding to the same wire. While every individual checker might be locally correct, one checker could work with value 0 while another checker could work with value 1 assigned to the same wire. Clearly, such bad cases should be detected. More precisely, it must be possible to verify efficiently that the coefficients a_i that were used in the gate checkers adjacent to some wire are consistent with a unique wire value.

We solve this issue as follows. Let Code be an efficient high-distance linear $[N, K, D]$ error-correcting code with $D > N/2$. For any wire η, consider all vectors from adjacent gate checkers that correspond to the claimed value x_η of this wire. Some of those vectors (say $\boldsymbol{u_i}$) are labelled by the positive literal x_η and some (say $\boldsymbol{v_i}$) by the negative literal $\bar{x}_\eta$. The individual gate checker's acceptance "fixes" certain coefficients a_i (that are used with $\boldsymbol{u_i}$) and b_i (that are used with $\boldsymbol{v_i}$) for all adjacent gate checkers. Roughly stating, for consistency of wire η one requires that either all a_i are zero (then unequivocally $x_\eta = 0$), or all b_i are zero (then unequivocally $x_\eta = 1$). We verify that this is the case by applying Code separately to the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. The high-distance property of Code guarantees that if $\boldsymbol{a}$ and $\boldsymbol{b}$ are not consistent, then there exists a coefficient i, s.t. $\mathsf{Code}(\boldsymbol{a})_i \cdot \mathsf{Code}(\boldsymbol{b})_i \neq 0$.

Motivated by this construction, we redefine QSPs [15] as follows. Let $\circ$ denote the pointwise product of two vectors. A QSP (that consists of two target vectors $\boldsymbol{u_0} = (u_{0j}) \in \mathbb{F}^d$ and $\boldsymbol{v_0} = (v_{0j}) \in \mathbb{F}^d$ and two $m \times d$ matrices $U = (u_{ij})$ and $V = (v_{ij})$ for $i \in [m]$ and $j \in [d]$) over some field $\mathbb{F}$ accepts an input iff for some vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, consistent with this input,

$$(\boldsymbol{a}^\top \cdot U - \boldsymbol{u_0}) \circ (\boldsymbol{b}^\top \cdot V - \boldsymbol{v_0}) = \boldsymbol{0} \ . \tag{1}$$

Clearly, Eq. (1) is equivalent to the requirement that for all $j \in [d]$, $(\sum_{i=1}^m a_i u_{ij} - u_{0j}) \cdot (\sum_{i=1}^m b_i v_{ij} - v_{0j}) = 0$. Since $\mathbb{F}$ is an integral domain, the latter holds iff for all $j \in [d]$, either $\sum_{i=1}^{m} a_i u_{ij} = u_{0j}$ or $\sum_{i=1}^{m} b_i v_{ij} = v_{0j}$, which can be seen as an element-wise OR of two span programs. This can be compared to the element-wise AND of two span programs that accepts iff for all $j \in [d]$, both $\sum_{i=1}^{m} a_i u_{ij} = u_{0j}$ and $\sum_{i=1}^{m} b_i v_{ij} = v_{0j}$ iff two span programs accept simultaneously, i.e., $\sum a_i \boldsymbol{u_i} = \boldsymbol{u_0}$ and $\sum b_i \boldsymbol{v_i} = \boldsymbol{v_0}$. On the other hand, it is not known how to implement an element-wise OR composition of two span programs as a small span program. QSPs add an element-wise OR to an element-wise AND, and thus it is not surprising that they increase the expressiveness of span programs significantly.

The above linear error-correcting code based construction implements a QSP (a *wire checker*), with U and V being related to the generating matrices of the code. (See Def. 2.) Basically, the wire checker verifies the consistency of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with the input.

We use the systematic Reed-Solomon code, since it is a maximum distance separable code with optimal support (i.e., it has the minimal possible number of non-zero elements in its generating matrix). It also results in the smallest degree of certain polynomials in the full NIZK argument. While no connection to error-correcting codes was made in [15], their wire checker can be seen as a suboptimal (overdefined) variant of the systematic Reed-Solomon code. Due to the better theoretical foundation, the new wire checker is more efficient, and optimal in its size and support. Moreover, one can use any efficient high-distance ($D > N/2$) linear error-correcting code, e.g., a near-MDS code [7]. Whether this would result in any improvement in the computational complexity of the final NIZK argument is an interesting open question.

Moreover, the wire checker of [15] is consistent (and thus their NIZK argument is sound) only if the gate checkers are conscientious. The new wire checker does not have this requirement. This not only enables one to use more efficient gate checkers but also potentially enables one to use known techniques (combinatorial characterization of span program size [11], semidefinite programming [24]) to construct more efficient checkers for larger unit computations.

We construct an aggregate wire checker by applying an AND composition to wire checkers, and then construct a single QSP (the *circuit checker*) that implements both the aggregate gate checker and the aggregate wire checker. At this point, the approach of the current paper pays off also conceptually: one can compare the description of the circuit checker (called a canonical QSP) in [15, Sect. 2.4], that takes about 3/4 of a page, with the description from the current paper (Def. 3) that takes only a couple of lines.

We prove that the circuit checker (the QSP) is satisfiable iff the original circuit is satisfiable. Since the efficiency of the new circuit checker depends on the fan-out of the circuit, we use the classical result from [17] about constructing low fan-out circuits that allows us to optimize the worst case size and other parameters, especially support, of the circuit checker.

To summarize, the new circuit checker consists of two elements. First, an aggregate gate checker (a span program) that verifies that every individual gate is executed correctly on their local variables. Second, an aggregate wire checker (a QSP, based on a high-distance linear error-correcting code) that verifies that individual gates are executed on the consistent assignments to the variables. Importantly (for the computational complexity of the NIZK argument), the circuit checker is a composition of small (quadratic) span programs, and has only a constant number of non-zero elements per vector.

This finishes the description of the CIRCUIT-SAT to QSP-SAT reduction. To construct an efficient NIZK argument for CIRCUIT-SAT, we need several extra steps. Based on the new circuit checker, we first construct a non-adaptive 2-query linear PCP([2], see Sect. 8 for a definition) for CIRCUIT-SAT with linear communication. This seems to be the first known non-trivial 2-query linear PCP. Moreover, we use a more elaborate extraction technique which, differently from the one from [15], also works with non-conscientious gate checkers. This improves the efficiency of the linear PCP. In particular, the computation of the decision functionality of the linear PCP is dominated by a small constant number of field operations. The same functionality required $\Theta(n)$ operations in [15,2]. Interestingly, this construction by itself is purely linear-algebraic, by using concepts like span programs, linear error-correcting codes, and linear PCPs.

To improve the communication of the linear PCP, as in [15], we define polynomial span programs and polynomial QSPs. Differently from [15] (that only gave the polynomial definition), our main definition of QSPs — as sketched above — is linear-algebraic, and we then use a transformation to get a QSP to a "polynomial" form. We feel the linear-algebraic definition is much more natural, and describes the essence of QSPs better. Based on the polynomial redefinition of QSPs and the Schwartz-Zippel lemma, we construct a succinct non-adaptive 3-query linear PCP for CIRCUIT-SAT. The prover's computation in this linear PCP is $\Theta(s \log s)$, where s is the size of the circuit, and the verifier's computation is again $\Theta(1)$. In [15], the corresponding parameters were $\Theta(s \log^2 s)$ and $\Theta(n)$. Thus, the new 3-query linear PCP is more efficient and conceptually simpler than the previously known 3-query linear PCPs [2].

By using techniques of [2], we convert the linear PCP to a succinct non-adaptive linear interactive proof, and then to a succinct non-adaptive NIZK argument. (See the full version, [20].) As in the case of the argument from [15], the latter can be made adaptive by using universal circuits [25].

Since the reduction from linear PCP to NIZK from [2] loses some efficiency and relies on a stronger security assumption than stated in [15], we also describe a direct NIZK argument with a (relatively complex) soundness proof that follows the outline of the soundness proof from [15]. The main difference in the proof is that we rephrase certain proof techniques from [15] in the language of multilinear universal hash functions. This might be an interesting contribution by itself. Apart from a more clear proof, this results in a slightly weaker security assumption. (See the full version [20] of this paper.)

The new non-adaptive CIRCUIT-SAT argument has CRS length $\Theta(s)$, prover's computation $\Theta(s \log s)$, verifier's computation $\Theta(1)$, and communication $\Theta(1)$. In all cases, the efficiency has been improved as compared to the (QSP-based) argument from [15]. Moreover, all additional optimization techniques applicable to the argument from [15] (e.g., the use of collision-resistant hash functions) are also applicable to the new argument.

We hope that by using our techniques, one can construct efficient NIZK arguments for other languages, like the techniques of [19] were used in [5] to construct an efficient range argument, and in [21] to construct an efficient shuffle. QSPs have more applications than just in the NIZK construction. We only mention that one can construct a related zap [8], and a related (public or designated-verifier) succinct non-interactive argument of knowledge (SNARK, see [22,6]) by using the techniques of [1,14].

It is also natural to apply our techniques to verifiable computation [13]: instead of gates, one can talk about small (but possibly much larger) computational units, and instead of wires, about the values transferred between the computational units. Since here one potentially deals with much larger span programs than in the case of the Circuit-SAT argument, the use of standard (non-conscientious) span programs is especially beneficial. Since in the case of verifiable computation, the computed function F (and thus also the circuit C) is known while generating the CRS, one can use the non-adaptively sound version of the new argument [23].

Gennaro et al. [15] also proposed a NIZK argument that is based on quadratic arithmetic programs (QAP-s), a novel computational model for arithmetic circuits. QAP-based arguments are often significantly more efficient than QSP-based arguments, see [15,23]. We can use our techniques to improve on QAP-based arguments, but here the improvements are less significant and thus we have omitted full discussion. (See the full version.) Briefly, differently from [15], we give an (again, more clean) linear-algebraic definition of QAP-s. This enables us to present a short alternative proof of the result from [15] that any arithmetic circuit with n inputs and s multiplication gates can be computed by a QAP of size $n + s$ and dimension s. We remark that the QAP-based construction results in a 4-query linear PCP, while the QSP-based construction from the current paper results in a 3-query linear PCP.

Due to the lack of space, many proofs are given only in the full version [20].

