

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

First eprint version, February 28, 2013

Helger Lipmaa²

University of Tartu, Estonia

Abstract. Recently, Gennaro, Gentry, Parno and Raykova [GGPR12] proposed an efficient non-interactive zero knowledge argument for Circuit-SAT, based on non-standard notions like conscientious and quadratic span programs. We propose a new non-interactive zero knowledge argument, based on a simple 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). We simplify all steps of the argument. As one of the corollaries, we design an (optimal) wire checker, based on systematic Reed-Solomon codes, of size $8n$ and degree $4n$, while the wire checker from [GGPR12] has size $24n$ and degree $76n$, where n is the circuit size. Importantly, the new argument has constant verifier's computation.

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

1 Introduction

Non-interactive zero knowledge (NIZK, [BFM88]) allows the prover to create a proof such that any verifier can later, without interaction, verify the truth of the intended statement without learning any side information. While NIZK proofs are important in many cryptographic applications like e-voting or verifiable computation, there are only a few different generic methodologies to construct efficient NIZK proofs. Most famously, Groth and Sahai [GS08] proposed NIZK proofs for a class of practically relevant languages. Their proofs have constant common reference string (CRS) length, and linear computational and communication complexity. However, since a single proof might get transferred and verified many times, one often requires better communication and verifier's computation.

Groth [Gro10] proposed the first NIZK argument (computationally-sound proof) for an **NP**-complete language with sublinear communication. Groth's construction was improved by Lipmaa [Lip12]. Groth and Lipmaa rewrote the Circuit-SAT argument in a parallel programming language that consists of primitive arguments (Hadamard sum, Hadamard product and permutation), and then constructed efficient arguments for the latter. The Circuit-SAT arguments of [Gro10, Lip12] have constant communication, quadratic prover's computation, and linear verifier's computation in n (the circuit size). In [Gro10], the CRS length is $\Theta(n^2)$. In [Lip12], the CRS length is $\Theta(r_3^{-1}(n)) = o(n2^{2\sqrt{2\log_2 n}})$, where $r_3(N) = \Omega(N \log^{1/4} N / 2^{2\sqrt{2\log_2 N}})$ [Elk11] is the cardinality of the largest progression-free subset of $[N]$. The arguments of Groth and Lipmaa are not applicable (unless n is really small) because of the quadratic prover's computation. Fauzi, Lipmaa and Zhang [FLZ12] constructed arguments for the **NP**-complete languages subset sum and decision knapsack with the CRS length $\Theta(r_3^{-1}(n))$ and subquadratic prover's computation $\Theta(r_3^{-1}(n) \log n)$. However, they did not propose a similar argument for the Circuit-SAT.

Gennaro, Gentry, Parno and Raykova [GGPR12] constructed a new NIZK Circuit-SAT argument, based on efficient (quadratic) span programs¹. Their non-adaptively sound NIZK argument has a linear CRS length, $\Theta(n \log^2 n)$ prover's computation, and linear-in-input size verifier's computation. The argument can be made

¹ We refer to Sect. 2 for standard preliminaries on span programs, and will assume during the rest of this introduction that the reader has basic familiarity with span programs. We note that [GGPR12] has been accepted for publication as [GGPR13].

adaptively sound by using universal circuits, [Val76], and the adaptively sound argument has CRS length $\Theta(n \log n)$, prover’s computation $\Theta(n \log^3 n)$ and verifier’s computation $\Theta(n)$.

Briefly, [GGPR12] first 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 a Circuit-SAT argument, 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 every individual gate’s operation in parallel. Second, [GGPR12] constructs 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 guarantees consistency only when all gate checkers are conscientious.) They define quadratic span programs, construct a quadratic span program that implements both the aggregate gate checker and the weak wire checker, and then construct an efficient NIZK argument that verifies (given a vector commitment to all coefficients) the quadratic span program.

Our Contributions. We improve the construction of [GGPR12] in several aspects. Some of our improvements are conceptual (e.g., we provide more clear definitions which result in better constructions) and some of the improvements are technical (with special emphasis on concrete efficiency). We outline our construction below, and briefly sketch the differences compared to [GGPR12].

To verify whether the circuit C accepts an input, we first use a constant-size *standard* (i.e., not necessary conscientious) span program to verify every gate separately. Then, by using the standard “AND composition” of span programs [Amb10], we construct a single large span program that verifies the computation of every gate in parallel.

Unfortunately, simple AND composition of the gate checkers is not secure, because it allows “double-assignments”. More precisely, there will be vectors from different adjacent gate checkers that correspond to the variable corresponding to the same wire. While every individual checker might be locally correct, one checker could work with value 0 assigned to this wire while another checker could work with value 1 assigned to the same wire. Clearly, such bad cases should be detected.

We solve this issue as follows. Let `Code` be an arbitrary high-distance linear $[N, K, D]$ error-correcting code that satisfies $D > N/2$. For a concrete 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 \mathbf{v}_i) are labelled by the positive literal x_η and some (say \mathbf{w}_i) by the negative literal \bar{x}_η . The individual gate checkers’s acceptance “fixes” certain coefficients a_i (that are used with \mathbf{v}_i) and b_i (that are used with \mathbf{w}_i) for all adjacent gate checkers. Roughly stating, for consistency one requires that either all values a_i are zero (then unequivocally $x_\eta = 0$), or all values b_i are zero (then unequivocally $x_\eta = 1$). We verify that this is the case by applying an efficient high-distance linear error-correcting code separately to the vectors \mathbf{a} and \mathbf{b} . The high-distance property of the linear-error correcting code guarantees that if \mathbf{a} and \mathbf{b} are not consistent, then there exists a coefficient i such that $\text{Code}(\mathbf{a})_i \cdot \text{Code}(\mathbf{b})_i \neq 0$. We use the systematic Reed-Solomon code [RS60], since it is a maximum distance separable (MDS) code with optimal support (that is, it has the minimal possible number of non-zero elements in its generating matrix).

Motivated by this construction, we redefine quadratic span programs [GGPR12] as follows. A quadratic span program — that consists of two target vectors \mathbf{t}_v and \mathbf{t}_w and two matrices \mathcal{V} and \mathcal{W} — accepts an input only if for some vectors \mathbf{a} and \mathbf{b} that are consistent with this input, $(\mathcal{V} \cdot \mathbf{a} - \mathbf{t}_v) \circ (\mathcal{W} \cdot \mathbf{b} - \mathbf{t}_w) = \mathbf{0}$. Here, \circ denotes the pointwise (Hadamard) product of two vectors. Clearly, the above linear error-correcting code based construction implements a quadratic span program, with \mathcal{V} and \mathcal{W} basically being the generating matrices of the code. (No connection to error-correcting codes was made in [GGPR12].) We also construct an aggregate wire checker by applying an AND composition rule to the individual wire checkers, and then construct a single quadratic span program (the circuit checker) that implements both the aggregate gate checker and the aggregate wire checker.

To summarize, the new circuit checker consists of two elements. First, an aggregate gate checker (a standard span program) that verifies that every individual gate is executed correctly on their local variables. Second, an aggregate wire checker (a quadratic span program, based on a high-distance linear error-correcting code) which verifies that individual gates are executed on the consistent assignments to the variables. Im-

portantly, the circuit checker is a composition of small (quadratic) span programs, and in total has only a constant number of non-zero elements per vector. This means that the final NIZK argument will have linear computational complexity (in the size of the universal circuit).

To construct an efficient NIZK argument, we need several extra steps that are similar to those taken by Gennaro et al. As in [GGPR12], we define polynomial span programs and polynomial quadratic span programs. Differently from [GGPR12] (that only gave the polynomial definition), our main definition of quadratic span programs is similar to the common definition of span programs, and we then use a transformation to get an arbitrary quadratic span program to a “polynomial” form. We feel the non-polynomial definition is much more natural, and helps to describe the essence of the construction better.

By using techniques related to those from [GGPR12], we construct a NIZK Circuit-SAT argument. The main difference in this part of the paper is in the security proof. The soundness of the argument from [GGPR12] is only proved in the non-adaptive case. It is then claimed in [GGPR12] that one can make the construction adaptive by using universal circuits, but this is not proven. We start by assuming that we work with the universal circuit [Val76], and that the corresponding quadratic span program was fixed while the CRS was generated. This allows us to achieve adaptive soundness. We also use an improved and more elaborate (non-deterministic) extraction technique which, differently from the one from [GGPR12], also works with non-conscientious gate checkers. While the technique of [GGPR12] clearly distinguished inputs of the circuit from intermediate values, in our case all wire values are handled similarly. Importantly, this allows us to achieve constant verifier’s computation.

More Details. Gennaro et al [GGPR12] defined a weak wire checker that guarantees consistency only when the gate checkers were conscientious. This also means that their NIZK argument is sound only if all gate checkers are conscientious. The new wire checker does not require the gate checkers to be conscientious. This, in turn, not only enables us to construct much more efficient gate checkers but also (potentially) enables one to use standard techniques (e.g., the combinatorial characterization of span program size [Gál01], or semidefinite programming [Rei11]) to construct more efficient checkers for larger unit computations. In addition, the new wire checker by itself is more efficient than the weak wire checker from [GGPR12]. We prove that in a certain well-defined sense, the new wire checker is optimal both in its size and its support (number of non-zero elements).

We construct several (optimally) efficient span programs for gate checkers, needed to construct the Circuit-SAT argument. In particular, we construct a size 6 and dimension 3 NAND checker (this can be compared to size 12 and dimension 9 conscientious NAND checker from [GGPR12]).

As a minor contribution, by using a classical result by Hoover, Klawe and Pippenger [HKP84] about constructing low fan-out circuits, we are able to more precisely quantify the size and other parameters, especially support, of the aggregate gate and wire checker.

We also rephrase certain proof techniques from [GGPR12] in the language of multilinear universal hash functions [GMS74,CW79,WC81]. This might be an interesting contribution by itself. Apart from a more clear proof, this results in a slightly weaker security assumption.

Finally, we note that by using efficient polynomial algebra [GG03], one can reduce the prover’s computation of the new argument from $\Theta(d \log^2 d)$ (as in [GGPR12]) to $\Theta(d \log d)$, where $d = \Theta(n \log n)$ is the degree of the circuit checker. The same optimization applies to the argument of [GGPR12]. We note that when using the Erdős-Turán progression-free set from [ET36] the subset sum argument of [FLZ12] requires prover’s computational complexity $\Theta(n^{\log_2 3} \cdot \log n)$ which, despite of fast asymptotic growth, is smaller than $\Theta(n \log^3 n)$ for $n \leq 10\,000$, at which point arguments from both [FLZ12] and [GGPR12] are computationally infeasible. On the other hand, $n \log^2 n$ (prover’s computational complexity after the mentioned optimization) is smaller than $n^{\log_2 3} \cdot \log n$ already for very small values of n .

Efficiency. The new Circuit-SAT argument has the same asymptotic CRS length ($\Theta(n \log n)$), prover’s computational complexity ($\Theta(n \log^2 n)$, after applying the optimization from the previous paragraph) and communication complexity (constant) as the adaptively sound variant of the argument from [GGPR12]. However, in the first two cases the constant inside Θ has decreased significantly. Importantly, due to the better extracting technique, we achieve constant (as opposed to $\Theta(n)$ in the adaptively sound variant of [GGPR12]) verifier’s computational complexity. We emphasize that all additional optimization techniques applicable to

the argument from [GGPR12] (e.g., the use of techniques from [BCCT13]) are also applicable to the new argument.