2 Preliminaries: Circuits and Circuit-SAT

For a fixed circuit C, let $s = |C|$ be its size (the number of gates), s_{e} its number of wires, and n be its input size. Every gate ι computes some unary or binary function $f_\iota : \{0,1\}^{\le 2} \to \{0,1\}$. We denote the set of gates of C by $[s]$ and the set of wires of C by $[s_{\mathsf{e}}]$. Assume that the first n wires, $\eta \in [n]$, start from n input gates $\iota \in [n]$. Every wire $\eta \in [s_{\mathsf{e}}]$ corresponds to a formal variable x_η in a natural way. This variable obtains an assignment w_η, $\eta \in [s_{\mathsf{e}}]$, computed by C from the input assignment $(w_i)_{i=1}^n$. Denote $\boldsymbol{w} := (w_\eta)_{\eta=1}^{s_{\mathsf{e}}}$. We write $C(\boldsymbol{w}) := C((w_i)_{i=1}^n)$. For a gate ι of C, let $\deg^+(\iota)$ be its fan-out, and let $\deg^-(\iota)$ be its fan-in. Let $\deg(\iota) = \deg^-(\iota) + \deg^+(\iota)$.

Let $\mathrm{poly}(x) := x^{O(1)}$. Let $\mathcal{R} = \{(C, \boldsymbol{w})\}$ be an efficiently computable binary relation with $|\boldsymbol{w}| = \mathrm{poly}(|C|)$ and $s := |C| = \mathrm{poly}(|\boldsymbol{w}|)$. Here, C is a statement, and $\boldsymbol{w}$ is a witness. Let $\mathcal{L} = \{C : \exists \boldsymbol{w}, (C, \boldsymbol{w}) \in \mathcal{R}\}$ be the related **NP**-language. For fixed s, we have a relation $\mathcal{R}_s$ and a language $\mathcal{L}_s$.

The *language* CIRCUIT-SAT consists of all (strings representing) circuits that produce a single bit of output and that have a satisfying assignment. That is, a string representing a circuit C is in CIRCUIT-SAT if there exists $\boldsymbol{w} \in \{0,1\}^{s_{\mathsf{e}}}$ such that $C(\boldsymbol{w}) = 1$.

As before, we assume that $s = |C|$ is the number of gates, not the bitlength needed to represent C. Thus, $\mathcal{L}_s = \{C : |C| = s \wedge (\exists \boldsymbol{w} \in \{0,1\}^{s_{\mathsf{e}}}, C(\boldsymbol{w}) = 1)\}$ and $\mathcal{R}_s = \{(C, \boldsymbol{w}) : |C| = s \wedge \boldsymbol{w} \in \{0,1\}^{s_{\mathsf{e}}} \wedge C(\boldsymbol{w}) = 1\}$.

Let $G = (V, E)$ be the hypergraph of the circuit C. The vertices of G correspond to the gates of C. A hyperedge η connects the input gate of some wire to (potentially many) output gates of the same wire. In C, an edge η (except input edges, that have ϕ adjacent vertices) has $\phi + 1$ adjacent vertices, where ϕ is the fan-out of η's designated input gate. Every vertex of G can only be the starting gate of one hyperedge and the final gate of two hyperedges (since we only consider unary and binary gate operations). Thus, $|E(G)| \leq 2(|V(G)| - n)$.

3 Preliminaries: Span Programs

Let $\mathbb{F} = \mathbb{Z}_q$ be a finite field of size $q \gg 2$, where q is a prime. However, most of the results can be generalized to arbitrary fields. By default, vectors like $\boldsymbol{u}$ denote row vectors. For matrix U, let $\boldsymbol{u_i}$ be its ith row vector. For an $m \times d$ matrix U over $\mathbb{F}$, let $\mathsf{span}(U) := \{\sum_{i=1}^{m} a_i \boldsymbol{u_i} : \boldsymbol{a} \in \mathbb{F}^m\}$. Let x_ι, $\iota \in [n]$, be formal variables. Denote the positive literals x_ι by x_ι^1 and the negative literals $\bar{x}_\iota$ by x_ι^0.

A *span program* [18] $P = (\boldsymbol{u_0}, U, \varrho)$ over a field $\mathbb{F}$ is a linear-algebraic computation model. It consists of a non-zero target vector $\boldsymbol{u_0} \in \mathbb{F}^d$, an $m \times d$ matrix U over $\mathbb{F}$, and a labelling $\varrho : [m] \to \{x_\iota, \bar{x}_\iota : \iota \in [n]\} \cup \{\perp\}$ of U's rows by one of $2n$ literals or by $\perp$. Let $U_{\boldsymbol{w}}$ be the submatrix of U consisting of those rows whose labels are satisfied by the assignment $\boldsymbol{w} \in \{0,1\}^n$, that is, belong to $\{x_\iota^{w_\iota} : \iota \in [n]\} \cup \{\perp\}$. *$P$ computes a function f*, if for all $\boldsymbol{w} \in \{0,1\}^n$: $\boldsymbol{u_0} \in \mathsf{span}(U_{\boldsymbol{w}})$ if and only if $f(\boldsymbol{w}) = 1$.

Let $\varrho_{\boldsymbol{w}}^{-1} = \{i \in [m] : \varrho(i) \in \{x_\iota^{w_\iota} : \iota \in [n]\} \cup \{\perp\}\}$ be the set of rows whose labels are satisfied by the assignment $\boldsymbol{w}$. The *size,* $\mathsf{size}(P)$*,* of P is m. The dimension, $\mathsf{sdim}(P)$, is equal to d. P has *support* $\mathsf{supp}(P)$, if all vectors $\boldsymbol{u} \in U$ have altogether $\mathsf{supp}(P)$ non-zero elements. Clearly, $\boldsymbol{u_0}$ can be replaced by an arbitrary non-zero vector; one obtains the corresponding new span program (of the same size and dimension, but possibly different support) by applying a basis change matrix. Let $D(x_\iota) := \max_{j \in \{0,1\}} |\varrho^{-1}(x_\iota^j)|$, for each $\iota \in [n]$ and $j \in \{0,1\}$, be the maximum number of vectors that have the same label (ι, j); this parameter is needed when we construct wire checkers.

Complex span programs are constructed by using simple span programs and their composition rules. The Boolean function NAND $\bar{\wedge}$ is defined as $\bar{\wedge}(x,y) = x \bar{\wedge} y = \neg(x \wedge y)$. Span programs for AND, NAND, OR, XOR, and equality of two variables x and y are as in Fig. 1. Given span programs $P_0 = SP(f_0)$

$$\left(\begin{array}{c|cc} & 1 & 1 \\ \hline x & 1 & 0 \\ y & 0 & 1 \end{array}\right) \left(\begin{array}{c|c} & 1 \\ \hline \bar{x} & 1 \\ \bar{y} & 1 \end{array}\right) \left(\begin{array}{c|c} & 1 \\ \hline x & 1 \\ y & 1 \end{array}\right) \left(\begin{array}{c|rr} & 0 & 1 \\ \hline x & 1 & 1 \\ y & 1 & 1 \\ \bar{x} & -1 & 0 \\ \bar{y} & -1 & 0 \end{array}\right) \left(\begin{array}{c|rr} & 0 & 1 \\ \hline x & 1 & 1 \\ y & -1 & 0 \\ \bar{x} & -1 & 0 \\ \bar{y} & 1 & 1 \end{array}\right) \left(\begin{array}{c|ccc} & 1 & 1 & 1 \\ \hline x & 1 & 0 & 0 \\ y & 0 & 1 & 0 \\ z & 1 & 1 & 0 \\ \bar{x} & 0 & 0 & 1 \\ \bar{y} & 0 & 0 & 1 \\ \bar{z} & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{c|ccc} & 1 & 1 & 1 \\ \hline x & 0 & 1 & 0 \\ y_1 & 0 & 0 & 1 \\ y_2 & 1 & 0 & 0 \\ \bar{x} & 1 & 0 & 0 \\ \bar{y}_1 & 0 & 1 & 0 \\ \bar{y}_2 & 0 & 0 & 1 \end{array}\right)$$

Fig. 1. From left to right: standard span programs $SP(\wedge)$, $SP(\bar{\wedge})$, $SP(\vee)$, $SP(\oplus)$, $SP(=)$ and new span programs $SP(c_{\bar{\wedge}})$ and $SP(c_{\mathsf{Y}})$

an $P_1 = SP(f_1)$ for functions f_0 and f_1, one uses well-known AND and OR compositions to construct span programs for $f_0 \wedge f_1$ and $f_0 \vee f_1$.

A span program $(\boldsymbol{u_0}, U, \varrho)$ is *conscientious* [15] if a linear combination associated to a satisfying assignment must use at least one vector associated to either x_ι or $\bar{x}_\iota$ for every $\iota \in [n]$. Clearly, $SP(\wedge)$, $SP(\oplus)$ and $SP(=)$ are conscientious, while $SP(\vee)$ is not.

4 Efficient Gate Checkers

A *gate checker* for a gate that implements $f : \{0,1\}^n \to \{0,1\}$ is a function $c_f : \{0,1\}^{n+1} \to \{0,1\}$, s.t. $c_f(\boldsymbol{x}, y) = 1$ iff $f(\boldsymbol{x}) = y$. The NAND-checker $c_{\bar{\wedge}} : \{0,1\}^3 \to \{0,1\}$ outputs 1 iff $z = x \bar{\wedge} y$.

Lemma 1. *$SP(c_{\bar{\wedge}})$ on Fig. 1 is a span program for $c_{\bar{\wedge}}$. It has size 6, dimension 3, and support 7.*

As seen from the proof , given an accepting assignment (x, y, z), one can efficiently find small values $a_i \in [-2, 1]$ such that $\sum_{i \geq 1} a_i \boldsymbol{u_i} = \boldsymbol{u_0}$. However, a satisfying input to $SP(c_{\bar{\wedge}})$ does not fix the values a_i unequivocally: if $(x, y, z) = (0, 0, 1)$ (that is, $a_1 = a_2 = a_6 = 0$), then one can choose an arbitrary a_4 and set $a_5 \leftarrow 1 - a_4$. Since one can set $a_4 = 0$, $SP(c_{\bar{\wedge}})$ is not conscientious.