Other Applications. We do hope that by using our techniques, one can construct efficient NIZK arguments for other languages. As an example, the techniques of [Lip12] were used in [CLZ12] to construct an efficient range argument, and in [LZ12] to construct an efficient shuffle. Quadratic span programs have more applications than just in the NIZK construction (or more generally, in the construction of the wire checker). We only mention that one can construct a related zap [DN00], a related (public or designated-verifier) succinct non-interactive argument of knowledge (SNARK, see [Mic94,DL08,GW11,BCCT12]) by using the techniques of [BCCT12,GGPR12], and implement verifiable computation [GGP10]. In fact, applying our techniques to verifiable computation is extremely natural: instead of gates, one can talk about small (but possibly much larger than gates) computational units, and instead of wires, about the values transferred between the small computational units. Since here one deals with much larger span programs than in the case of the Circuit-SAT argument, it is especially beneficial that one can use standard (non-conscientious) span programs.

We leave it as an open question whether the non-cryptographic part of the new construction (splitting the verification of a computation into small steps and then using high-distance linear error-correcting codes to verify the consistency of individual steps) has some non-cryptographic applications.

2 Preliminaries: Span Programs

We assume that \mathbb{F} is 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 \mathbf{v} denote row vectors. For a matrix \mathcal{V} , let \mathbf{v}_i be its i th row vector. For an $m \times d$ matrix \mathcal{V} over \mathbb{F} , let $\text{span } \mathcal{V} := \{\sum_{i=1}^m a_i \mathbf{v}_i : \mathbf{a} \in \mathbb{F}^m\}$. Let x_i be formal variables. We denote the positive literals x_i by x_i^1 and the negative literals \bar{x}_i by x_i^0 .

A *span program* [KW93] $P = (\mathbf{t}, \mathcal{V}, \varrho)$ over a field \mathbb{F} consists of a non-zero target vector $\mathbf{t} \in \mathbb{F}^d$, an $m \times d$ matrix \mathcal{V} over \mathbb{F} , and a labelling $\varrho : [m] \rightarrow \{x_\iota, \bar{x}_\iota : \iota \in [n_0]\} \cup \{\perp\}$ of \mathcal{V} 's rows by one of $2n$ literals or by \perp . Let $\mathcal{V}_{\mathbf{u}}$ be the submatrix of \mathcal{V} consisting of those rows whose labels are satisfied by the assignment \mathbf{u} , that is, by $\{x_\iota^{u_\iota} : \iota \in [n_0]\} \cup \{\perp\}$. The span program computes a function f , if for all $\mathbf{u} \in \{0, 1\}^{n_0}$: $\mathbf{t} \in \text{span } \mathcal{V}_{\mathbf{u}}$ if and only if $f(\mathbf{u}) = 1$.

We define $\varrho_{\mathbf{u}}^{-1} = \{i \in [m] : \varrho(i) \in \{x_\iota^{u_\iota} : \iota \in [n_0]\} \cup \{\perp\}\}$ to be the set of rows whose labels are satisfied by the assignment \mathbf{u} . The size, $\text{size } P$, of the span program is m . The dimension $\text{sdim } P$ is equal to d . We say that the span program P has *support* $\text{supp } P$, if all vectors $\mathbf{v} \in \mathcal{V}$ have altogether $\text{supp } P$ non-zero elements. Clearly, \mathbf{t} 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. Since linear algebra can be implemented in log-space uniform-NC2 [BGP95,BW03], polynomial-sized span programs can implement only languages in the complexity class NC2.

Let $D(x_\iota) := \max_{j \in \{0, 1\}} |\varrho^{-1}(x_\iota^j)|$, for each $\iota \in [n_0]$, be the maximum number of vectors that have the same label (ι, j) with $j \in \{0, 1\}$. This parameter is needed later when we construct wire checkers.

Span programs were defined in [KW93], originally to help proving various lower bounds (see, for example, [Gál01]). Later, they have been used to design quantum algorithms [RS08] (see also [Rei11,Bel12b,Bel12a], or the survey [Amb10]), linear secret sharing schemes (as already shown in [KW93], see for example [CF02]), and non-interactive zero knowledge (NIZK) arguments [GGPR12]. See [Juk12] for a general exposition of span programs.

One commonly constructs more complex span programs by using simple span programs and their composition rules, see, e.g., [Amb10]. Span programs for AND, OR, XOR, and equality of two variables x and y are as follows:

$$SP(\wedge) : \left(\begin{array}{c|cc} & 1 & 1 \\ \hline x & 1 & 0 \\ y & 0 & 1 \end{array} \right), \quad SP(\vee) : \left(\begin{array}{c|c} & 1 \\ \hline x & 1 \\ y & 1 \end{array} \right), \quad SP(\oplus) : \left(\begin{array}{c|cc} & 0 & 1 \\ \hline x & 1 & 1 \\ y & 1 & 1 \\ \bar{x} & 0 & -1 \\ \bar{y} & 0 & -1 \end{array} \right), \quad SP(=) : \left(\begin{array}{c|cc} & 0 & 1 \\ \hline x & 1 & 1 \\ y & -1 & 0 \\ \bar{x} & -1 & 0 \\ \bar{y} & 1 & 1 \end{array} \right).$$

Given span programs $P_0 = SP(f_0)$ and $P_1 = SP(f_1)$ for functions f_0 and f_1 , it is well known how to build span programs for $SP(f_0 \wedge f_1)$ and $SP(f_0 \vee f_1)$. Both compositions assume that the target vector is in a specific form — $\mathbf{t} = (1, \dots, 1)$ in the first case and $\mathbf{t} = (0, \dots, 0, 1)$ in the second case. Thus, in a circuit that consists of both AND and OR gates, one has to implement a basis change to transform \mathbf{t} to the correct form.

Assume that the size m_i and dimension d_i span program $SP(f_i)$ has the target vector $\mathbf{t}_i = (1, \dots, 1)$, with the j th row vector \mathbf{v}_{ij} being labelled by x_{ij} . The span program $SP(f_0 \wedge f_1)$ has size $m_0 + m_1$ and dimension $d_0 + d_1$. In $SP(f_0 \wedge f_1)$, \mathbf{t} is a concatenation of \mathbf{t}_0 and \mathbf{t}_1 , the first d_0 vectors are equal to $(\mathbf{v}_{0j}, \mathbf{0}_{d_0})$ (and labelled by x_{0j}), and the last d_1 vectors are equal to $(\mathbf{0}_{d_1}, \mathbf{v}_{1j})$ (and labelled by x_{1j}). The following span program for $P = SP(f_0 \vee f_1)$ has size $m_0 + m_1$ and dimension $d_0 + d_1 - 1$. Write the vectors of $P_i = SP(f_i)$ as $\mathbf{v} = (\mathbf{v}_{-d_i}, v_{d_i})$, where v_{d_i} is their last coordinate. Let the target vector of P_i be $(\mathbf{0}_{d_i-1}, 1)$. The target vector of P is $(\mathbf{0}_{d_0+d_1-2}, 1)$. For each \mathbf{v}_i from P_0 , we add the vector $(\mathbf{v}_i, -\mathbf{0}, v_{id_0})$ to P . For each \mathbf{v}_i from P_1 , we include the vector $(\mathbf{0}, \mathbf{v}_i, -d_1, v_{id_1})$.

3 Efficient Gate Checkers

A *gate checker* for a gate function $f : \{0, 1\}^{n_0} \rightarrow \{0, 1\}^{n_1}$ is a function $c_f : \{0, 1\}^{n_0+n_1} \rightarrow \{0, 1\}$, such that $c_f(\mathbf{x}, \mathbf{y}) = 1$ iff $f(\mathbf{x}) = \mathbf{y}$. We are mainly interested in unary and binary Boolean functions f .

The Boolean function NAND $\bar{\wedge}$ is defined as $\bar{\wedge}(x, y) = x \bar{\wedge} y = \neg(x \wedge y)$. The NAND-checker $c_{\bar{\wedge}} : \{0, 1\}^3 \rightarrow \{0, 1\}$ outputs 1 iff $z = x \bar{\wedge} y$. We now propose an efficient span program for $c_{\bar{\wedge}}$.

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

Proof. We obtained $SP(c_{\bar{\wedge}})$ by using simple AND and OR compositions from the observation that $c_{\bar{\wedge}}(x, \bar{x}, y, \bar{y}, z, \bar{z}) = (x \vee z) \wedge (y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$. One can use a simple case analysis to see that $SP(c_{\bar{\wedge}})$ computes $c_{\bar{\wedge}}$:

- $a_1 = a_2 = a_3 = 0$ (i.e., $x = y = z = 0$) does not give a solution,
- $a_1 = a_2 = a_6 = 0$ (i.e., $x = y = 0$ and $z = 1$) gives a solution with $a_3 = 1$, $a_4 \in \mathbb{Z}_q$, and $a_5 = 1 - a_4$,
- $a_1 = a_5 = a_3 = 0$ (i.e., $x = z = 0$ and $y = 1$) does not give a solution,
- $a_1 = a_5 = a_6 = 0$ (i.e., $x = 0$ and $y = z = 1$) gives a solution with $a_3 = 1$, $a_2 = -2$, $a_4 = 1$,
- $a_4 = a_2 = a_3 = 0$ (i.e., $x = 1$ and $y = z = 0$) does not give a solution,
- $a_4 = a_2 = a_6 = 0$ (i.e., $x = z = 1$ and $y = 0$) gives a solution with $a_1 = -2$, $a_3 = 1$, $a_5 = 1$,
- $a_4 = a_5 = a_3 = 0$ (i.e., $x = y = 1$ and $z = 0$) gives a solution with $a_1 = 1$, $a_2 = 1$, $a_5 = 1$,
- $a_4 = a_5 = a_6 = 0$ (i.e., $x = y = z = 1$) does not give a solution.

The claim about the size, dimension, and support is straightforward. □

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 $\sum a_i \mathbf{v}_i = \mathbf{t}$. However, a satisfying input to $SP(c_{\bar{\wedge}})$ does not fix the values a_i unequivocally. Namely, 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 $a_4 = 0$ (and $a_5 = 1$), $SP(c_{\bar{\wedge}})$ is not conscientious.

Given $SP(c_{\bar{\wedge}})$, one can construct a size 6 and dimension 3 span program for the AND-checker function $c_{\wedge}(x, y, z) := (x \wedge y) \oplus \bar{z}$ by interchanging the rows labelled by z and \bar{z} in $SP(c_{\bar{\wedge}})$. Similarly, one can construct a size 6 and dimension 3 span program for the OR-checker function $c_{\vee}(x, y, z) := (\bar{x} \wedge \bar{y}) \oplus z$ by interchanging the rows labelled by x and \bar{x} , and the rows labelled by y and \bar{y} , in $SP(c_{\bar{\wedge}})$.

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. (See Sect. 2.)

$$\begin{pmatrix} \mathbf{t} & 1 & 1 & 1 & 1 \\ x & 1 & 0 & 0 & 0 \\ y & 0 & 1 & 0 & 0 \\ z & 1 & 1 & 0 & 0 \\ \bar{x} & 0 & 0 & 1 & 0 \\ \bar{y} & 0 & 0 & 1 & 0 \\ \bar{z} & 0 & 0 & 1 & 0 \end{pmatrix}$$

Fig. 1. $SP(c_{\bar{\wedge}})$

We need a fork gate that computes $y_1 \leftarrow x, y_2 \leftarrow x$. That is, $c_Y(x, y_1, y_2) = (x \wedge y_1 \wedge y_2) \vee (\bar{x} \wedge \bar{y}_1 \wedge \bar{y}_2)$. We write c_Y in the CNF form, $c_Y(x, y_1, y_2) = (\bar{x} \vee y_2) \wedge (x \vee \bar{y}_1) \wedge (y_1 \vee \bar{y}_2)$. Since every literal is mentioned only once in the CNF, we can use AND and OR compositions to derive the span program on Fig. 2. Thus, c_Y has a span program of size 6, dimension 3, and support 6.

We also need a 1-to- t fork-checker which has 1 input x and t outputs y_ι , with $y_\iota = x$. The t -fork checker is then $c_Y^t(x, \mathbf{y}) = (x \wedge y_1 \wedge \dots \wedge y_t) \vee (\bar{x} \wedge \bar{y}_1 \wedge \dots \wedge \bar{y}_t)$. It is easy to see that c_Y^t has a CNF $c_Y^t(x, \mathbf{y}) = (x \vee \bar{y}_1) \wedge (y_1 \vee \bar{y}_2) \wedge \dots \wedge (y_{t-1} \vee \bar{y}_t) \wedge (y_t \vee \bar{x})$. From this we construct a span program exactly as in the case $t = 2$. The span program has size $2(t+1)$ and dimension $t+1$. 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, we note that $SP^*(c_Y^t)$ has two 1-entries in every column, and one in every row. Thus, it has support $\text{supp}(SP(c_Y^t)) = \sum_{i=1}^{t+1} 2 = 2(t+1) = 2t+2$.

$$\begin{pmatrix} t & 1 & 1 & 1 \\ x & 0 & 1 & 0 \\ y_1 & 0 & 0 & 1 \\ y_2 & 1 & 0 & 0 \\ \bar{x} & 1 & 0 & 0 \\ \bar{y}_1 & 0 & 1 & 0 \\ \bar{y}_2 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 2. $SP(c_Y)$

4 Aggregate Gate Checker

Given a circuit that consists only of NAND, AND, OR, XOR, and NOT gates, we combine the individual gate checkers by using the span program AND composition rule from Sect. 2. In addition, to make the wire checker of Sect. 6.1 (and thus also the final NIZK argument) more efficient, all gates of the circuit C need to have a small fan-out. In [GGPR12], the authors designed a circuit of size $3 \cdot |C|$ that implements the functionality of the circuit C but only has fan-out 2 except for a specially introduced dummy input. The GGPR aggregate gate checker has size $36 \cdot |C|$ and dimension $27 \cdot |C|$. By using the techniques of [HKP84] (that replace every high fan-out gate with an inverse binary tree of fork gates, and then gives a precise estimation of the resulting circuit size), we prove a much more precise result. Differently from [GGPR12], we also do not introduce dummy gates at every input, or the dummy input.

Let C be a circuit. For a gate i of C , let $\text{deg}^+(i)$ be its fan-out, and let $\text{deg}^-(i)$ be its fan-in. Let $\text{deg}(i) = \text{deg}^-(i) + \text{deg}^+(i)$. The *aggregate gate checker function* agc of a circuit C is a function $\text{agc} : \{0, 1\}^{\sum_{i=1}^{|C|} \text{deg}(i)} \rightarrow \{0, 1\}^{|C|}$. If c_i is the gate checker of the i th gate and $\dim \mathbf{x}_i = \text{deg}(i)$, then $\text{agc}(\mathbf{x}_1, \dots, \mathbf{x}_{|C|}) = (c_1(\mathbf{x}_1), \dots, c_{|C|}(\mathbf{x}_{|C|}))$.

Theorem 1. *Let $f : \{0, 1\}^{n_0} \rightarrow \{0, 1\}$ be a function implemented by a fan-in ≤ 2 circuit C with $n = |C|$ NAND, AND, OR, XOR, and NOT gates. There exists a fan-in ≤ 2 and fan-out $\leq t$ circuit C_{bnd} for f which has the same n gates as C and up to $(n - 2n_0)/(t - 1)$ additional t -fork gates. Denote $t^* := 1/(t - 1)$. The aggregate gate checker $\text{agc} = \text{agc}(C_{\text{bnd}})$ for f has a span program P with size $P \leq (8 + 4t^*)n - (10 + 8t^*)n_0$, $\text{sdim } P \leq (4 + 2t^*)n - (5 + 4t^*)n_0$, and $\text{supp } P \leq (9 + 4t^*)n - (11 + 8t^*)n_0$. If $t = 3$, then size $P \leq 10n - 14n_0$, $\text{sdim } P \leq 5n - 7n_0$, and $\text{supp } P \leq 11n - 15n_0$.*

The proof of this theorem is given in App. B.

We emphasize that the optimal choice of t depends on the parameter that we are going to optimize. The aggregate gate checker has optimal size, dimension and support when t 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. 6.2) is minimized when $t = 2$. To somewhat balance the parameters, we concentrate on the case $t = 3$.

5 Quadratic Span Programs

A quadratic span program is an extension of span programs, motivated by what one can actually do by using bilinear maps. We will first give a definition of quadratic span programs by using the language of linear algebra. After that, in Sect. 8, we will provide a polynomial redefinition of quadratic span programs and show that the result is equivalent to the definition given in [GGPR12].

Definition 1. *A quadratic span program $P = (\mathbf{t}_v, \mathbf{t}_w, \mathcal{V}, \mathcal{W}, \varrho)$ over a field \mathbb{F} consists of two (possibly all-zero) target vectors $\mathbf{t}_v, \mathbf{t}_w \in \mathbb{F}^d$, two $m \times d$ matrices \mathcal{V} and \mathcal{W} , and a common labelling $\varrho : [m] \rightarrow \{x_i, \bar{x}_i : i \in$*

$[n_0] \cup \{\perp\}$ of the rows of \mathcal{V} and \mathcal{W} . P accepts an input $\mathbf{u} \in \{0, 1\}^{n_0}$ iff there exist two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^m$, with $a_i = 0 = b_i$ for all $i \notin \varrho_{\mathbf{u}}^{-1}$, such that $(\mathcal{V} \cdot \mathbf{a} - \mathbf{t}_v) \circ (\mathcal{W} \cdot \mathbf{b} - \mathbf{t}_w) = \mathbf{0}$, where $\mathbf{x} \circ \mathbf{y}$ denotes the pointwise (Hadamard) product of \mathbf{x} and \mathbf{y} . The quadratic span program computes a function f if for all $\mathbf{u} \in \{0, 1\}^{n_0}$: $f(\mathbf{u}) = 1$ iff P accepts \mathbf{u} .

The size, size P , of the quadratic span program is m . The dimension $\text{sdim } P$ is equal to d . The support $\text{supp } P$ of a quadratic span program P is equal to the sum of the supports (that is, the number of non-zero elements) of all vectors \mathbf{v}_i and \mathbf{w}_i .

Clearly, $(\mathcal{V} \cdot \mathbf{a} - \mathbf{t}_v) \circ (\mathcal{W} \cdot \mathbf{b} - \mathbf{t}_w) = \mathbf{0}$ is equivalent to the requirement that for all $j \in [d]$, $(\sum_{i=1}^m a_i v_{ij} - t_{vj}) (\sum_{i=1}^m b_i w_{ij} - t_{wj}) = 0$. Since \mathbb{F} is an integral domain, this is equivalent to the requirement that for all $j \in [d]$, either $\sum_{i=1}^m a_i v_{ij} = t_{vj}$ or $\sum_{i=1}^m b_i w_{ij} = t_{wj}$, 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 only if for all $j \in [d]$, both $\sum_{i=1}^m a_i v_{ij} = t_{vj}$ and $\sum_{i=1}^m b_i w_{ij} = t_{wj}$. This AND composition accepts exactly if two span programs accept simultaneously, that is, $\sum a_i \mathbf{v}_i = \mathbf{t}_v$ and $\sum b_i \mathbf{w}_i = \mathbf{t}_w$. On the other hand, one cannot implement an element-wise OR composition (quadratic span program) as a span program. Quadratic span programs add an element-wise OR to an element-wise AND, and thus it is not surprising that they increase the expressiveness of span programs.

Clearly, one can compose quadratic span programs by using the AND and OR composition rules of span programs. One has to take care to apply the same transformation to both \mathcal{V} and \mathcal{W} simultaneously.

6 Wire Checker and Aggregate Wire Checker

6.1 Wire Checker

Gate checkers verify that every individual gate is followed correctly, that is, that its output wire obtains a value which is consistent with its input wires. On top of that, one also requires intra-gate (wire) consistency that ensures that adjacent gate checkers do not make double assignments to any of the wires. Following [GGPR12], we construct a wire checker to verify such intra-gate consistency. 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 AND composition of quadratic span programs.

We will need the following notation. Let $G = (V, E) = G(C)$ be the hypergraph of the circuit 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 t adjacent vertices) has $t + 1$ adjacent vertices, where t 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). Clearly, $|E(G)| \leq 2(|V(G)| - n_0)$, where n_0 is the number of inputs to the circuit, e.g., the number of the sources of G . We denote the set of gates of C by $V(C)$ and the set of wires of C by $E(C)$.

Every wire $\eta \in E(C)$ corresponds to a formal variable x_η in a natural way. This variable obtains an assignment, computed from the input assignment \mathbf{u} . Let $N(\eta)$ be the set of η 's adjacent gates. For every $i \in N(\eta)$, let $P_i = (\mathbf{t}_i, \mathcal{V}_i, \varrho_i)$ be the corresponding gate checker. For every $i \in N(\eta)$, one of the input or output variables of P_i (that we denote by $x_{i:\eta}$) corresponds to x_η . Recall that for a local variable y of a span program P_i , $D(y) = \max(|\varrho^{-1}(y)|, |\varrho^{-1}(\bar{y})|)$. We assume that $|\varrho^{-1}(y)| = |\varrho^{-1}(\bar{y})|$, by adding zero vectors to the span programs if necessary. Let $D(\eta) := \sum_{i \in N(\eta)} D(x_{i:\eta})$.

We define the η th wire checker between the rows of adjacent gates $i \in N(\eta)$ that are labelled either by the local variable $x_{i:\eta}$ or its negation $\bar{x}_{i:\eta}$, i.e., between the rows $\{i : \exists k \in N(\eta) \text{ s.t. } \varrho_k(i) = x_{k:\eta} \vee \varrho_k(i) = \bar{x}_{k:\eta}\}$, where ϱ_k is defined as in the previous paragraph. Let ψ be the natural labelling of the wire checkers, with $\psi(i) = x_\eta^j$ iff $\varrho_k(i) = x_{k:\eta}^j$ for some $k \in N(\eta)$. After possible re-enumerating of rows, assume that $[D_{cum} + 1, D_{cum} + D(\eta)]$ are ψ -labelled by \bar{x}_η and $[D_{cum} + D(\eta) + 1, D_{cum} + 2D(\eta)]$ are ψ -labelled by x_η , where $D_{cum} = 2 \sum_{\eta^* = 1}^{\eta-1} D(\eta^*)$.

We first give a definition of wire checkers for the case where there is only one wire η and thus only one variable x_η . In Sect. 6.2, we will give a definition and a construction for the aggregate case.

For $\mathbf{x} = (x_1, \dots, x_{2D})$, define $\mathbf{x}^{(1)} := (x_1, \dots, x_D)^\top$ and $\mathbf{x}^{(2)} := (x_{D+1}, \dots, x_{2D})^\top$. Fix a wire η , and assume that $D = D(\eta)$, $\psi^{-1}(\bar{x}_\eta) = [1, D]$ and $\psi^{-1}(x_\eta) = [D+1, 2D]$. Let $m = 2D$. Let $Q = (\mathbf{t}_v, \mathbf{t}_w, \mathcal{V}, \mathcal{W}, \psi)$, with two $m \times d$ matrices \mathcal{V} and \mathcal{W} , be a quadratic span program. Q is a *wire checker*, if for any tuples $\mathbf{a}, \mathbf{b} \in \mathbb{F}^{2D}$, $(\mathcal{V} \cdot \mathbf{a} - \mathbf{t}_v) \circ (\mathcal{W} \cdot \mathbf{b} - \mathbf{t}_w) = \mathbf{0}$ iff \mathbf{a} and \mathbf{b} indicate consistent bit assignments in the following sense: either $\mathbf{a}^{(1)} = \mathbf{0}$ or $\mathbf{b}^{(1)} = \mathbf{0}$, and either $\mathbf{a}^{(2)} = \mathbf{0}$ or $\mathbf{b}^{(2)} = \mathbf{0}$.