Given $SP(c_{\bar{\wedge}})$, one can construct a size 6 and dimension 3 span program for the AND-checker $c_\wedge(x, y, z) := (x \wedge y) \oplus \bar{z}$ by interchanging in $SP(c_{\bar{\wedge}})$ the rows labelled by z and $\bar{z}$. Similarly, one can construct a size 6 and dimension 3 span program for the OR-checker $c_\vee(x, y, z) := (\bar{x} \wedge \bar{y}) \oplus z$ by interchanging in $SP(c_{\bar{\wedge}})$ the rows labelled by x and $\bar{x}$, and the rows labelled by y and $\bar{y}$. NOT-checker $[x \neq y] = x \oplus y$ is just the XOR function, and thus one can construct a size 4 and dimension 2 span program for the NOT-checker function.

We need the dummy gates $y \leftarrow x$, and corresponding dummy checkers $c_=(x, y) = [x = y]$. Clearly, the dummy checker function is just to the equality test, and thus has a conscientious span program of size 4 and dimension 2. Moreover, if $x = y \in \{0,1\}$, then $a_1 = a_2 = x$, while $a_3 = a_4 = 1 - x$.

We need the fork-checker $c_{\mathsf{Y}}(x, y_1, y_2)$ for the fork gate that computes $y_1 \leftarrow x$, $y_2 \leftarrow x$. In the CNF form, $c_{\mathsf{Y}}(x, y_1, y_2) = (\bar{x} \vee y_2) \wedge (x \vee \bar{y}_1) \wedge (y_1 \vee \bar{y}_2)$. Since every literal is mentioned once in the CNF, we can use AND and OR compositions to derive the span program on Fig. 1. It has size 6, dimension 3, and support 6. We also need a 1-to-ϕ fork-checker that has 1 input x and ϕ outputs y_ι, with $y_\iota \leftarrow x$. The ϕ-fork checker is $c_{\mathsf{Y}}^{\phi}(x, \boldsymbol{y}) = (x \wedge y_1 \wedge \cdots \wedge y_\phi) \vee (\bar{x} \wedge \bar{y}_1 \wedge \cdots \wedge \bar{y}_\phi)$. Clearly, c_{Y}^{ϕ} has CNF $c_{\mathsf{Y}}^{\phi}(x, \boldsymbol{y}) = (x \vee \bar{y}_1) \wedge (y_1 \vee \bar{y}_2) \wedge \cdots \wedge (y_{\phi-1} \vee \bar{y}_\phi) \wedge (y_\phi \vee \bar{x})$. From this we construct a span program exactly as in the case $\phi = 2$, with size $2(\phi + 1)$ and dimension $\phi + 1$. It has only one vector labelled with every x_ι/y or its negation, thus $D(x) = D(y_\iota) = 1$ for all ι. To compute the support, note that $SP^*(c_{\mathsf{Y}}^{\phi})$ has two 1-entries in every column, and one in every row. Thus, $\mathsf{supp}(SP(c_{\mathsf{Y}}^{\phi})) = \sum_{i=1}^{\phi+1} 2 = 2\phi + 2$.

5 Aggregate Gate Checker

Given a circuit that consists of NAND, AND, OR, XOR, and NOT gates, we combine the individual gate checkers by using the AND composition rule. In addition, for the wire checker of Sect. 6.2 (and thus also the final NIZK argument) to be more efficient, all gates of the circuit C need to have a small fan-out. In [15], the authors designed a circuit of size $3 \cdot |C|$ that implements the functionality of C but only has fan-out 2 except for a specially introduced dummy input. Their aggregate gate checker (AGC) has size $36 \cdot |C|$ and dimension $27 \cdot |C|$. By using the techniques of [17] (that replaces every high fan-out gate with an inverse binary tree of fork gates, and then gives a more precise upper bound of the resulting circuit size), we prove a more precise result. We do not introduce the dummy input but we still add a dummy gate for every input. We then say that we deal with a circuit with dummy gates.

Since we are interested in circuit satisfiability, the X-checker (where say X = NAND) of the circuit's output gate simplifies to the X gate (e.g., NAND checker simplifies to NAND). Since X has a more efficient span program than X checker, then for the sake of simplicity, we will not mention this any more.

Let C be a circuit. The *AGC function* agc of a circuit C is a function $\mathsf{agc} : \{0,1\}^{\sum_{\iota=1}^{|C|} \deg(\iota)} \rightarrow \{0,1\}^{|C|}$. I If c_ι is the gate checker of the ιth gate and $\boldsymbol{x_\iota}$ has dimension $\deg(\iota)$, then $\mathsf{agc}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_{|C|}}) = (c_1(\boldsymbol{x_1}), \ldots, c_{|C|}(\boldsymbol{x_{|C|}}))$.

As in [15], we construct the AGC by AND-composition of the gate checkers of the individual gate checkers. Since for an individual gate checker and a satisfying assignment, one can compute the corresponding coefficient vector $\boldsymbol{a}$ in constant time, the aggregate coefficient vector $\boldsymbol{a}$ can be computed from w in time $\Theta(s)$. Let $\boldsymbol{a} \leftarrow \mathsf{c2q}(w)$ be the corresponding algorithm.

Theorem 1. *Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be the function computed by a fan-in ≤ 2 circuit C with $s = |C|$ NAND, AND, OR, XOR, and NOT gates. There exists a fan-in ≤ 2 and fan-out $\leq \phi$ circuit with dummy gates C_{bnd} for f, that has the same s gates as C, n additional dummy gates, and up to $(s - 2n)/(\phi - 1)$ additional ϕ-fork gates. Let $\phi^* := 1/(\phi - 1)$. The AGC $\mathsf{agc}(C_{\mathsf{bnd}})$ has a span program P with $\mathsf{size}(P) \leq (8 + 4\phi^*)\, s - (6 + 8\phi^*)\, n$, $\mathsf{sdim}(P) \leq (4 + 2\phi^*)\, s - (3 + 4\phi^*)\, n$, and $\mathsf{supp}(P) \leq (9 + 4\phi^*)\, s - (5 + 8\phi^*)\, n$. If $\phi = 3$, then $\mathsf{size}(P) \leq 10s - 10n$, $\mathsf{sdim}(P) \leq 5s - 5n$, and $\mathsf{supp}(P) \leq 11s - 9n$.*

The upper bounds of this theorem are worst-case, and often imprecise. The optimal choice of ϕ depends on the parameter that we are going to optimize. The AGC has optimal size, dimension and support if ϕ is large (preferably even if the fan-out bounding procedure of Thm. 1 is not applied at all). The support of the aggregate wire checker (see Sect. 6.3) is minimized if $\phi = 2$. To balance the parameters, we concentrate on the case $\phi = 3$.

6 Quadratic Span Programs and Wire Checker

6.1 Quadratic Span Programs

An intuitive definition of quadratic span programs (QSPs) was given in the introduction and will not be repeated here. We now give a formal (linear-algebraic) definition of QSPs. In Sect. 9, we will provide an equivalent polynomial redefinition of QSPs that is the same as the definition given in [15].

Definition 1. *A* quadratic span program (QSP) $Q = (\boldsymbol{u_0}, \boldsymbol{v_0}, U, V, \varrho)$ *over a field* $\mathbb{F}$ *consists of two target vectors* $\boldsymbol{u_0}, \boldsymbol{v_0} \in \mathbb{F}^d$*, two* $m \times d$ *matrices* U *and* V*, and a common labelling* $\varrho : [m] \to \{x_\iota, \bar{x}_\iota : \iota \in [n]\} \cup \{\perp\}$ *of the rows of* U *and* V*.* Q *accepts an input* $\boldsymbol{w} \in \{0,1\}^n$ *iff there exist* $(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{F}^m \times \mathbb{F}^m$*, with* $a_i = 0 = b_i$ *for all* $i \notin \varrho_{\boldsymbol{w}}^{-1}$*, such that* $(\boldsymbol{a}^\top \cdot \mathcal{V} - \boldsymbol{u_0}) \circ (\boldsymbol{b}^\top \cdot \mathcal{W} - \boldsymbol{v_0}) = \boldsymbol{0}$*, where* $\boldsymbol{x} \circ \boldsymbol{y}$ *denotes the pointwise (Hadamard) product of* $\boldsymbol{x}$ *and* $\boldsymbol{y}$*.* Q computes a function f *if for all* $\boldsymbol{w} \in \{0,1\}^n$*:* $f(\boldsymbol{w}) = 1$ *iff* Q *accepts* $\boldsymbol{w}$*.*

We remark that one can have $\boldsymbol{u_0} = \boldsymbol{v_0} = \boldsymbol{0}$. (See Def. 2, for example.)

The size, $\mathsf{size}(Q)$, of Q is m. The dimension, $\mathsf{sdim}(Q)$, of Q is d. The *support*, $\mathsf{supp}(Q)$, of Q is equal to the sum of the supports (that is, the number of non-zero elements) of all vectors $\boldsymbol{u_i}$ and $\boldsymbol{v_i}$. Clearly, one can compose QSPs by using the AND and OR composition rules of span programs, though one has to take care to apply the same transformation to both U and V simultaneously.

The language QSP-SAT consists of all (strings representing) QSPs that produce a single bit of output and that have a satisfying assignment. I.e., a string representing an n-input QSP Q is in QSP-SAT if there exists $\boldsymbol{w} \in \{0,1\}^n$, such that $Q(\boldsymbol{w}) = 1$. The witness of this fact is $(\boldsymbol{a}, \boldsymbol{b})$, and we write $Q(\boldsymbol{a}, \boldsymbol{b}) = Q(\boldsymbol{w})$.

6.2 Wire Checker

Gate checkers verify that every individual gate is followed correctly, i.e., that its output wire obtains a value which is consistent with its input wires. One also requires inter-gate (wire) consistency that ensures that adjacent gate checkers do not make double assignments to any of the wires. Here, we consider hyperwires that have one input gate and potentially many output gates. Following [15], for this purpose we construct a wire checker. We first construct a wire checker for every single wire (that verifies that the variables involved in the span programs of the vertices that are adjacent to this concrete wire do not get inconsistent assignments), and then aggregate them by using an AND composition.

For a (hyper)wire η, let $N(\eta)$ be the set of η's adjacent gates. For gate $\iota \in N(\eta)$, let $P_\iota = (\boldsymbol{u}_\mathbf{0}^{(\boldsymbol{\iota})}, U^{(\iota)}, \varrho^{(\iota)})$ be its gate checker. For every $\iota \in N(\eta)$, one of the input or output variables of P_ι (that we denote by $x_{\iota:\eta}$) corresponds to x_η. Recall that for a local variable y of a span program P_ι, $D(y) = \max(|\varrho^{-1}(y)|, |\varrho^{-1}(\bar{y})|)$. We assume $|\varrho^{-1}(y)| = |\varrho^{-1}(y)|$, by adding zero vectors to the span programs if necessary. Let $D(\eta) := \sum_{\iota \in N(\eta)} D(x_{\iota:\eta})$ be the number of the times the rows of adjacent gate checkers have been labelled by a local copy of x^1_η.

We define the ηth wire checker between the rows of adjacent gates $i \in N(\eta)$ in the AGC that are labelled either by the local variable $x_{i:\eta}$ or its negation $\bar{x}_{i:\eta}$, i.e., between $2D(\eta)$ rows $\{i : \exists k \in N(\eta) \text{ s.t. } \varrho^{(k)}(i) = x_{k:\eta} \vee \varrho^{(k)}(i) = \bar{x}_{k:\eta}\}$. Let ψ be the natural labelling of the wire checkers, with $\psi(i) = x^j_\eta$ iff $\varrho^{(k)}(i) = x^j_{k:\eta}$ for some $k \in N(\eta)$.

Example 1. Consider a (hyper)wire η that has one input gate ι_1 and two output gates ι_2 and ι_3. Assume that all three gates implement NAND, and thus they have gate checkers $SP(c_{\bar{\wedge}})$ from Fig. 1. Assume that $x_\eta = z_{\iota_1} = x_{\iota_2} = y_{\iota_3}$. Thus, the ηth wire checker is defined between the rows 3 and 6 of the checker for ι_1, rows 1 and 4 of the checker for ι_2, and rows 2 and 5 of the checker for ι_3. Thus, $D(\eta) = D(z_{\iota_1}) + D(x_{\iota_2}) + D(y_{\iota_3}) = 6$. □

We first define the wire checker for a wire η and thus for one variable x_η. In Sect. 6.3, we will give a definition and a construction in the aggregate case.

For $\boldsymbol{y} = (y_1, \dots, y_{2D})^\top$, let $\boldsymbol{y}^{(1)} := (y_1, \dots, y_D)^\top$ and $\boldsymbol{y}^{(2)} := (y_{D+1}, \dots, y_{2D})^\top$. Fix a wire η. Assume that $D = D(\eta)$. Let $Q = (\boldsymbol{u}_\mathbf{0}, \boldsymbol{v}_\mathbf{0}, U, V, \psi)$, with $m \times d$ matrices U and V, be a QSP. Q is a *wire checker*, if for any $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^{2D}$, Eq. (1) holds iff $\boldsymbol{a}$ and $\boldsymbol{b}$ are consistent bit assignments in the following sense: for both $k \in \{1, 2\}$, either $\boldsymbol{a}^{(k)} = \mathbf{0}$ or $\boldsymbol{b}^{(k)} = \mathbf{0}$.

We propose a new wire checker that is based on the properties of high-distance linear error-correcting codes, see the introduction for some intuition. To obtain optimal efficiency, we choose particular codes (namely, systematic Reed-Solomon codes).

Definition 2. *Let $D^* := 2D - 1$. Let RS_D be the $D \times D^*$ generator matrix of the $[D^*, D, D]_q$ systematic Reed-Solomon code. Let $m = 2D$ and $d = 2D^*$. Let $U = U_D = \begin{pmatrix} RS_D & 0_{D\times D^*} \\ 0_{D\times D^*} & RS_D \end{pmatrix}$ and $V = V_D = \begin{pmatrix} 0_{D\times D^*} & RS_D \\ RS_D & 0_{D\times D^*} \end{pmatrix}$. Let $Q_{\mathsf{wc}} := (\mathbf{0}, \mathbf{0}, U, V, \psi)$, where $\psi^{-1}(\bar{x}_\eta) = [1, D]$ and $\psi^{-1}(x_\eta) = [D+1, 2D]$.*

We informally define the degree $\mathsf{sdeg}(Q)$ of a (quadratic) span program Q as the degree of the interpolating polynomial that obtains the value u_{ij} at point j. See Sect. 9 for a formal definition.

Lemma 2. *Q_{wc} is a wire checker of size $2D$, degree $D+D^* = 3D-1$, dimension $2D^* = 4D - 2$, and support $4D^2$.*

Proof. The claim about the parameters follows straightforwardly from the properties of the code. It is easy to see that if $\boldsymbol{a}$ and $\boldsymbol{b}$ are consistent bit assignments, then Q_{wc} accepts. For example, if $\boldsymbol{a}^{(\mathbf{1})} = \boldsymbol{b}^{(\mathbf{2})} = \mathbf{0}$, then clearly

$(\boldsymbol{a}^\top \cdot U)_j = \sum_{i=1}^m a_i u_{ij} = 0$ for $j \in [1, D^*]$ and $(\boldsymbol{b}^\top \cdot V)_j = \sum_{i=1}^m b_i v_{ij} = 0$ for $j \in [D^* + 1, 2D^*]$. Thus, $(\boldsymbol{a}^\top \cdot U)_j \cdot (\boldsymbol{b}^\top \cdot V)_j = 0$ for $j \in [1, 2D^*]$, and thus $(\boldsymbol{a}^\top \cdot U - \boldsymbol{0}) \circ (\boldsymbol{b}^\top \cdot V - \boldsymbol{0}) = \boldsymbol{0}$.

Now, assume that $\boldsymbol{a}$ and $\boldsymbol{b}$ are inconsistent bit assignments, i.e., $\boldsymbol{a}^{(k)} \neq \boldsymbol{0}$ and $\boldsymbol{b}^{(k)} \neq \boldsymbol{0}$ for $k \in \{1, 2\}$. W.l.o.g., let $k = 1$. Since RS_D is the generator matrix of the systematic Reed-Solomon code, the vectors $\boldsymbol{a}^\top \cdot RS_D$ and $\boldsymbol{b}^\top \cdot RS_D$ have at least $D > D^*/2$ non-zero coefficients among its first D^* coefficients. Thus, both $\sum_{i=1}^m a_i u_{ij}$ and $\sum_{i=1}^m b_i v_{ij}$ are non-zero for more than $D^*/2$ different values $j \in [D^*]$. Hence, there exists a coefficient $j \in [D^*]$, such that $(\sum_{i=1}^m a_i u_{ij})(\sum_{i=1}^m b_i v_{ij}) \neq 0$. Thus, Q_{wc} does not accept. $\square$

We chose a Reed-Solomon code since it is a maximum distance separable (MDS) code and thus minimizes the number of columns in RS_D. It also naturally minimizes the degree of the wire checker. Moreover, RS_D has D^2 non-zero elements. Clearly (and this is the reason we use a systematic code), D^2 is also the smallest support a generator matrix G of an $[n = 2D - 1, k = D, d = D]_q$ code can have, since every row of G is a codeword and thus must have at least d non-zero entries. Thus, G must have at least $dD \geq D^2$ non-zero entries, where the last inequality is due to the singleton bound.

The (weak) wire checker of [15], while described by using a completely different terminology, can be seen as implementing an overdefined version (with $D^* = 3D - 2$) of the construction from Def. 2. The linear-algebraic reinterpretation of QSPs together with the introducing of coding-theoretic terminology allowed us to better exposit the essence of wire checkers. It also allowed us to improve on the efficiency, and prove the optimality of the new construction.

A wire checker with $U = V = RS_D$ satisfies the even stronger security requirement that Eq. (1) holds iff either $\boldsymbol{a} = 0$ or $\boldsymbol{b} = 0$. One may hope to pair up literals corresponding to x_η in the U part and literals corresponding to $\bar{x}_\eta$ in the V part. This is impossible in our application: when we aggregate the wire checkers, we must use vectors labelled with both negative and positive literals in the same part, U or V, and we cannot pair up columns from U and V that have different indices. (See Def. 3.) The construction of Def. 2 allows one to do it, though one has to use V that is a dual of U according to the following definition.

For a labelling ψ, we define the *dual labelling* ψ_{dual}, such that $\psi_{\mathsf{dual}}(i) = x_\eta^j$ iff $\psi(i) = x_\eta^{1-j}$. Let $V = U_{\mathsf{dual}}$ be the same matrix as U, except that it has rows from $\psi^{-1}(\bar{x}_\eta)$ and $\psi^{-1}(x_\eta)$ switched, for every η. To simplify the notation, we will not mention the dual labelling ψ_{dual} unless absolutely necessary, and we will assume implicitly that (as it was in Def. 2) always $V = U_{\mathsf{dual}}$. Now, [15] constructed a *weak* wire checker that guarantees consistency if all individual gate checkers are conscientious. The new wire checker is both more efficient and more secure.

6.3 Aggregate Wire Checker

Let $Q = (\boldsymbol{0}, \boldsymbol{0}, U, V, \psi)$, with two $m \times d$ matrices U and $V = U_{\mathsf{dual}}$, be a QSP. Q is *an aggregate wire checker (AWC)* for circuit C, if Eq. (1) holds iff $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^m$ are consistent bit assignments in the following sense: for each $\eta \in [s_{\mathsf{e}}]$ and $k \in \{0,1\}$, either $a_i = 0$ for all $i \in \psi^{-1}(x_\eta^k)$ or $b_i = 0$ for all $i \in \psi^{-1}(x_\eta^k)$.

We construct the AWC by AND-composing wire checkers for the individual wires. The AWC first resets all vectors $\boldsymbol{u_i}$ and $\boldsymbol{v_i}$ to $\mathbf{0}$, and precomputes RS_{D_η} for all relevant values $D_\eta \le 2(\phi + 1)$. After that, for every wire η, it sets the entries in rows, labelled by either x_η or $\bar{x}_\eta$, and columns corresponding to wire η, according to the ηth wire checker.

We recall from Sect. 6.2 that for the wire checker of some wire to work, the vectors in U and V of this wire checker must have dual orderings. To keep notation simple, we will not mention this in what follows.