We propose a new wire checker which is based on the properties of high-distance linear error-correcting codes. (See App. A for background on coding theory.) To obtain optimal efficiency, we choose particular codes (namely, systematic Reed-Solomon codes [RS60]). Let RS_D be the $D \times D^*$ generator matrix of the $[D^* = 2D - 1, D, D]_q$ systematic Reed-Solomon code.

Definition 2. Let RS_D be the $D \times D^*$ generator matrix of the $[D^* = 2D - 1, D, D]_q$ systematic Reed-Solomon code. Let $m = 2D$ and $d = D^*$. Define $\mathcal{V} = \mathcal{W} = \begin{pmatrix} RS_D \\ RS_D \end{pmatrix}$. Let $Q^{\text{wc}} := (\mathbf{0}, \mathbf{0}, \mathcal{V}, \mathcal{W}, \psi)$.

We informally define the degree $\text{sdeg } P$ of a (quadratic) span program P as the degree of the interpolating polynomial that obtains the value v_{ij} at point j . See Sect. 8 for a formal definition.

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

Proof. It is easy to see that if \mathbf{a} and \mathbf{b} indicate a consistent bit assignment, then the new wire checker accepts. For example, if $\mathbf{a}^{(1)} = \mathbf{b}^{(2)} = \mathbf{0}$, then clearly $\sum_{i=1}^D a_i v_{ij} = 0$ for $j \in [1, D^*]$ and $\sum_{i=D+1}^{2D} b_i w_{ij} = 0$ for $j \in [1, D^*]$.

Now, assume that \mathbf{a} and \mathbf{b} indicate an inconsistent bit assignment, that is, $\mathbf{a}^{(k)} \neq \mathbf{0}$ and $\mathbf{b}^{(k)} \neq \mathbf{0}$ for either $k = 1$ or $k = 2$. W.l.o.g., assume that $\mathbf{a}^{(1)} \neq \mathbf{0}$ and $\mathbf{b}^{(1)} \neq \mathbf{0}$. Since RS_D is the generator matrix of the systematic Reed-Solomon code, the vector $(\mathbf{a}^{(1)})^\top \cdot RS_D$ has at least $d > D^*/2$ non-zero coefficients. Similarly, so does $(\mathbf{b}^{(1)})^\top \cdot RS_D$. That means that both $\sum_{i=1}^D a_i v_{ij}$ and $\sum_{i=1}^D b_i w_{ij}$ are non-zero for more than $D^*/2$ different values $j \in [D^*]$. Hence, there exists at least one coefficient $j \in [D^*]$, such that $(\sum_{i=1}^D a_i v_{ij})(\sum_{i=1}^D b_i w_{ij}) \neq 0$. Thus, Q^{wc} does not accept.

The claim about the size, the dimension, the degree, and the support is straightforward. \square

Intuitively, we use 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 the generator matrix of an $[n = 2D - 1, k = D, d = D]_q$ code can have, since every row of RS_D is a codeword and thus must have at least d non-zero entries, and thus RS_D must have at least $dD \geq D^2$ non-zero entries, where the last inequality is due to the singleton bound [Rot06].

We note that a wire checker with $\mathcal{V} = \mathcal{W} = RS_D$ satisfies the even stronger security requirement that either $\mathbf{a} = \mathbf{0}$ or $\mathbf{b} = \mathbf{0}$. One could hope to pair up literals corresponding to x_η in the \mathcal{V} part and literals corresponding to \bar{x}_η in the \mathcal{W} part. This is impossible in our application, since when we aggregate the wire checkers, we have to use vectors labelled with both negative and positive literals in the same part, \mathcal{V} or \mathcal{W} , and we cannot pair up columns from \mathcal{V} and \mathcal{W} that have different indices.

For labelling ψ , we define the *dual labelling* ψ_{dual} , such that $\psi_{\text{dual}}(i) = x_\eta^j$ iff $\psi(i) = x_\eta^{1-j}$. Let $\mathcal{W} = \mathcal{V}_{\text{dual}}$ be the same matrix as \mathcal{V} , 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 \mathcal{W} has been constructed as in the current paragraph.

Now, [GGPR12] constructed a *weak* wire checker that guarantees consistency when all individual gate checkers are conscientious. The new wire checker is both more efficient and more secure.

6.2 Aggregate Wire Checker

Let $P = (\mathbf{0}, \mathbf{0}, \mathcal{V}, \mathcal{W}, \psi)$, with two $m \times d$ matrices \mathcal{V} and $\mathcal{W} = \mathcal{V}_{\text{dual}}$, be a quadratic span program. P is an *aggregate wire checker*, if $(\mathcal{V} \cdot \mathbf{a} - \mathbf{t}_v) \circ (\mathcal{W} \cdot \mathbf{b} - \mathbf{t}_w) = \mathbf{0}$ if and only if \mathbf{a} and \mathbf{b} indicate consistent bit assignments in the following sense: for each $\eta \in E(C)$,

```

1 for  $i \leftarrow 1$  to  $m$  do  $\mathbf{v}_i \leftarrow \mathbf{0}$ ;  $\mathbf{w}_i \leftarrow \mathbf{0}$  ;
2 for all possible values of  $D_\eta$  of all different wire checkers do Precompute  $RS_{D_\eta}$  ;
3 for  $\eta \leftarrow 1$  to  $|E(C_{\text{bnd}})|$  do
4    $D_{cum} \leftarrow \sum_{\eta^*=1}^{\eta-1} D_{\eta^*}$  ;
5   Let  $M_\eta^v$  be the submatrix of  $\mathcal{V}$  indexed by rows  $\psi^{-1}(\bar{x}_\eta) \cup \psi^{-1}(x_\eta)$  and columns  $[2D_{cum} + 1, 2D_{cum} + 2D_\eta]$ ;
6   Let  $M_\eta^w$  be the same submatrix of  $\mathcal{W}$ ;
7   Set  $M_\eta^v \leftarrow (RS_{D_\eta}^\top | RS_{D_\eta}^\top)$  and  $M_\eta^w \leftarrow (RS_{D_\eta}^\top | RS_{D_\eta}^\top)$ ;
8 end

```

Protocol 1: The new aggregate wire checker Q^{awc}

1. either $a_i = 0$ for all $i \in \psi^{-1}(\bar{x}_\eta)$ or $b_i = 0$ for all $i \in \psi^{-1}(x_\eta)$, and
2. either $a_i = 0$ for all $i \in \psi^{-1}(x_\eta)$ or $b_i = 0$ for all $i \in \psi^{-1}(\bar{x}_\eta)$.

We construct an aggregate wire checker by AND-composing wire checkers for the individual wires. The aggregate wire checker, see Prot. 1, first resets all vectors \mathbf{v}_i and \mathbf{w}_i to $\mathbf{0}$, and precomputes RS_{D_η} for all possible values D_η (clearly, $D_\eta \leq t + 1$). After that, for every wire η , it sets the entries in rows, labelled by either x_η or \bar{x}_η , and columns corresponding to wire η , according to the η th wire checker. The variables D_η and D_{cum} are defined as in Sect. 6.1.

We recall from Sect. 6.1 that for the wire checker of some wire to work, it must be the case that the vectors in \mathcal{V} and \mathcal{W} of this wire checker have different (but consistent) orderings. To keep notation simple, we will not mention this in what follows.

Theorem 3. *Let $t \geq 2$. Assume that C_{bnd} is the circuit, obtained by the transformation described in Thm. 1. For any $\eta \in E(C_{\text{bnd}})$, denote $D_\eta^* = 2D_\eta - 1$. We obtain an aggregate wire checker Q^{awc} , see Prot. 1, by merging wire checkers for the individual indices $\eta \in E(C_{\text{bnd}})$ as in Prot. 1 from the span program S that compute the aggregate gate checker function agc for C_{bnd} .*

Proof. Let m be the size of the aggregate wire checker (computed in Thm. 4). If \mathbf{a}, \mathbf{b} indicate consistent assignments, then they indicate consistent assignments of the η th bit for i restricted to $\psi^{-1}(\bar{x}_\eta) \cup \psi^{-1}(x_\eta)$. For every $\eta \in E(C_{\text{bnd}})$, the wire checker for wire η guarantees that $(\sum_{i=1}^m a_i v_{ij})(\sum_{i=1}^m b_i w_{ij}) = 0$ for $j \in [2\sum_{\eta^*=1}^{\eta-1} D_{\eta^*} + 1, 2\sum_{\eta^*=1}^{\eta-1} D_{\eta^*}]$ iff the bit assignments of the η th wire are consistent. Thus, $(\sum_{i=1}^m a_i v_{ij})(\sum_{i=1}^m b_i w_{ij}) = 0$ for $j \in [1, \text{sdim } Q^{\text{awc}}]$ iff the bit assignments of all wires are consistent. \square

Theorem 4. *Let $t^* := 1/(t - 1)$. Assume C implements some $f : \{0, 1\}^{n_0} \rightarrow \{0, 1\}$, and $n = |C|$. The aggregate wire checker Q^{awc} has size $Q_{\text{awc}} \leq (6 + 4t^*)n - (4 + 8t^*)n_0 - 4$, $\text{sdim } Q_{\text{awc}} \leq (6 + 4t^*)n - (4 + 8t^*)n_0 - 4$, $\text{sdeg } Q_{\text{awc}} \leq (3 + 2t^*)n - (2 + 4t^*)n_0 - 2$, $\text{supp } Q_{\text{awc}} \leq 4(t + 1)^2((1 + t^*)n - 2t^*n_0 - 1)$. If $t = 3$, then size $Q_{\text{awc}} \leq 8n - 8n_0 - 4$, $\text{sdim } Q_{\text{awc}} \leq 8n - 8n_0 - 4$, $\text{sdeg } Q_{\text{awc}} \leq 4n - 4n_0 - 2$, and $\text{supp } Q_{\text{awc}} \leq 72n - 72n_0 - 36$.*

(The proof of this theorem is given in App. C.) Clearly, other parameters but support are minimized when t is large. If the support is not important than one can dismiss the bounding fan-out step, and get size $2n$, dimension $2n$ and degree n .

Gennaro et al [GGPR12] only defined a weak aggregate wire checker that guarantees the required “no double assignments” property only when the individual gate checkers are conscientious. The new aggregate wire checker does not have this restriction. The size of the GGPR weak aggregate wire checker is $24n$ and the degree of it is $76n$. Differently from [GGPR12], we gave the description of our aggregate wire checker by using the non-polynomial interpretation of quadratic span program.

7 Circuit Checker

Next, we combine the aggregate gate and wire checkers to perform the verification of a Circuit-SAT instance. We will give two different descriptions of the resulting *circuit checker*², based on wire checkers.

Combined Circuit Checker. We construct the combined circuit checker for C as follows: let $P_w = (\mathbf{0}, \mathbf{0}, \mathcal{V}_w, \mathcal{W}_w, \psi)$, be an aggregate wire checker for C_{bnd} . Let $P_g = (\mathbf{t}, \mathcal{V}_g, \varrho)$ be an aggregate gate checker for C_{bnd} . Here, ϱ and ψ are related as in Sect. 6.2. Let $m = \text{size } P_w = \text{size } P_g$. Assume that the vectors $\mathcal{V}_w = \{\mathbf{v}_1^w, \dots, \mathbf{v}_m^w\}$ and $\mathcal{V}_g = \{\mathbf{v}_1^g, \dots, \mathbf{v}_m^g\}$ (and similarly, $\mathcal{W}_w = \{\mathbf{w}_1^w, \dots, \mathbf{w}_m^w\}$ and \mathcal{V}_g) are ordered consistently (see Sect. 6.2).