Theorem 2. *Let $\phi \ge 2$. Assume that C_{bnd} is the circuit, obtained by the transformation described in Thm. 1 (including the added dummy gates). For $\eta \in E(C_{\mathsf{bnd}})$, denote $D_\eta^* = 2D_\eta - 1$. Let $d \leftarrow \sum D_\eta^*$. We obtain the AWC Q_{awc} by merging wire checkers for the individual wires $\eta \in E(C_{\mathsf{bnd}})$ as described above.*

Proof. Let m be the size of the AWC (see Thm. 3). If $\boldsymbol{a}$, $\boldsymbol{b}$ are consistent assignments, then their restrictions to $\psi^{-1}(\bar{x}_\eta) \cup \psi^{-1}(x_\eta)$ are consistent assignments of the ηth wire. For every $\eta \in E(C_{\mathsf{bnd}})$, the ηth wire checker guarantees that $(\sum_{i=1}^m a_i u_{ij})(\sum_{i=1}^m b_i v_{ij}) = 0$, for columns j corresponding to this wire, iff the bit assignments of the ηth wire are consistent. Thus, $(\sum_{i=1}^m a_i u_{ij})(\sum_{i=1}^m b_i v_{ij}) = 0$ for $j \in [1, d]$ iff the bit assignments of all wires are consistent. $\Box$

Theorem 3. *Let $\phi^* := 1/(\phi - 1)$. Assume C implements $f : \{0,1\}^n \to \{0,1\}$, and $s = |C|$. Then $\mathsf{size}(Q_{\mathsf{awc}}) \le (6 + 4\phi^*)s - (2 + 8\phi^*)n - 4$, $\mathsf{sdim}(Q_{\mathsf{awc}}) \le (12 + 8\phi^*)s - (6 + 16\phi^*)n - 8$, $\mathsf{sdeg}(Q_{\mathsf{awc}}) \le (9 + 6\phi^*)s - (4 + 12\phi^*)n - 6$, $\mathsf{supp}(Q_{\mathsf{awc}}) \le 4(\phi + 1)^2((1 + \phi^*)s + (4 - 2\phi^*)n - 1)$. If $\phi = 3$, then $\mathsf{size}(Q_{\mathsf{awc}}) \le 8s - 6n - 4$, $\mathsf{sdim}(Q_{\mathsf{awc}}) \le 16s - 14n - 8$, $\mathsf{sdeg}(Q_{\mathsf{awc}}) \le 12s - 10n - 6$, and $\mathsf{supp}(Q_{\mathsf{awc}}) \le 72s - 68n - 36$.*

Clearly, other parameters but support are minimized when ϕ is large. If support is not important, then one can dismiss the bounding fan-out step, and get size $2s$, dimension $12s$, and degree $9s$.

Like in the case of wire checkers, [15] constructed a *weak AWC* that guarantees the required "no double assignments" property only if the individual gate checkers are conscientious. The new AWC does not have this restriction. The size of the weak AWC from [15] is $24s$ and the degree of it is $76s$.

7 Circuit Checker

Next, we combine the aggregate gate and wire checkers into a circuit checker, that can be seen as a reduction from Circuit-SAT to QSP-SAT. Circuit checker was called a canonical quadratic span program in [15]. Since [18] introduced canonical span programs in a completely different context, we changed the terminology.

Let C be a circuit, and let $P^{\mathsf{w}} = (\mathbf{0}, \mathbf{0}, U^{\mathsf{w}}, V^{\mathsf{w}}, \psi)$ be an AWC for C_{bnd}. Let $P^{\mathsf{g}} = (\boldsymbol{u_0}, U^{\mathsf{g}}, \varrho)$ be an AGC for C_{bnd}. Let $P^{\mathsf{g}}_{\mathsf{dual}} = (\boldsymbol{u_0}, V^{\mathsf{g}}, \varrho_{\mathsf{dual}})$ be the corresponding dual span program. As before, $V^{\mathsf{g}} = U^{\mathsf{g}}_{\mathsf{dual}}$ and $V^{\mathsf{w}} = U^{\mathsf{w}}_{\mathsf{dual}}$, and ϱ and ψ are related as in Sect. 6.3. Let $m_{\mathsf{g}} = \mathsf{size}(P^{\mathsf{w}}) = \mathsf{size}(P^{\mathsf{g}}) = \mathsf{size}(P^{\mathsf{g}}_{\mathsf{dual}})$. Assume that $U^{\mathsf{w}} = \{\boldsymbol{u}^{\mathsf{w}}_{\boldsymbol{1}}, \dots, \boldsymbol{u}^{\mathsf{w}}_{\boldsymbol{m}_{\mathsf{g}}}\}$ and $U^{\mathsf{g}} = \{\boldsymbol{u}^{\mathsf{g}}_{\boldsymbol{1}}, \dots, \boldsymbol{u}^{\mathsf{g}}_{\boldsymbol{m}_{\mathsf{g}}}\}$ (and similarly, $V^{\mathsf{w}} = \{\boldsymbol{v}^{\mathsf{w}}_{\boldsymbol{1}}, \dots, \boldsymbol{v}^{\mathsf{w}}_{\boldsymbol{m}_{\mathsf{g}}}\}$ and V^{g}) are ordered consistently (see Sect. 6.3).

Definition 3. *For $m_{\mathsf{g}} = \mathsf{size}(P^{\mathsf{g}})$, $d_{\mathsf{g}} = \mathsf{sdim}(P^{\mathsf{g}})$ and $d_{\mathsf{w}} = \mathsf{sdim}(P^{\mathsf{w}})$, define the* circuit checker *to be the QSP $c_\Lambda(C) = (\boldsymbol{u_0}, \boldsymbol{v_0}, U, V, \varrho)$, where*

$$\begin{pmatrix} \boldsymbol{u_0} \\ U \\ \hline \boldsymbol{v_0} \\ V \end{pmatrix} = \left(\begin{array}{ccc} \boldsymbol{u_0} & \boldsymbol{1}_{d_{\mathsf{g}}} & \boldsymbol{0}_{d_{\mathsf{w}}} \\ U^{\mathsf{g}} & 0_{m_{\mathsf{g}} \times d_{\mathsf{g}}} & U^{\mathsf{w}} \\ \hline \boldsymbol{1}_{d_{\mathsf{g}}} & \boldsymbol{u_0} & \boldsymbol{0}_{d_{\mathsf{w}}} \\ 0_{m_{\mathsf{g}} \times d_{\mathsf{g}}} & V^{\mathsf{g}} & V^{\mathsf{w}} \end{array}\right) . \tag{2}$$

Here, $U = (\boldsymbol{u_1}, \dots, \boldsymbol{u_m})^\top$, $V = U_{\mathsf{dual}} = (\boldsymbol{v_1}, \dots, \boldsymbol{v_m})^\top$.

Recall that we denoted by $\mathsf{c2q}$ that computed the witness $\boldsymbol{a}$ of the AGC from w. We also denote $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow \mathsf{c2q}(w)$, given that $\boldsymbol{b}$ is the dual of $\boldsymbol{a}$.

Theorem 4. *Let $\boldsymbol{w} \in \{0,1\}^{s_{\mathsf{e}}}$. $C(\boldsymbol{w}) = 1$ iff $c_\Lambda(C)(\mathsf{c2q}(\boldsymbol{w})) = 1$.*

Proof. Clearly, $c_\Lambda(C)(\boldsymbol{a}, \boldsymbol{b}) = 1$ iff P^{g}, $P^{\mathsf{g}}_{\mathsf{dual}}$ and P^{w} all accept with the same witness $(\boldsymbol{a}, \boldsymbol{b})$: (i) $(\sum_{i=1}^m a_i u^{\mathsf{g}}_{ij} - u_{0j})(0-1) = 0$ for $j \in [d_{\mathsf{g}}]$ iff $\sum_{i=1}^m a_i u^{\mathsf{g}}_{ij} = u_{0j}$ for $j \in [d_{\mathsf{g}}]$ iff $\sum_{i=1}^m a_i \boldsymbol{u}^{\mathsf{g}}_{\boldsymbol{i}} = \boldsymbol{u_0}$, (ii) $(0-1)(\sum_{i=1}^m b_i v^{\mathsf{g}}_{ij} - u_{0j}) = 0$ for $j \in [d_{\mathsf{g}}]$ iff $\sum_{i=1}^m b_i v^{\mathsf{g}}_{ij} = u_{0j}$ for $j \in [d_{\mathsf{g}}]$ iff $\sum_{i=1}^m b_i \boldsymbol{v}^{\mathsf{g}}_{\boldsymbol{i}} = \boldsymbol{u_0}$, (iii) $(\sum_{i=1}^m a_i u^{\mathsf{w}}_{ij}) \cdot (\sum_{i=1}^m b_i v^{\mathsf{w}}_{ij}) = 0$ for $j \in [d_{\mathsf{w}}]$.

Assume $C(\boldsymbol{w}) = 1$. By the construction of P^{g}, there exists $\boldsymbol{a} \in \mathbb{F}^m$, with $a_i = 0$ for $i \notin \psi^{-1}_{\boldsymbol{w}}$, s.t. $\boldsymbol{a}^\top \cdot U^{\mathsf{g}} = \boldsymbol{u_0}$. Let $\boldsymbol{b} \leftarrow \boldsymbol{a}$, then also $\boldsymbol{b}^\top \cdot V^{\mathsf{g}} = \boldsymbol{u_0}$. Since $\boldsymbol{a}$ and $\boldsymbol{b}$ are consistent bit assignments in the evaluation of $C(\boldsymbol{w})$, P^{w} accepts.

Second, assume that there exist $(\boldsymbol{a}, \boldsymbol{b})$, s.t. $c_\Lambda(C)(\boldsymbol{a}, \boldsymbol{b}) = 1$. Since P^{w} accepts, there are no double assignments. That means, that for each η, for some (possibly non-unique) bit $w_\eta \in \{0,1\}$ and all $i \in \psi^{-1}(x_\eta^{\overline{w}_\eta})$, $a_i = 0$. Dually, $b_i = 0$ for all $i \in \psi^{-1}_{\mathsf{dual}}(x_\eta^{\overline{w}_\eta})$ (w_η clearly has to be the same in both cases). Since this holds for every wire, there exists an assignment $\boldsymbol{w}$ of input values, s.t. for all $i \notin \psi^{-1}_{\boldsymbol{w}}$ and $j \notin (\psi^{-1}_{\mathsf{dual}})_{\boldsymbol{w}}$, $a_i = b_j = 0$. Moreover, $C(\boldsymbol{w}) = 1$. $\square$

We will explain in the full version how the parameters of $Q := c_\Lambda(C)$ influence the efficiency of the Circuit-SAT NIZK argument. For example, the support of Q affects the prover's computation, while its degree d affects the CRS length but also the prover's computation and the security assumption. More precisely, the prover's computation of the non-adaptive NIZK argument is $\Theta(\mathsf{supp}(Q) + d \cdot \log d)$ non-cryptographic operations and $\Theta(d)$ cryptographic operations. One should choose ϕ such that the prover's computation will be minimal. This value depends on the constants in Θ. For simplicity, we will consider the case $\phi = 3$.

Theorem 5. *Let $s = |C|$ and $Q := c_\Lambda(C_{\mathsf{bnd}})$. Let ϕ be the fanout of C_{bnd}, and $\phi^* = 1/(\phi - 1)$. Then $\mathsf{sdeg}(Q) \leq (17 + 10\phi^*)s - (6 + 20\phi^*)n - 6$, $\mathsf{supp}(Q) \leq (50 + 8\phi(3+\phi) + 40\phi^*)s + 2(-13 + 8\phi(3 + 2\phi) - 40\phi^*)n - 8(1+\phi)^2$, and $\mathsf{size}(Q) \leq \mathsf{size}(P^{\mathsf{w}}) + \mathsf{size}(P^{\mathsf{g}}) \leq 2(7 + 4\phi^*)s - (8 + 16\phi^*)n - 4$. If $\phi = 3$, then $\mathsf{sdeg}(Q) \leq 22s - 16n - 6$, $\mathsf{size}(Q) \leq 18s - 16n - 4$, and $\mathsf{supp}(Q) \leq 214s + 366n - 128$.*

The degree of the circuit checker from [15] is $130s$ and its size is $36s$. Thus, even when $\phi = 3$, we have improved on their construction about 6 times degree-wise and 2 times size-wise. The QSP-SAT witness $(\boldsymbol{a}, \boldsymbol{b})$ can be computed in linear time $\Theta(s)$ by using the algorithm $\mathsf{c2q}$.

8 Two-Query Linear PCP for Circuit-SAT

In Thm. 4, we presented a reduction from Circuit-SAT to QSP-SAT. That is, we showed that if for some $\boldsymbol{w}$, $C(\boldsymbol{w}) = 1$, then one can efficiently construct a witness $(\boldsymbol{a}, \boldsymbol{b}) = \mathsf{c2q}(\boldsymbol{w})$ such that $c_\Lambda(C)(\boldsymbol{a}, \boldsymbol{b}) = 1$. In this section, we construct a two-query non-adaptive linear PCP [2] for Circuit-SAT. In the rest of the paper, we modify this to succinct three-query non-adaptive linear PCP, to a non-adaptive linear interactive proof and finally to a non-adaptive non-interactive zero knowledge argument. Here, non-adaptivity means that the query algorithm (in the linear PCP and linear interactive proof) or the CRS generation algorithm (in the NIZK argument) may depend on the statement C.

Let $\mathcal{R} = \{(C, \boldsymbol{w})\}$ be a binary relation, $\mathbb{F}$ be a finite field, $\mathcal{P}_{\mathsf{lpcp}}$ be a deterministic prover algorithm and $\mathcal{V}_{\mathsf{lpcp}} = (\mathcal{Q}_{\mathsf{lpcp}}, \mathcal{D}_{\mathsf{lpcp}})$, where $\mathcal{Q}_{\mathsf{lpcp}}$ is a probabilistic query algorithm and $\mathcal{D}_{\mathsf{lpcp}}$ is an oracle deterministic decision algorithm. The pair $(\mathcal{P}_{\mathsf{lpcp}}, \mathcal{V}_{\mathsf{lpcp}})$ is a *non-adaptive k-query linear PCP* [2] for $\mathcal{R}$ over $\mathbb{F}$ with query length m if it satisfies the following conditions.

Syntax: on any input C and oracle $\boldsymbol{\pi}$, the verifier $\mathcal{V}_{\mathsf{lpcp}}$ works as follows. $\mathcal{Q}_{\mathsf{lpcp}}(C)$ generates k queries $\boldsymbol{q_1}, \dots, \boldsymbol{q_k} \in \mathbb{F}^m$ to $\boldsymbol{\pi}$, and a state information st. Given k oracle answers $z_1 \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_1} \rangle$, ..., $z_k \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_k} \rangle$, such that $\boldsymbol{z} = (z_1, \dots, z_k)$, $\mathcal{D}^{\boldsymbol{\pi}}_{\mathsf{lpcp}}(\mathsf{st}; \boldsymbol{w}) = \mathcal{D}_{\mathsf{lpcp}}(\mathsf{st}, \boldsymbol{z}; \boldsymbol{w})$ accepts or rejects.

Completeness: for every $(C, \boldsymbol{w}) \in \mathcal{R}$, the output of $\mathcal{P}_{\mathsf{lpcp}}(C, \boldsymbol{w})$ is a description of a linear function $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$ such that $\mathcal{D}^{\boldsymbol{\pi}}_{\mathsf{lpcp}}(\mathsf{st}; \boldsymbol{w})$ accepts with probability 1.

Knowledge: there exists a knowledge extractor $\mathcal{X}_{\mathsf{lpcp}}$, such that for every linear function $\boldsymbol{\pi}^* : \mathbb{F}^m \to \mathbb{F}$: if the probability that $\mathcal{V}^{\boldsymbol{\pi}^*}_{\mathsf{lpcp}}(C)$ accepts is at least ε, then $\mathcal{X}^{\boldsymbol{\pi}^*}_{\mathsf{lpcp}}(C)$ outputs $\boldsymbol{w}$ such that $(C, \boldsymbol{w}) \in \mathcal{R}$.

$(\mathcal{P}_{\mathsf{lpcp}}, \mathcal{V}_{\mathsf{lpcp}})$ has *degree* $(d_\mathcal{Q}, d_\mathcal{D})$, if $\mathcal{Q}_{\mathsf{lpcp}}$ (resp., $\mathcal{D}_{\mathsf{lpcp}}$) can be computed by an arithmetic circuit of degree $d_\mathcal{Q}$ (resp., $d_\mathcal{D}$).

We remark that in the following non-adaptive linear PCP, $\mathcal{D}_{\mathsf{lpcp}}$ does not depend on w.

Theorem 6. *Let $\mathbb{F}$ be a field, and let C be a circuit with dummy gates. Let $\mathcal{P}^{(2)}_{\mathsf{lpcp}}$ and $\mathcal{V}^{(2)}_{\mathsf{lpcp}} = (\mathcal{Q}^{(2)}_{\mathsf{lpcp}}, \mathcal{D}^{(2)}_{\mathsf{lpcp}})$ be as follows:*

$\mathcal{Q}^{(2)}_{\mathsf{lpcp}}(C)$**:** *$Q \leftarrow c_\Lambda(C)$; $m \leftarrow \mathsf{size}(Q)$; $\boldsymbol{q_u} \leftarrow (\boldsymbol{u_i}, \boldsymbol{0}_m)_{i=1}^m$; $\boldsymbol{q_v} \leftarrow (\boldsymbol{0}_m, \boldsymbol{v_i})_{i=1}^m$; $\boldsymbol{q} \leftarrow (\boldsymbol{q_u}, \boldsymbol{q_v})$; $\mathsf{st} \leftarrow (\boldsymbol{u_0}, \boldsymbol{v_0})$; return $(\boldsymbol{q}, \mathsf{st})$;*

$\mathcal{P}^{(2)}_{\mathsf{lpcp}}(C, \boldsymbol{w})$**:** *$Q \leftarrow c_\Lambda(C)$; $(\boldsymbol{\pi_u}, \boldsymbol{\pi_v}) = (\boldsymbol{a}, \boldsymbol{b}) \leftarrow \mathsf{c2q}(\boldsymbol{w})$; return $\boldsymbol{\pi} = (\boldsymbol{\pi_u}, \boldsymbol{\pi_v})$;*

$\mathcal{D}^{(2)}_{\mathsf{lpcp}}(\mathsf{st}, (\boldsymbol{z_u}, \boldsymbol{z_v}); \boldsymbol{w})$**:** *if $(\boldsymbol{z_u} - \boldsymbol{u_0}) \circ (\boldsymbol{z_v} - \boldsymbol{v_0}) = \boldsymbol{0}$ then return 1 else return 0;*

$(\mathcal{P}^{(2)}_{\mathsf{lpcp}}, \mathcal{V}^{(2)}_{\mathsf{lpcp}})$ is a non-adaptive 2-query linear PCP for Circuit-SAT *with query length $2md$ and knowledge error 0.*

Proof. COMPLETENESS: Clearly, $\boldsymbol{z_u} \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_u} \rangle = \sum_{i=1}^m a_i \boldsymbol{u_i}$, $\boldsymbol{z_v} \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_v} \rangle = \sum_{i=1}^m b_i \boldsymbol{v_i}$. Thus, $\boldsymbol{z_u} - \boldsymbol{u_0} = \boldsymbol{a}^\top \cdot U - \boldsymbol{u_0}$ and $\boldsymbol{z_v} - \boldsymbol{v_0} = \boldsymbol{b}^\top \cdot V - \boldsymbol{v_0}$, and the circuit checker accepts.

KNOWLEDGE PROPERTY: Due to the construction of $\mathcal{Q}_{\mathsf{lpcp}}^{(2)}$, $\boldsymbol{z_u} = \sum_{i=1}^m a_i \boldsymbol{u_i}$, and $\boldsymbol{z_v} = \sum_{i=1}^m b_i \boldsymbol{v_i}$. If $\mathcal{D}_{\mathsf{lpcp}}^{(2)}$ accepts, then by Thm. 4, the wire checker implies that no wire η gets a double assignment. However, it may be the case that some wire has no assignment. Nevertheless, on input (st, C) and access to the oracle $\boldsymbol{\pi^*}$, we will now extract a CIRCUIT-SAT witness $\boldsymbol{w} = (w_\eta)_{i=\eta}^{s_e}$ (i.e., the vector of wire values) such that $C(\boldsymbol{w}) = 1$.

First, the extractor obtains the whole linear function $\boldsymbol{\pi^*} = (\boldsymbol{a}, \boldsymbol{b})$, by querying the oracle $\boldsymbol{\pi^*}$ up to $2m$ times. We deduce $\boldsymbol{w}$ from $\boldsymbol{\pi^*}$ as follows.

Let η be any wire of the circuit C. Since the wire checker accepts, the gate checkers of its neighbouring gates do not assign multiple values to the wire η. There are two different cases.

If η is an input wire to the circuit, then its output gate ι is a conscientious dummy gate. Therefore, the value w_η can be extracted from the local values of a_i corresponding to the gate ι.