Definition 3. *The combined circuit checker $c_A(C)$ for C consists of P_g and P_w . It accepts \mathbf{u} (that is, $c_A(C)(\mathbf{u}) = 1$) if there exist two vectors \mathbf{a} and \mathbf{b} , such that $a_i = b_i = 0$ for $i \notin \psi_{\mathbf{u}}^{-1}$, which make both P_g and P_w simultaneously accept, in the sense that the following holds true:*

1. $\sum_i a_i \mathbf{v}_i^g = \mathbf{t}$,
2. $\sum_i b_i \mathbf{v}_i^g = \mathbf{t}$,
3. $(\sum_i a_i v_{ij}^w)(\sum_i b_i w_{ij}^w) = 0$ for $j \in [d]$.

We note that the instantiation of P_g used in conjunction with vector \mathbf{b} differs from the instantiation used in conjunction with vector \mathbf{a} : as explained in Sect. 6.1, the two instantiations have a different ordering of the vectors \mathbf{v}_i^g . To ease on notation, we will not make it explicit.

Theorem 5. *Assume that P_w is an aggregate wire checker. $C(\mathbf{u}) = 1$ iff $c_A(C)(\mathbf{u}) = 1$.*

Proof. First, assume $C(\mathbf{u}) = 1$. By the construction of the aggregate gate checker, there exists \mathbf{a} , with $a_i = 0$ for $i \notin \psi_{\mathbf{u}}^{-1}$, such that $\sum_{i=1}^m a_i \mathbf{v}_i^g = \mathbf{t}$. Let $\mathbf{b} \leftarrow \mathbf{a}$, then also $\sum b_i \mathbf{v}_i^g = \mathbf{t}$. Since \mathbf{a} and \mathbf{b} indicate bit assignments of wires in the evaluation of $C(\mathbf{u})$, the aggregate wire checker accepts.

Second, assume that there exist vectors \mathbf{a} and \mathbf{b} , such that $c_A(C)$ accepts with those vectors. Since P_w accepts, there are no double assignments. That means, that for some (possibly non-unique) bit $u_\eta \in \{0, 1\}$ and all $i \in \psi^{-1}(x_\eta^{\bar{u}_\eta})$, $a_i = 0$. Dually, $b_i = 0$ for all $i \in \psi^{-1}(x_\eta^{\bar{u}_\eta})$ (u_η clearly has to be the same in both cases). Since this holds for every wire, we get that there exists an assignment \mathbf{u} of wire values, such that for all $i \notin \psi_{\mathbf{u}}^{-1}$, $a_i = b_i = 0$. Moreover, $C(\mathbf{u}) = 1$. \square

Pure circuit checker. The previous construction of $c_A(C)$ consists of two span programs and one quadratic span program. Following the ideas of [GGPR12], one can represent everything as one (slightly larger) quadratic span program. Namely, for $d_g = \text{sdim } P_g$, consider the quadratic span program

$$c_A(C) : \begin{pmatrix} \mathcal{V} \\ \mathcal{W} \end{pmatrix} = \begin{pmatrix} \mathbf{t} & \mathbf{1} & \mathbf{0} \\ \mathcal{V}_g & 0_{d_g \times d_g} & \mathcal{V}_w \\ \mathbf{1} & \mathbf{t} & \mathbf{0} \\ 0_{d_g \times d_g} & \mathcal{W}_g & \mathcal{W}_w \end{pmatrix}. \quad (1)$$

Here, $\mathcal{V} = (\mathbf{v}_0, \dots, \mathbf{v}_m)^\top$, $\mathcal{W} = (\mathbf{w}_0, \dots, \mathbf{w}_m)^\top$, and $\mathbf{1} = (1, \dots, 1)$. Clearly, $(\sum_i a_i v_{ij} - v_{0j})(\sum_i b_i w_{ij} - w_{0j}) = 0$ for $j \in [1, 2 \cdot \text{sdim } P_g + \text{sdim } P_w]$ iff the following three things hold:

- $(\sum a_i v_{ij}^g - t_j)(0 - 1) = 0$ for $j \in [\text{sdim } P_g]$ holds iff $\sum a_i v_{ij}^g = t_j$ for $j \in [\text{sdim } P_g]$ iff $\sum a_i \mathbf{v}_i^g = \mathbf{t}$,
- $(0 - 1)(\sum b_i v_{ij}^g - t_j) = 0$ for $j \in [\text{sdim } P_g]$ holds iff $\sum b_i v_{ij}^g = t_j$ for $j \in [\text{sdim } P_g]$ iff $\sum b_i \mathbf{v}_i^g = \mathbf{t}$,
- $(\sum a_i v_{ij}^w) \cdot (\sum b_i w_{ij}^w) = 0$ for $j \in [\text{sdim } P_w]$.

² This was called a canonical quadratic span program in [GGPR12]. However, the notion of canonical span programs was already introduced in [KW93], and has a completely different definition. Therefore, we have changed the terminology

Thus, this quadratic span program accepts iff the combined circuit checker accepts. Thus, $c_A(C)$ is a circuit checker for C . However, it also increases the dimension of the final quadratic span program.

Clearly, $\text{sdeg } c_A(C) \leq 2 \cdot \text{sdim } P_g + \text{sdeg } \mathcal{V}_w$. Let $n = |C|$. From Thm. 1 and Thm. 4, $\text{sdeg } c_A(C) \leq (11 + 6t^*)n - 12(1 + t^*)n_0 - 2$. This decreases when t increases, obtaining the value $\leq 11n - 12n_0 - 2$ when one does not apply Thm. 1 at all, or $14n - 18n_0 - 2$ when $t = 3$. Analogously, $\text{size } c_A(C) = \text{size } P_w + \text{size } P_g \leq 2(7 + 4t^*)n - 2(7 + 8t^*)n_0 - 4$. This decreases when t increases, obtaining the value $\leq 14n - 14n_0 - 4$ when one does not apply Thm. 1 at all, or $18n - 22n_0 - 4$ when $t = 3$.

One can similarly compute the dimension and the support $\text{supp } c_A(C) = 2(\text{supp } P_g + \text{supp } P_w) = (50 + 8t(3 + t) + 40t^*)n - 2(5 + t(27 + 8t))t^*n_0 - 8(1 + t)^2$ of the circuit checker. The support is upperbounded by $214n - 158n_0 - 128$ when $t = 3$. We note that the degree of the circuit checker from [GGPR12] is $130n$ and its size is $36n$. Thus, even when $t = 3$, we have improved the efficiency of their construction more than 9 times degree-wise and 2 times size-wise.

8 Polynomial Span Programs and Quadratic Span Programs

One can build a linear-communication NIZK argument on top of the circuit checker by using well-known techniques. However, since we are interested in succinct arguments, we need to be able to somehow compress the witness vectors \mathbf{a} and \mathbf{b} . As in [GGPR12], we will do it by using polynomial interpolation.

For large prime q , let $\mathbb{F} = \mathbb{Z}_q$. Instead of considering the target and row vectors as being members of the vector space \mathbb{F}^d , we reinterpret them as degree- d polynomials in $\mathbb{F}[X]$. The map $\mathbf{v} \rightarrow v(X)$ is implemented via first choosing d arbitrary but different field elements (that are the same for all vectors \mathbf{v}) $r_j \leftarrow \mathbb{F}$, and then defining a degree- $(\leq d)$ polynomial $v(X)$ via polynomial interpolation to be such that $v(r_j) = v_j$ for all $j \in [d]$. (For our purpose, the choice of r_j only influences efficiency. Namely, if r_j are arbitrary, then multipoint evaluation and polynomial interpolation can be performed in time $O(d \log^2 d)$. However, if d is a power of 2 and $r_j = \omega_d^j$, where ω_d is the d th primitive root of unity, then both operations can be done in time $O(d \log d)$ by using Fast Fourier Transform, see App. D.) Via this conversion, one maps all vectors \mathbf{v}_i of the original span program to polynomials $v_i(X)$. The target vector \mathbf{t} of the span program is mapped to a polynomial $v_0(X)$, where $v_0(r_j) = -t_j$ for $j \in [d]$. Finally, one defines the polynomial $z(X) = \prod_{j=1}^d (x - r_j)$. Note that $z(X)$ is the mapping of the all-zero vector $\mathbf{0} = (0, \dots, 0)$.

The requirement that \mathbf{t} is in the span of the vectors that belong to $\varrho_{\mathbf{u}}^{-1}$ is equivalent to the requirement that $\mathbf{t} = \sum_{i \in \varrho_{\mathbf{u}}^{-1}} a_i \mathbf{v}_i$ for some $a_i \in \mathbb{F}$. In the polynomial notation, the latter translates to the requirement that $z(X)$ divides $v(X) := v_0(X) + \sum_{i \in \varrho_{\mathbf{u}}^{-1}} a_i v_i(X)$. (See Lem. 5 of [GGPR12].) This is since $-\mathbf{t}$ is the vector of evaluations (at r_1, \dots, r_d) of $v_0(X)$, and \mathbf{v}_i is the vector of evaluations of $v_i(X)$. Thus, $\sum a_i \mathbf{v}_i - \mathbf{t} = \mathbf{0}$ holds iff $v_0(X) + \sum a_i v_i(X)$ evaluates to 0 at all r_j , and hence is divisible by $z(X)$.

Definition 4. A polynomial span program P over a field \mathbb{F} consists of a target polynomial $z(X) \in \mathbb{F}[X]$, a tuple $\mathcal{V} = (v_0(X), v_1(X), \dots, v_m(X))$ of polynomials from $\mathbb{F}[X]$, and a labelling $\varrho : [m] \rightarrow \{x_i, \bar{x}_i : i \in [n]\} \cup \{\perp\}$ of the polynomials from $\mathcal{V} \setminus \{v_0\}$. Let $\mathcal{V}_{\mathbf{u}}$ be a subset of $\mathcal{V} \setminus \{v_0\}$ consisting of those polynomials whose labels are satisfied by the assignment \mathbf{u} , that is, by $\{x_i^{u_i} : i \in [n]\} \cup \{\perp\}$. The span program P computes a function f , iff for all $\mathbf{u} \in \{0, 1\}^n$: there exists $\mathbf{a} \in \mathbb{F}^m$ such that $z(X) \mid (v_0(X) + \sum_{v \in \mathcal{V}_{\mathbf{u}}} a_i v(X))$ (P accepts) iff $f(\mathbf{u}) = 1$.

Alternatively, P accepts $\mathbf{u} \in \{0, 1\}^n$ iff there exists a vector $\mathbf{a} \in \mathbb{F}^m$, with $a_i = 0$ for all $i \notin \varrho_{\mathbf{u}}^{-1}$, such that $z(X) \mid v_0(X) + \sum_{i=1}^m a_i v_i(X)$. The size of the span program is m and the degree of P is $\deg z(X)$. We now give exactly the same definition of quadratic span programs as it was given in [GGPR12].

Definition 5. A polynomial quadratic span program P over a field \mathbb{F} consists of a target polynomial $z(X) \in \mathbb{F}[X]$, two sets $\mathcal{V} = \{v_0(X), v_1(X), \dots, v_m(X)\}$ and $\mathcal{W} = \{w_0(X), w_1(X), \dots, w_m(X)\}$ of polynomials from $\mathbb{F}[X]$, and a labelling $\varrho : [m] \rightarrow \{x_i, \bar{x}_i : i \in [n]\} \cup \{\perp\}$. P accepts an input $\mathbf{u} \in \{0, 1\}^n$ iff there exist two vectors \mathbf{a} and \mathbf{b} from \mathbb{F}^m , with $a_i = 0 = b_i$ for all $i \notin \varrho_{\mathbf{u}}^{-1}$, such that $z(X) \mid (v_0(X) + \sum_{i=1}^m a_i v_i(X))(w_0(X) + \sum_{i=1}^m b_i w_i(X))$. P computes a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ if it accepts exactly those inputs \mathbf{u} where $f(\mathbf{u}) = 1$.