Assume that η is an internal wire. Since all gates implement functions with well-defined outputs, the gate checker of the input gate of η assigns some value w_η to this wire. Moreover, every output gate ι of η either assigns the same value w_η or does not assign any value. In the latter case, the output value of ι does not depend on w_η, and thus assigning w_η to η is consistent with the output value of ι. Therefore, also here the value w_η can be extracted, but this time from the local values of a_i and b_i corresponding to the input gate of η. □

A simple corollary of this theorem is that the algorithm $\mathsf{c2q}$ is efficiently invertible. Thus, the constructed **NP**-reduction from CIRCUIT-SAT to QSP-SAT preserves knowledge (i.e., it is a Levin reduction).

Note that the communication and computation can be optimized by defining $\boldsymbol{q_u} \leftarrow (\boldsymbol{u_i})_{i=1}^m$, $\boldsymbol{q_v} \leftarrow (\boldsymbol{v_i})_{i=1}^m$, and computing say $\boldsymbol{z_u} \leftarrow \langle \boldsymbol{\pi_u}, \boldsymbol{q_u} \rangle$.

9 Succinct 3-Query Linear PCP from Polynomial QSPs

Since we are interested in succinct arguments, we need to be able to compress the witness vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. As in [15], we will do it by using polynomial interpolation to define polynomial QSPs. We employ the Schwartz-Zippel lemma to show that the resulting succinct 3-query linear PCP has the knowledge property.

9.1 Polynomial Span Programs and QSPs

Instead of considering the target and row vectors of a span program or a QSP as being members of the vector space $\mathbb{F}^d$, interpret them as degree-$(d-1)$ polynomials in $\mathbb{F}[X]$. The map $\boldsymbol{u} \to \hat{u}(X)$ is implemented by choosing d different field elements (that are the same for all vectors $\boldsymbol{u}$) $r_j \leftarrow \mathbb{F}$, and then defining a degree-$(\leq d-1)$ polynomial $\hat{u}(X)$ via polynomial interpolation, so that $\hat{u}(r_j) = u_j$ for all $j \in [d]$. This maps the vectors $\boldsymbol{u_i}$ of the original span program P to polynomials $\hat{u}_i(X)$, and the target vector $\boldsymbol{u_0}$ to the polynomial $\hat{u}_0(X)$. Finally, let $Z(X) := \prod_{j=1}^{d}(X - r_j)$; this polynomial can be thought of as a mapping of the all-zero vector $\mathbf{0} = (0, \dots, 0)$.

The choice of r_j influences efficiency. If r_j are arbitrary, then multipoint evaluation and polynomial interpolation take time $O(d \log^2 d)$ [12]. If d is a power of 2 and $r_j = \omega_d^j$, where ω_d is the dth primitive root of unity, then both operations can be done in time $O(d \log d)$ by using Fast Fourier Transform [12]. In what follows, d and r_j are chosen as in the current paragraph.

Clearly, $\boldsymbol{u_0}$ is in the span of the vectors that belong to $\varrho_{\boldsymbol{w}}^{-1}$ iff $\boldsymbol{u_0} = \sum_{i \in \varrho_{\boldsymbol{w}}^{-1}} a_i \boldsymbol{u_i}$ for some $a_i \in \mathbb{F}$. The latter is equivalent to the requirement that $Z(X)$ divides $\hat{u}(X) := \sum_{i \in \varrho_{\boldsymbol{w}}^{-1}} a_i \hat{u}_i(X) - \hat{u}_0(X)$. Really, $\boldsymbol{u_0}$ is the vector of evaluations of $\hat{u}_0(X)$, and $\boldsymbol{u_i}$ is the vector of evaluations of $\hat{u}_i(X)$. Thus, $\sum a_i \boldsymbol{u_i} - \boldsymbol{u_0} = \mathbf{0}$ iff $\sum a_i \hat{u}_i(X) - \hat{u}_0(X)$ evaluates to 0 at all r_j, and hence is divisible by $Z(X)$.

A *polynomial span program* $P = (\hat{u}_0, U, \varrho)$ over a field $\mathbb{F}$ consists of a target polynomial $\hat{u}_0(X) \in \mathbb{F}[X]$, a tuple $U = (\hat{u}_i(X))_{i=1}^{m}$ of polynomials from $\mathbb{F}[X]$, and a labelling $\varrho : [m] \to \{x_\iota, \bar{x}_\iota : \iota \in [n]\} \cup \{\perp\}$ of the polynomials from U. Let $U_{\boldsymbol{w}}$ be the subset of U consisting of those polynomials whose labels are satisfied by the assignment $\boldsymbol{w} \in \{0,1\}^n$, that is, by $\{x_\iota^{w_\iota} : \iota \in [n]\} \cup \{\perp\}$. The span program P *computes a function* f, if for all $\boldsymbol{w} \in \{0,1\}^n$: there exists $\boldsymbol{a} \in \mathbb{F}^m$ such that $Z(X) \mid (\hat{u}_0(X) + \sum_{u \in U_{\boldsymbol{w}}} a_i \hat{u}(X))$ (P accepts) iff $f(\boldsymbol{w}) = 1$.

Alternatively, P accepts $\boldsymbol{w} \in \{0,1\}^n$ iff there exists a vector $\boldsymbol{a} \in \mathbb{F}^m$, with $a_i = 0$ for all $i \notin \varrho_{\boldsymbol{w}}^{-1}$, s.t. $Z(X) \mid \sum_{i=1}^{m} a_i \hat{u}_i(X) - \hat{u}_0(X)$. The size of P is m and the degree of P is $\deg Z(X)$.

Definition 4. *A* polynomial QSP $Q = (\hat{u}_0, \hat{v}_0, U, V, \varrho)$ *over a field* $\mathbb{F}$ *consists of target polynomials* $\hat{u}_0(X) \in \mathbb{F}[X]$ *and* $\hat{v}_0(X) \in \mathbb{F}[X]$, *two tuples* $U = (\hat{u}_i(X))_{i=1}^{m}$ *and* $V = (\hat{v}_i(X))_{i=1}^{m}$ *of polynomials from* $\mathbb{F}[X]$, *and a labelling* $\varrho : [m] \to \{x_\iota, \bar{x}_\iota : \iota \in [n]\} \cup \{\perp\}$. Q accepts an input $\boldsymbol{w} \in \{0,1\}^n$ *iff there exist two vectors* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *from* $\mathbb{F}^m$, *with* $a_i = 0 = b_i$ *for all* $i \notin \varrho_{\boldsymbol{w}}^{-1}$, *s.t.* $Z(X) \mid (\sum_{i=1}^{m} a_i \hat{u}_i(X) - \hat{u}_0(X))(\sum_{i=1}^{m} b_i \hat{v}_i(X) - \hat{v}_0(X))$. Q computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ *if* Q *accepts* $\boldsymbol{w}$ *iff* $f(\boldsymbol{w}) = 1$.

The size of Q is m and the degree of Q is $\deg Z(X)$. Keeping in mind the reinterpretation of span programs, Def. 4 is clearly equivalent to Def. 1. (Also here, $V = U_{\mathsf{dual}}$, with the dual operation defined appropriately.)

To get from the linear-algebraic interpretation to polynomial interpretation, one has to do the following. Assume that the dimension of the QSP is d and that the size is m. Let $r_j \leftarrow \omega_d^j$, $j \in [d]$. For $i \in [m]$, interpolate the polynomial $\hat{u}_i(X)$ (resp., $\hat{v}_i(X)$) from the values $\hat{u}_i(r_j) = u_{ij}$ (resp., $\hat{v}_i(r_j) = v_{ij}$) for $j \in [d]$. Set $Z(X) := \prod_{j=1}^{d}(X - r_j)$. The labelling ψ is left unchanged. It is clear that the resulting polynomial QSP $(\hat{u}_0, \hat{v}_0, U, V, \psi)$ computes the same Boolean function as the original QSP.

The *polynomial circuit checker* $c_\Lambda^{\text{poly}}(C) = (\hat{u}_0, \hat{v}_0, U, V, \psi)$, with $U = (\hat{u}_0, \dots, \hat{u}_m)$ and $V = (\hat{v}_0, \dots, \hat{v}_m)$, is the polynomial version of $c_\Lambda(C)$.

Theorem 7. *Let $\boldsymbol{w} \in \{0,1\}^n$. $C(\boldsymbol{w}) = 1$ iff $c_{\Lambda}^{\mathrm{poly}}(C)(\mathsf{c2q}(\boldsymbol{w})) = 1$.*

Proof. Follows from Thm. 4 and the construction of polynomial QSPs. □

9.2 Succinct Three-Query Linear PCP

To achieve better efficiency, following [2], we define a 3-query linear PCP with $|\boldsymbol{z}| = \Theta(1)$ that is based on the polynomial QSPs. For a set $\mathcal{P}$ of polynomials, let $\mathsf{span}(\mathcal{P})$ be their span (i.e., the set of $\mathbb{F}$-linear combinations). Then, $\boldsymbol{u}$ is in the span of vectors $\boldsymbol{u_i}$, $\boldsymbol{u} = \sum_{i=1}^{m} a_i \boldsymbol{u_i}$, iff the corresponding interpolated polynomial $\hat{u}(X)$ is in the span of polynomials $\hat{u}_i(X)$, i.e., $\hat{u}(X) = \sum_{i=1}^{m} a_i \hat{u}_i(X)$.

Let $\mathbb{F}$ be any field. We recall that according to the Schwartz-Zippel lemma, for any nonzero polynomial $f : \mathbb{F}^m \to \mathbb{F}$ of total degree d and any finite subset S of $\mathbb{F}$, $\Pr_{\boldsymbol{x} \leftarrow S^m}[f(\boldsymbol{x}) = 0] \le d/|S|$.