The size of the polynomial quadratic span program is m and the degree of P is $\deg z(X)$.

Now, keeping in mind the reinterpretation of span programs, Def. 5 is clearly equivalent to Def. 1. (Also here, $\mathcal{W} = \mathcal{V}_{\text{dual}}$, with the dual operation defined appropriately.)

To get from the non-polynomial interpretation to polynomial interpretation, one has to do the following. Assume that the dimension of the quadratic span program is d and that the size is m . Choose d different values r_j , $j \in [d]$. For $i \in [m]$, interpolate the polynomial $v_i(X)$ (resp., $w_i(X)$) from the values $v_i(r_j) = v_{ij}$ (resp., $w_i(r_j) = w_{ij}$) 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 quadratic span program $(z(X), \mathcal{V}, \mathcal{W}, \psi)$ computes the same Boolean function as the original quadratic span program.

Polynomial circuit checker. Fig. 3 describes a polynomial circuit checker $c_A^{\text{poly}}(C) = (z(X), \mathcal{V}, \mathcal{W}, \psi)$, with $\mathcal{V} = (v_0, \dots, v_m)$ and $\mathcal{W} = (w_0, \dots, w_m)$. It is directly constructed from the above pure circuit checker $c_A(C)$. Here, PI denotes polynomial interpolation, and $d = 2d_g + d_w$.

Theorem 6. $C(X) = 1$ iff $c_A^{\text{poly}}(C)$ outputs 1.

Proof. Follows from the general construction of polynomial quadratic span programs. \square

9 New NIZK Argument

In this section, we propose the new Circuit-SAT NIZK argument. We start with definitions.

Definitions. Let κ be the security parameter. We abbreviate probabilistic polynomial-time by PPT. Let $\text{poly}(\kappa) := \kappa^{O(1)}$ and $\text{negl}(\kappa) := \kappa^{-\omega(1)}$.

Let $R = \{(C, w)\}$ be an efficiently computable binary relation with $|w| = \text{poly}(|C|)$. Here, C is a statement, and w is a witness. Let $\mathcal{L} = \{C : \exists w, (C, w) \in R\}$ be an NP-language. Let n be the input length $n = |C|$. For fixed n , we have a relation R_n and a language \mathcal{L}_n . A *non-interactive argument* Π for R consists of the following PPT algorithms: a common reference string (CRS) generator \mathcal{G} , a prover \mathcal{P} , and a verifier \mathcal{V} . For $\text{crs} \leftarrow \mathcal{G}(1^\kappa, n)$, $\mathcal{P}(\text{crs}; C, w)$ produces an argument π . The verifier $\mathcal{V}(\text{crs}; C, \pi)$ outputs either 1 (accept) or 0 (reject). Π is *perfectly complete*, if $\forall n = \text{poly}(\kappa)$,

$$\Pr[\text{crs} \leftarrow \mathcal{G}(1^\kappa, n), (C, w) \leftarrow R_n : \mathcal{V}(\text{crs}; C, \mathcal{P}(\text{crs}; C, w)) = 1] = 1 .$$

Π is *perfectly zero-knowledge*, if there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, such that for all stateful non-uniform PPT adversaries \mathcal{A} and $n = \text{poly}(\kappa)$ (with td being the *simulation trapdoor*),

$$\Pr \left[\begin{array}{l} \text{crs} \leftarrow \mathcal{G}(1^\kappa, n), (C, w) \leftarrow \mathcal{A}(\text{crs}), \\ \pi \leftarrow \mathcal{P}(\text{crs}; C, w) : (C, w) \in R_n \wedge \mathcal{A}(\pi) = 1 \end{array} \right] = \Pr \left[\begin{array}{l} (\text{crs}; \text{td}) \leftarrow \mathcal{S}_1(1^\kappa, n), (C, w) \leftarrow \mathcal{A}(\text{crs}), \\ \pi \leftarrow \mathcal{S}_2(\text{crs}; C, \text{td}) : (C, w) \in R_n \wedge \mathcal{A}(\pi) = 1 \end{array} \right] .$$

Π is *adaptively computationally sound*, if for all non-uniform PPT \mathcal{A} and all $n = \text{poly}(\kappa)$,

$$\Pr[\text{crs} \leftarrow \mathcal{G}(1^\kappa, n), (C, \pi) \leftarrow \mathcal{A}(\text{crs}) : C \notin \mathcal{L} \wedge \mathcal{V}(\text{crs}; C, \pi) = 1] = \text{negl}(\kappa) .$$

For algorithms \mathcal{A} and $\mathcal{E}_{\mathcal{A}}$, we write $(y; z) \leftarrow (\mathcal{A} || \mathcal{E}_{\mathcal{A}})(x)$ if \mathcal{A} on input x outputs y , and $\mathcal{E}_{\mathcal{A}}$ on the same input (including the random tape of \mathcal{A}) outputs z . A non-interactive argument is a *non-interactive argument of knowledge*, if for any non-uniform PPT prover \mathcal{A} there exists an extractor $\mathcal{E}_{\mathcal{A}}$ such that for $n = \text{poly}(\kappa)$ and any auxiliary information $z \in \{0, 1\}^\kappa$,

$$\Pr[\text{crs} \leftarrow \mathcal{G}(1^\kappa, n), (C, \pi; w) \leftarrow (\mathcal{A} || \mathcal{E}_{\mathcal{A}})(\text{crs}; z) : \mathcal{V}(\text{crs}; C, \pi) = 1 \wedge (C, w) \notin R] = \text{negl}(\kappa) .$$

```

1 for  $j \leftarrow 1$  to  $d$  do  $r_j \leftarrow \mathbb{F} \setminus \{r_1, \dots, r_{j-1}\}$ ;
2  $z(X) \leftarrow \prod_{j \in [d]} (X - r_j)$ ;
3 for  $i \leftarrow 0$  to  $m$  do
4    $v_i(X) \leftarrow PI((r_1, v_{i1}), \dots, (r_d, v_{id}))$ ;
5    $w_i(X) \leftarrow PI((r_1, w_{i1}), \dots, (r_d, w_{id}))$ ;
6 end
7  $\mathcal{V} \leftarrow (v_0(X), \dots, v_m(X))$ ;
8  $\mathcal{W} \leftarrow (w_0(X), \dots, w_m(X))$ ;
9 return  $c_A(C) = (z(X), \mathcal{V}, \mathcal{W}, \psi)$ ;

```

Fig. 3. Polynomial circuit checker $c_A^{\text{poly}}(C)$

Construction. Gennaro et al [GGPR12] constructed a quadratic span program-based NIZK for Circuit-SAT. Their NIZK argument is non-adaptive, i.e., it incorporates a function-specific CRS $\text{crs}(C)$. As mentioned in [GGPR12], their argument can be made adaptive by using universal circuits [Val76]. We will give a direct construction through universal circuits [Val76]. That is, we assume that UC_n is a universal circuit of size $\Theta(n \log n)$ that accepts an input (C, \mathbf{u}) iff C is an n -gate circuit that accepts the input \mathbf{u} . More precisely, assuming that the original circuit has fan-out 2, Valiant's universal circuit consists of $19n \log n$ controlled crossbar gates (omitting lower-order terms), and $\Theta(n)$ gates for the universal function from $\{0, 1\}^2$ to $\{0, 1\}$. To enable a better comparison with [GGPR12], we just assume that UC_n can be implemented by using $\Theta(n \log n)$ unary or binary gates.

In the new Circuit-SAT NIZK argument, polynomials (e.g., $v_i(X)$) are represented by encodings of these polynomials evaluated at some secret point σ (e.g., by $\{\llbracket v_i(\sigma) \rrbracket\}$), where $\llbracket \cdot \rrbracket$ is an additively homomorphic encoding scheme for elements of \mathbb{F} . That is, $\llbracket a \rrbracket \cdot \llbracket b \rrbracket = \llbracket a+b \rrbracket$ and $\llbracket a \rrbracket^x = \llbracket xa \rrbracket$. We assume that the encoding is equal to the bilinear exponentiation, $\llbracket x \rrbracket = g^x$ in a (symmetric) bilinear group. Thus, one can verify bilinear relations between encodings by using a symmetric bilinear map $\hat{e}(\cdot, \cdot)$ [SOK00, Jou00, BF01]. For example, $\hat{e}(g = \llbracket 1 \rrbracket, \llbracket \alpha x \rrbracket) = \hat{e}(\llbracket \alpha \rrbracket, \llbracket x \rrbracket)$. By taking some extra care, one can also use an asymmetric pairing.

Next, we will describe the three constituent algorithms of the new NIZK argument. Briefly, the prover uses the circuit checker to create the necessary coefficients a_i and b_i . He encodes $v(X) = \sum a_i v_i(X)$ and $w(X) = \sum b_i w_i(X)$ as $\llbracket v(\sigma) \rrbracket$ and $\llbracket w(\sigma) \rrbracket$. He also encodes $\llbracket h(\sigma) \rrbracket$, where $h(X)z(X) = v(X)w(X)$. He then creates an argument π that convinces the verifier that $h(X)$ satisfies this condition.. To achieve zero-knowledge, the prover additionally masks the argument accordingly. The verifier verifies that the argument is created correctly. She also verifies that $h(X)$, $v(X)$ and $z(X)$ satisfy $h(\sigma)z(\sigma) = v(\sigma)w(\sigma)$. We prove in Thm. 8 that it follows from this and the PDH and PKE assumptions, $h(X)z(X) = v(X)w(X)$ and thus $z(X) \mid v(X)w(X)$. Computational soundness follows due to the properties of the circuit checker. The actual proof is significantly more complicated. Since the verifier will require less information than the prover, we define the verifier's CRS separately as vcrs . As in [GGPR12], the secret elements β_v, β_w and γ (and corresponding public elements like $\llbracket \beta_v z(\sigma) \rrbracket$ and π_y) are required for us to be able to reduce the soundness to the PKE and PDH assumptions. This part of the proof also uses multilinear universal hash functions.

CRS generation $\mathcal{G}(1^\kappa, n)$:

- 1 Let UC_n be the universal circuit for circuits of size $|C_{\text{bnd}}|$, where $|C| = n$;
- 2 Let $Q := c_A(\text{UC}_n) = (z(X), \mathcal{V}, \mathcal{W}, \psi)$ be the pure polynomial circuit checker for UC_n with $m = \text{size } c_A(\text{UC}_n)$, $d = \text{sdeg } c_A(\text{UC}_n)$, $\mathcal{V} = (v_0(X), \dots, v_m(X))$, and $\mathcal{W} = \mathcal{V}_{\text{dual}} = (w_0(X), \dots, w_m(X))$;
- 3 $\alpha, \sigma, \beta_v, \beta_w, \gamma \leftarrow \mathbb{F}$;
- 4 $(V_0, V_0^*) \leftarrow (\llbracket v_0(\sigma) \rrbracket, \llbracket \alpha v_0(\sigma) \rrbracket)$; $(W_0, W_0^*) \leftarrow (\llbracket w_0(\sigma) \rrbracket, \llbracket \alpha w_0(\sigma) \rrbracket)$; $(Z, Z^*) \leftarrow (\llbracket z(\sigma) \rrbracket, \llbracket \alpha z(\sigma) \rrbracket)$;
- 5 $\text{crs} \leftarrow (Q, (\llbracket \sigma^j \rrbracket, \llbracket \alpha \sigma^j \rrbracket)_{j \in [0, d]}$, $V_0, V_0^*, W_0, W_0^*, Z, Z^*, (\llbracket \beta_v v_i(\sigma) \rrbracket, \llbracket \beta_w w_i(\sigma) \rrbracket)_{i \in [m]}$, $\llbracket \beta_v z(\sigma) \rrbracket, \llbracket \beta_w z(\sigma) \rrbracket)$;
- 6 $\text{vcrs} \leftarrow (\llbracket 1 \rrbracket, \llbracket \alpha \rrbracket, V_0, W_0, Z, \llbracket \gamma \rrbracket, \llbracket \beta_v \gamma \rrbracket, \llbracket \beta_w \gamma \rrbracket)$;
- 7 The trapdoor is $(\sigma, \alpha, \beta_v, \beta_w)$;