Theorem 8. *Let $\mathbb{F}$ be a field, and C a circuit with dummy gates. Let $\mathcal{P}_{\mathsf{lpcp}}^{(3)}$ and $\mathcal{V}_{\mathsf{lpcp}}^{(3)} = (\mathcal{Q}_{\mathsf{lpcp}}^{(3)}, \mathcal{D}_{\mathsf{lpcp}}^{(3)})$ be as follows. Here,* $\mathsf{PolyInt}$ *is polynomial interpolation.*

$\mathcal{Q}_{\mathsf{lpcp}}^{(3)}(C)$: *$Q \leftarrow c_{\Lambda}(C)$; $m \leftarrow \mathsf{size}(Q)$; $d \leftarrow \mathsf{sdeg}(Q)$; For $i \leftarrow 1$ to d do: $r_i \leftarrow \omega_d^i$; $\sigma \leftarrow_r \mathbb{F}$; Compute $(\sigma^i)_{i=0}^{d-1}$; $Z(\sigma) \leftarrow \prod_{j=1}^{d}(\sigma - r_j)$; Compute $(\hat{u}_i(\sigma))_{i=0}^{m}$, $(\hat{v}_i(\sigma))_{i=0}^{m}$; $\mathsf{st} \leftarrow (Z(\sigma), \hat{u}_0(\sigma), \hat{v}_0(\sigma))$; $\boldsymbol{q_u} \leftarrow (((\hat{u}_i(\sigma))_{i=1}^{m}, \boldsymbol{0}_m, \boldsymbol{0}_d)$; $\boldsymbol{q_v} \leftarrow (\boldsymbol{0}_m, (\hat{v}_i(\sigma))_{i=1}^{m}, \boldsymbol{0}_d)$ $\boldsymbol{q_h} \leftarrow (\boldsymbol{0}_m, \boldsymbol{0}_m, (\sigma^i)_{i=0}^{d-1})$; $\boldsymbol{q} \leftarrow (\boldsymbol{q_u}, \boldsymbol{q_v}, \boldsymbol{q_h})$; return $(\boldsymbol{q}, \mathsf{st})$;*

$\mathcal{P}_{\mathsf{lpcp}}^{(3)}(C, \boldsymbol{w})$: *Compute $(Q, m, (r_i)_{i=1}^{d})$ as in $\mathcal{Q}_{\mathsf{lpcp}}^{(3)}(C)$; $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow \mathsf{c2q}(\boldsymbol{w})$; $\boldsymbol{u}^{\dagger} \leftarrow \boldsymbol{u_0} + \sum_{i=1}^{m} a_i \boldsymbol{u_i}$; $\hat{u}^{\dagger}(X) \leftarrow \mathsf{PolyInt}((r_i, u_i^{\dagger})_{i=1}^{d})$; $\boldsymbol{v}^{\dagger} \leftarrow \boldsymbol{v_0} + \sum_{i=1}^{m} a_i \boldsymbol{v_i}$; $\hat{v}^{\dagger}(X) \leftarrow \mathsf{PolyInt}((r_i, v_i^{\dagger})_{i=1}^{d})$; $Z(X) \leftarrow \prod_{i=1}^{d}(X - r_i)$; $\hat{h}(X) = \sum_{i=0}^{d-1} h_i X^i \leftarrow \hat{u}^{\dagger}(X)\hat{v}^{\dagger}(X)/Z(X) \in \mathbb{F}^{d-2}$; return $\boldsymbol{\pi} = (\boldsymbol{\pi_u}, \boldsymbol{\pi_v}, \boldsymbol{\pi_h}) \leftarrow (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{\hat{h}}) \in \mathbb{F}^{2m+d}$;*

$\mathcal{D}_{\mathsf{lpcp}}^{(3)}(\mathsf{st}, (z_u, z_v, z_h); \boldsymbol{w})$: *if $(z_u - \hat{u}_0(\sigma)) \cdot (z_v - \hat{v}_0(\sigma)) = Z(\sigma) \cdot z_h$ then return 1 else return 0;*

$(\mathcal{P}_{\mathsf{lpcp}}^{(3)}, \mathcal{V}_{\mathsf{lpcp}}^{(3)})$ is a non-adaptive 3-query linear PCP over $\mathbb{F}$ for CIRCUIT-SAT *with query length $2m + d$ and knowledge error $2d/|\mathbb{F}|$.*

Proof. COMPLETENESS: again straightforward, since $z_u = \hat{u}_w(\sigma) \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_u} \rangle = \sum_{i=1}^{m} a_i \hat{u}_i(\sigma)$, $z_v = \hat{v}(\sigma) \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_v} \rangle = \sum_{i=1}^{m} b_i \hat{v}_i(\sigma)$, and $z_h = \hat{h}(\sigma) \leftarrow \langle \boldsymbol{\pi}, \boldsymbol{q_h} \rangle = \sum_{i=0}^{d-1} \hat{h}_i \sigma^i$. KNOWLEDGE: assume that the verifier accepts with probability $\varepsilon \ge 2d/|\mathbb{F}|$. That is, $\Pr_{\sigma \leftarrow \mathbb{F}}[(\sum_{i=1}^{m} a_i \hat{u}_i(\sigma) - \hat{u}_0(\sigma))(\sum_{i=1}^{m} a_i \hat{v}_i(\sigma) - \hat{v}_0(\sigma)) = Z(\sigma) \cdot (\sum_{i=0}^{d-1} h_i \sigma^i)] = \varepsilon$. Due to the Schwartz-Zippel lemma, since $\varepsilon \ge 2d/|\mathbb{F}|$, $(\sum_{i=1}^{m} a_i \hat{u}_i(X) - \hat{u}_0(X))(\sum_{i=1}^{m} a_i \hat{v}_i(X) - \hat{v}_0(X)) = Z(X) \cdot (\sum_{i=0}^{d-1} h_i X^i)$, and due to the equivalence between QSPs and polynomial QSPs, Eq. (1) holds. The claim now follows from Thm. 6. □

Theorem 9. *Assume d is a power of 2. $\mathcal{P}_{\mathsf{lpcp}}^{(3)}$ runs in time $\Theta(d \log d)$, $\mathcal{Q}_{\mathsf{lpcp}}^{(3)}$ runs in time $\Theta(d \log d)$, and the time of $\mathcal{D}_{\mathsf{lpcp}}^{(3)}$ is dominated by 2 $\mathbb{F}$-additions and by 2 $\mathbb{F}$-multiplications. $\mathcal{V}_{\mathsf{lpcp}}^{(3)}$ has degree $(d, 2)$.*

A similar result was proven in [15] (though without using the terminology of linear PCPs) in the case of conscientious gate checkers. We only require the dummy gates to be conscientious.

In [15], it was only shown that $\hat{h}(X)$ can be computed by using multipoint evaluation and polynomial interpolation in time $\Theta(d \log^2 d)$. Moreover, the computation of $\mathcal{D}$ was $\Theta(n)$ due to a different extraction technique.

10 From Non-Adaptive Linear PCP to Adaptive NIZK

Given the 3-query linear PCP of Thm. 8, one can use the transformation [2] to construct first a non-adaptive NIZK argument for Circuit-SAT. See the full version. The The non-adaptive NIZK argument can be made adaptive by using universal circuits [25], see [15] for details.

We will provide more details in the full version [20]. There, we will also provide a direct construction of the non-adaptive NIZK argument. The latter has a (quite complex) soundness proof related to the soundness proof from [15] that results in the use of a weaker security assumption. Here, we state only the following straightforward corollary of Thm. 9 and the transformations from [2].

Theorem 10. *Assume d is a power of 2. There exists a non-adaptive NIZK* Circuit-SAT *argument, s.t. the prover and the CRS generation take $\Theta(d \log d)$ cryptographic operations, the verification time is dominated by $\Theta(1)$ pairings, and the communication is a $\Theta(1)$ group elements.*

Acknowledgements. We thank Andris Ambainis, Aleksandrs Belovs, Vitaly Skachek, Hendri Tan, and anonymous reviewers for useful comments and discussions. The author was supported by the Estonian Research Council, and European Union through the European Regional Development Fund.

References

1. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, And Back Again. In: Goldwasser, S. (ed.) ITCS 2012. pp. 326–349. ACM Press
2. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct Non-interactive Arguments via Linear Interactive Proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg
3. Blum, M., Feldman, P., Micali, S.: Non-Interactive Zero-Knowledge and Its Applications. In: STOC 1988. pp. 103–112. ACM Press
4. Brassard, G., Chaum, D., Crépeau, C.: Minimum Disclosure Proofs of Knowledge. Journal of Computer and System Sciences 37(2), 156–189 (1988)
5. Chaabouni, R., Lipmaa, H., Zhang, B.: A Non-Interactive Range Proof with Constant Communication. In: Keromytis, A. (ed.) FC 2012. LNCS, vol. 7397, pp. 179–199. Springer, Heidelberg

6. Di Crescenzo, G., Lipmaa, H.: Succinct NP Proofs from an Extractability Assumption. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) Computability in Europe, CIE 2008. LNCS, vol. 5028, pp. 175–185. Springer, Heidelberg
7. Dodunekov, S., Landgev, I.: On Near-MDS Codes. Journal of Geometry 54(1–2), 30–43 (1995)
8. Dwork, C., Naor, M.: Zaps and Their Applications. In: FOCS 2000. pp. 283–293. IEEE Computer Society Press
9. Elkin, M.: An Improved Construction of Progression-Free Sets. Israel J. of Math. 184, 93–128 (2011)
10. Fauzi, P., Lipmaa, H., Zhang, B.: Efficient Modular NIZK Arguments from Shift and Product. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. ?, pp. ?–? Springer, Heidelberg
11. Gál, A.: A Characterization of Span Program Size and Improved Lower Bounds for Monotone Span Programs. Computational Complexity 10(4), 277–296 (2001)
12. Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, 2 edn. (2003)
13. Gennaro, R., Gentry, C., Parno, B.: Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg
14. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. Tech. Rep. 2012/215, IACR (Apr 19, 2012), available at http://eprint.iacr.org/2012/215, last retrieved version from June 18, 2012
15. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg
16. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg
17. Hoover, H.J., Klawe, M.M., Pippenger, N.: Bounding Fan-out in Logical Networks. Journal of the ACM 31(1), 13–18 (1984)
18. Karchmer, M., Wigderson, A.: On Span Programs. In: Structure in Complexity Theory Conference 1993. pp. 102–111. IEEE Computer Society Press
19. Lipmaa, H.: Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 169–189. Springer, Heidelberg
20. Lipmaa, H.: Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes. Tech. Rep. 2013/121, IACR (Feb 28, 2013), available at http://eprint.iacr.org/2013/121
21. Lipmaa, H., Zhang, B.: A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In: Visconti, I., Prisco, R.D. (eds.) SCN 2012. LNCS, vol. 7485, pp. 477–502. Springer, Heidelberg
22. Micali, S.: CS Proofs. In: Goldwasser, S. (ed.) FOCS 1994. pp. 436–453. IEEE, IEEE Computer Society Press
23. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society
24. Reichardt, B.: Reflections for Quantum Query Algorithms. In: Randall, D. (ed.) SODA 2011. pp. 560–569. SIAM
25. Valiant, L.G.: Universal Circuits (Preliminary Report). In: STOC 1976. pp. 196–203. ACM