Prove $\mathcal{P}(\text{crs}; C, w)$:

- 1 \mathcal{P} evaluates $c_A(\text{UC}_n)$ on input (C, w) to obtain (i) $\mathbf{a}, \mathbf{b} \in \mathbb{F}^m$, (ii) $v(X) = \sum_{j=0}^d \hat{v}_j X^j \leftarrow \sum_{i=1}^m a_i v_i(X)$, (iii) $w(X) = \sum_{j=0}^d \hat{w}_j X^j \leftarrow \sum_{i=1}^m b_i w_i(X)$, and (iv) a polynomial $h(X)$ such that for $v^\dagger(X) = v_0(X) + v(X)$ and $w^\dagger(X) = w_0(X) + w(X)$, $h(X) \cdot z(X) = v^\dagger(X)w^\dagger(X)$;
- 2 $(V, V^*) \leftarrow \prod_{j=0}^d (\llbracket \sigma^j \rrbracket^{\hat{v}_j}, \llbracket \alpha \sigma^j \rrbracket^{\hat{v}_j})$;
- 3 $(W, W^*) \leftarrow \prod_{j=0}^d (\llbracket \sigma^j \rrbracket^{\hat{w}_j}, \llbracket \alpha \sigma^j \rrbracket^{\hat{w}_j})$;
- 4 $(H, H^*) \leftarrow \prod_{j=0}^d (\llbracket \sigma^j \rrbracket^{h_j}, \llbracket \alpha \sigma^j \rrbracket^{h_j})$;
- 5 $r_v, r_w \leftarrow \mathbb{F}$;
- 6 $(\pi_v, \pi_v^*) \leftarrow (V, V^*) \cdot (Z, Z^*)^{r_v}$;
- 7 $(\pi_w, \pi_w^*) \leftarrow (W, W^*) \cdot (Z, Z^*)^{r_w}$;
- 8 $(\pi_h, \pi_h^*) \leftarrow (H, H^*) \cdot (W_0 W, W_0^* W^*)^{r_v} (V_0 V, V_0^* V^*)^{r_w} \cdot (Z, Z^*)^{r_v r_w}$;
- 9 $\pi_y \leftarrow \prod_{i=1}^m (\llbracket \beta_v v_i(\sigma) \rrbracket^{a_i} \llbracket \beta_w w_i(\sigma) \rrbracket^{b_i}) \cdot \llbracket \beta_v z(\sigma) \rrbracket^{r_v} \llbracket \beta_w z(\sigma) \rrbracket^{r_w}$;
- 10 \mathcal{P} outputs $\pi = (\pi_v, \pi_v^*, \pi_w, \pi_w^*, \pi_h, \pi_h^*, \pi_y)$;

Verify $\mathcal{V}(\text{vcrs}; C, \pi)$:

- 1 \mathcal{V} confirms that the terms are in the support of validly encoded elements;
- 2 \mathcal{V} confirms that the following equations hold:
 1. $\hat{e}(V_0\pi_v, W_0\pi_w) = \hat{e}(\pi_h, Z)$;
 2. $\hat{e}(\pi_v^*, \llbracket \alpha \rrbracket) = \hat{e}(\pi_v, \llbracket 1 \rrbracket)$;
 3. $\hat{e}(\pi_w^*, \llbracket \alpha \rrbracket) = \hat{e}(\pi_w, \llbracket 1 \rrbracket)$;
 4. $\hat{e}(\pi_h^*, \llbracket \alpha \rrbracket) = \hat{e}(\pi_h, \llbracket 1 \rrbracket)$;
 5. $\hat{e}(\pi_y, \llbracket \gamma \rrbracket) = \hat{e}(\pi_v, \llbracket \beta_v \gamma \rrbracket) \cdot \hat{e}(\pi_w, \llbracket \beta_w \gamma \rrbracket)$;

Theorem 7. *The NIZK argument of this section is complete and statistically zero-knowledge.*

Proof. COMPLETENESS: the properties of the circuit checker guarantee that the prover can always construct an argument for a satisfying input. Let $v^\dagger(X) = v_0(X) + \sum a_i v_i(X)$ and $w^\dagger(X) = w(X) + \sum b_i w_i(X)$. Since the rest is trivial, we will only prove that the first verification equation holds. The discrete logarithm of the left hand side of this equation is equal to

$$(v^\dagger(\sigma) + r_v z(\sigma)) \cdot (w^\dagger(\sigma) + r_w z(\sigma)) = v^\dagger(\sigma) \cdot w^\dagger(\sigma) + v^\dagger(\sigma) \cdot r_w z(\sigma) + w^\dagger(\sigma) \cdot r_v z(\sigma) + r_v r_w z^2(\sigma) ,$$

while the discrete logarithm of the right hand side is equal to

$$(h(\sigma) + r_v w^\dagger(\sigma) + r_w v^\dagger(\sigma) + r_v r_w z(\sigma)) \cdot z(\sigma) = h(\sigma) z(\sigma) + v^\dagger(\sigma) \cdot r_w z(\sigma) + w^\dagger(\sigma) \cdot r_v z(\sigma) + r_v r_w z^2(\sigma) ,$$

and thus the LHS and RHS are equal since $h(\sigma) z(\sigma) = v^\dagger(\sigma) w^\dagger(\sigma)$.

ZERO-KNOWLEDGE: We construct the following simulator (S_1, S_2) . S_1 outputs crs and a trapdoor $\text{td} \leftarrow (\sigma, \alpha, \beta_v, \beta_w, \gamma)$.

Consider the distribution of the real argument. Let (V, V^*) , (W, W^*) , (H, H^*) , and Y be what is encoded by the elements in π . Since σ is random, $z(\sigma)$ is in \mathbb{F}^* with overwhelming probability $1 - \text{sdeg } c_\Lambda(C)/|\mathbb{F}|$. Since r_v and r_w are uniformly random, $V = v(\sigma) + r_v z(\sigma)$ and $W = w(\sigma) + r_w z(\sigma)$ are statistically close to uniform. Once V and W are fixed, they determine all other elements V^* (2nd equation), W^* (3rd equation), Y (5th equation), H (1st equation), and H^* (4th equation) that are encoded in the proof.

Motivated by this discussion, we construct the simulator S_2 which is depicted by Prot. 2. Clearly, π_v and π_w encode statistically uniform values $v(\sigma) + r_v z(\sigma)$ and $w(\sigma) + r_w z(\sigma)$, while the rest of the argument is fixed by these two values. The theorem follows. \square

- 1 S_2 picks random degree- d polynomials $v^\dagger(X), w^\dagger(X)$ such that $z(X)$ divides $v^\dagger(X)w^\dagger(X)$;
- 2 $h(X) \leftarrow v^\dagger(X)w^\dagger(X)/z(X)$;
- 3 $v(\sigma) \leftarrow v^\dagger(\sigma) - v_0(\sigma)$, $w(\sigma) \leftarrow w^\dagger(\sigma) - w_0(\sigma)$;
- 4 $r_v, r_w \leftarrow \mathbb{F}$;
- 5 $\pi_v \leftarrow \llbracket v(\sigma) \rrbracket \cdot \llbracket z(\sigma) \rrbracket^{r_v}$, $\pi_w \leftarrow \llbracket w(\sigma) \rrbracket \cdot \llbracket z(\sigma) \rrbracket^{r_w}$;
- 6 $\pi_h \leftarrow \llbracket h(\sigma) + r_v w^\dagger(\sigma) + r_w v^\dagger(\sigma) + r_v r_w z(\sigma) \rrbracket$;
- 7 $\pi_v^* \leftarrow \pi_v^\alpha$, $\pi_w^* \leftarrow \pi_w^\alpha$, $\pi_h^* \leftarrow \pi_h^\alpha$;
- 8 **return** $(\pi_v, \pi_v^*, \pi_w, \pi_w^*, \pi_h, \pi_h^*, (\pi_v^*)^{\beta_v} \cdot (\pi_w^*)^{\beta_w})$;

Protocol 2: The simulator $S_2(\text{crs}, C, \text{td})$

We base computational soundness and the argument of knowledge property on two assumptions, the (d_1, d_2) -power Diffie-Hellman $((d_1, d_2)$ -PDH) assumption and the d -power knowledge of exponent (d -PKE) assumption. Variants of these assumptions are well known, see [Gro10, Lip12, GGPR12], where their security was proven in the generic group model.

Let $d_1, d_2 = \text{poly}(\kappa)$ with $0 < d_1 < d_2$. The (d_1, d_2) -power Diffie-Hellman $((d_1, d_2)$ -PDH) assumption holds for the encoding $\llbracket \cdot \rrbracket$ if for all non-uniform PPT adversaries \mathcal{A}

$$\Pr \left[\sigma \leftarrow \mathbb{F}^*, y \leftarrow \mathcal{A}(\llbracket \sigma^j \rrbracket)_{j \in [0, d_2] \setminus \{d_1\}} : y = \llbracket \sigma^{d_1} \rrbracket \right] = \text{negl}(\kappa) .$$

Gennaro et al [GGPR12] use the $(\lambda + 1, 2\lambda)$ -PDH assumption for some $\lambda \approx 2d$, while we use the $(d + 1, 2d + 3)$ -PDH assumption for d . In both cases, d is the degree of the underlying circuit checker. The corresponding security definition in [Gro10, Lip12] is somewhat weaker, since there the adversary was required to return the secret key σ .

Let $d = \text{poly}(\kappa)$. The d -power knowledge of exponent (d -PKE) assumption [Gro10] holds for the encoding $\llbracket \cdot \rrbracket$ if for any non-uniform PPT adversary \mathcal{A} there exists a non-uniform PPT extractor $\mathcal{E}_{\mathcal{A}}$, s.t. for any auxiliary information $z \in \{0, 1\}^{\text{poly}(\kappa)}$ which is generated independently of α ,

$$\Pr \left[\begin{array}{l} \alpha, \sigma \leftarrow \mathbb{F}, (\llbracket c \rrbracket, \llbracket \hat{c} \rrbracket; a_0, \dots, a_d) \leftarrow (\mathcal{A} | \mathcal{E}_{\mathcal{A}})(\llbracket \sigma^j \rrbracket, \llbracket \alpha \sigma^j \rrbracket)_{j \in [0, d]} ; z \\ \hat{c} = \alpha c \wedge c \neq \sum_{k=0}^d a_k \sigma^k \end{array} \right] = \text{negl}(\kappa) .$$

Theorem 8. Fix the circuit size n . Let the pure circuit checker $c_A(\text{UC}_n) = (z(X), \mathcal{V}, \mathcal{W}, \psi)$ for UC_n have degree d . If the $(d + 1, 2d + 3)$ -PDH and d -PKE assumptions hold, then the NIZK argument of this section is an adaptively sound argument of knowledge.

We postpone the soundness proof to Sect. 10.

Efficiency. The new NIZK argument behaves efficiency-wise similarly to the (adaptive variant) of the Circuit-SAT argument from [GGPR12] when we recall that using universal circuits results in a logarithmic increase of most of the complexity parameters. Like the latter argument³, the new argument has CRS of size $\Theta(n \log n)$, prover's computational complexity $\Theta(n \log^3 n)$, and constant communication complexity. The main difference is that the new argument has significantly smaller constants. (As always, we assume that the complexity measures are in appropriate units like the number of group elements in the case of the CRS and argument length.)

Given \mathbf{a} , \mathbf{b} , (\mathbf{v}_i) and (\mathbf{w}_i) , both $\mathbf{v} = \sum_{i=1}^m a_i \mathbf{v}_i$ and $\mathbf{w} = \sum_{i=1}^m b_i \mathbf{w}_i$ can be computed in $\Theta(\text{supp } c_A(\text{UC}_n)) = \Theta(|\text{UC}_n|)$ time due to the sparsity of the vectors \mathbf{v}_i and \mathbf{w}_i . The only superlinear (in $|\text{UC}_n| = \Theta(d) = \Theta(n \log n)$) part of the prover's computation is the computation of the degree- d polynomial h . As explained in [GGPR12], this can be done by using multipoint evaluation and polynomial interpolation in time $\Theta(d \log^2 d)$.⁴

We note that under the mild assumption that d is a power of 2, $h(x) \leftarrow v(x)w(x)/z(x)$ can be computed in time $\Theta(d \log d)$ by using a polynomial multiplication followed by a polynomial division. This can be further optimized by letting $c(x) = b^{-1}(x) \pmod{x^d}$, where $b(x) = x^d z(1/x)$ is the reversal of $z(x)$, to be a part of the CRS. Then the prover essentially has to execute only two multiplications, first to compute $a(x) = v(x)w(x)$, and then to compute $h(x) = (a^{\text{rev}}(x)c(x))^{\text{rev}} \pmod{x^d}$. (See App. D.)

The cryptographic part of the prover's computation is dominated by 8 $\Theta(d)$ -wide multiexponentiations. One can use Pippenger's multiexponentiation algorithm [Pip80] to implement a d -wide multiexponentiation in $(2 + o(1)) \frac{d}{\log_2 d} \cdot \log \alpha + O(d)$ bilinear-group multiplications, where α is the largest exponent. We expect that the cryptographic part will dominate the prover's computation when $d = \Omega(2\sqrt{\log_2 q})$, and in practice even sooner.

³ We refer to [GGPR12] for a detailed analysis of the computational complexity issues that surround polynomial interpolation.

⁴ Briefly, compute the values of $v(X)$, $w(X)$, $v_0(X)$, $w_0(X)$ and $z(X)$ on another set of d distinct roots $\{r_i^*\}$. Since $\deg z(X) = d$, we have that $z(r_i^*) \neq 0$. Interpolate $h(X)$ from $h(r_i^*) = (v_0(r_i^*) + v(r_i^*))(w_0(r_i^*) + w(r_i^*))/z(r_i^*)$. This can be done in time $\Theta(d \log d)$, when d is a power of 2 and $r_i^* = \omega_d^i$, for the primitive d th root of unity ω_d . Otherwise, it takes $\Theta(d \log_2 d)$ steps.

On the other hand, in the construction of [GGPR12], the verifier’s computational complexity is linear in the statement length. Because of the use of universal circuits, in the adaptively sound case, the statement length is linear in the circuit size $n = |C|$. In the new argument, the verifier’s computational complexity is dominated by 11 invocations of the bilinear pairings. The main reason behind this difference is that in the argument of [GGPR12], the verifier has to compute the coefficients a_i for all $i \in [n_0]$. In the new argument, this is not necessary due to the use of “non-weak” wire checkers and the extraction technique of Lem. 2 that does not require the gate checkers to be conscientious.

Further Optimizations. We can use the result of Bitansky et al. [BCCT13] that says that any SNARK with preprocessing can be transformed into a SNARK with no preprocessing. A related result holds for NIZK arguments. We refer to [GGPR12] for a description of a number of other optimizations that all apply also to the new argument.

10 Proof of Theorem 8

Before the soundness proof we prove two technical lemmas. The first lemma basically says that the honest verifier will be unconditionally convinced, if the prover sends her the actual vectors $\mathbf{v} = \sum_{i \in [m]} a_i \mathbf{v}_i$ and $\mathbf{w} = \sum_{i \in [m]} b_i \mathbf{w}_i$ such that the (pure) circuit checker will accept. Moreover, one can extract the whole witness from this proof. A similar lemma was proven in [GGPR12] only in the case all gate checkers are conscientious. This allowed to extract the value of every wire uniquely. Our proof, that does not assume this property, extracts the values recursively (and possibly non-deterministically).

Lemma 2. *Let C be an n -gate circuit. Let $c_A(C) = (\mathbf{0}, \mathbf{0}, \mathcal{V}, \mathcal{W}, \psi)$, with $\mathcal{V} = (\mathbf{v}_0, \dots, \mathbf{v}_m)$ and $\mathcal{W} = (\mathbf{w}_0, \dots, \mathbf{w}_m)$, be a pure circuit checker for the circuit C . Let $\pi = (\mathbf{v}, \mathbf{w})$ be such that*

1. $(\mathbf{v}_0 + \mathbf{v}) \circ (\mathbf{w}_0 + \mathbf{w}) = \mathbf{0}$,
2. $\mathbf{v} \in \text{span}\{\mathbf{v}_i : i \in [m]\}$, and
3. $\mathbf{w} \in \text{span}\{\mathbf{w}_i : i \in [m]\}$.

Then, π implies unconditionally that there exists a witness $\mathbf{u} \in \{0, 1\}^n$ such that $C(\mathbf{u}) = 1$. Such \mathbf{u} can be extracted from π .

Proof. Let \mathbf{a}, \mathbf{b} be such that $\mathbf{v} = \sum_{i \in [m]} a_i \mathbf{v}_i$ and $\mathbf{w} = \sum_{i \in [m]} b_i \mathbf{w}_i$. We now show how to construct a witness \mathbf{u} such that $C(\mathbf{u}) = 1$. The construction is bottom-up recursive on the circuit. We note that if the three requirements hold then the wire checker implies that no wire η gets a double assignment.

First, let η be one of the wires starting from some input gate ι . Since the output gate of η can have a non-conscientious gate checker, η might have no assigned value. However, recall that the wire checker checks the consistency of η with all other wires that start from ι . Since this wire checker accepts, all those wires have been assigned either u_η (for an unambiguous bit $u_\eta \in \{0, 1\}$) or no value; no wire has got the assignment \bar{u}_η . If some output wire of ι got the assignment u_η , we assign u_η to all output wires of ι . Otherwise, we pick some value $u_\eta \in \{0, 1\}$, and assign this u_η to all output wires of ι . Since the wire was originally unassigned, the value of the output gate of unassigned wire η does not depend on the particular assignment, and thus the given assignment is u_η is consistent with the gate checkers of the output gates.

Consider now some internal gate ι . Assume that it has t_0 incoming edges η_j that start from some gates ι_j . By recursive construction, the gate checkers of ι_j have assigned an assignment to the wires η_j . (This is true since every gate implements a function, and therefore the corresponding gate checker must only accept for one possible value of the output wire. In the case this is the input wire, the assignment was done in the previous paragraph.) However, since the gate checker for ι might not be conscientious, the gate checker $P = P(\iota)$ of ι might not have assigned any value to these wires (that is, the corresponding coefficients a_i and b_i are zero). In this case, given the values of all other wires that have assignments, the output of ι does not depend on the values of the unassigned input wires. We then can assign arbitrary values to the unset coefficients a_i and b_i , and in particular we can assign values that are consistent with the output values of all ι_j . This in particular also assigns unequivocal values u_{η_j} to all wires η_j .

The total witness is defined as the concatenation of u_η for all wires. □

Clearly, this lemma can be restated in the language of polynomial circuit checker. For a set \mathcal{P} of polynomials, let $\text{span } \mathcal{P}$ be their span (that is, the set of \mathbb{F} -linear combinations). In particular, \mathbf{v} is in the span of vectors \mathbf{v}_i , $\mathbf{v} = \sum a_i \mathbf{v}_i$, iff the corresponding interpolated polynomial $v(X)$ is in the span of polynomials $v_i(X)$, that is, $v(X) = \sum a_i v_i(X)$. Thus, in Lem. 2, the corresponding requirements will be (i) $z(X) \mid (v_0(X) + v(X)) \cdot (w_0(X) + w(X))$, (ii) $v(X) \in \text{span}\{v_i(X) : i \in [m]\}$, and (iii) $w(X) \in \text{span}\{w_i(X) : i \in [m]\}$.

For the next lemma and the main theorem (Thm. 8) we need the standard *multilinear universal hash function family* [GMS74, CW79, WC81] $\text{ML} : \mathbb{F}^{d+3} \times \mathbb{F}^{d+2} \rightarrow \mathbb{F}$, where for $\mathbf{k} = (k_0, \dots, k_{d+1})$ and $\mathbf{x} = (x_0, \dots, x_{d+1})$, $\text{ML}_{(\mathbf{k}^*, \mathbf{k})}(\mathbf{x}) = \sum_{i=0}^{d+1} k_i x_i + k^*$.

Lemma 3. *Let m and d be two positive integers. For $\text{ML} : \mathbb{F}^{d+3} \times \mathbb{F}^{d+2} \rightarrow \mathbb{F}$ defined as above, define $\text{ML}_{\mathbf{k}}^-(\mathbf{x}) := \text{ML}_{(0, \mathbf{k})}(\mathbf{x}) - \text{ML}_{(0, \mathbf{k})}(0) = \sum_{i=0}^{d+1} k_i x_i = \mathbf{k} \cdot \mathbf{x}$. Let $\mathcal{P} = (\mathbf{v}_1, \dots, \mathbf{v}_{m+1}) \subset \mathbb{F}^{d+2}$ with $\mathbf{v}_i = (v_{i0}, \dots, v_{i, d+1})$ be such that $v_{i, d+1} = 0$ for all i . Define*

$$\mathbb{V}(\mathcal{P}) := \{\mathbf{k} \in \mathbb{F}^{d+1} : \text{ML}_{\mathbf{k}}^-(\mathbf{v}) = 0 \text{ for all } \mathbf{v} \in \mathcal{P}\} = \{\mathbf{k} \in \mathbb{F}^{d+1} : \mathbf{k} \cdot \mathbf{v} = 0 \text{ for all } \mathbf{v} \in \mathcal{P}\} .$$

Then for any $\mathbf{x}_1 \in \mathbb{F}^{d+2} \setminus \text{span } \mathcal{P}$, $\mathbf{x}_2 \in \mathbb{F}^{d+2} \setminus \text{span}(\mathcal{P} \cup \{\mathbf{x}_1\})$, and any $y_1, y_2 \in \mathbb{F}$,

$$\Pr_{\mathbf{k} \leftarrow \mathbb{V}(\mathcal{P})} [\text{ML}_{\mathbf{k}}^-(\mathbf{x}_2) = y_2 \mid \text{ML}_{\mathbf{k}}^-(\mathbf{x}_1) = y_1] = \frac{1}{q} .$$

Proof. The key \mathbf{k} is drawn completely random, except that it has to satisfy $\mathbf{k} \cdot \mathbf{v} = 0$ for $\mathbf{v} \in \mathcal{P}$. The only other thing which is known about \mathbf{k} is the value $\mathbf{k} \cdot \mathbf{x}_1$. Since $\mathbf{x}_2 \notin \text{span}(\mathcal{P} \cup \{\mathbf{x}_1\})$, the inner product $\mathbf{k} \cdot \mathbf{x}_2$ looks completely random. \square

We are now ready to prove the soundness of the new NIZK argument.

Proof (Of Thm. 8). Only within this proof, we will implicitly use the canonical isomorphism between polynomials $g(X) = \sum_{i=0}^{d+1} g_i X^i$ in $\mathbb{F}[X]^{\leq d+1}$ and their coefficient vectors $\mathbf{g} = (g_0, \dots, g_{d+1})$ from \mathbb{F}^{d+2} . (In most of the current paper, $v(X)$ denotes the interpolated polynomial obtained from $v(r_i) = v_i$. This is not the case in this proof.) For a polynomial $g(X)$, let $\text{cf}(g(X))$ be the coefficient of X^{d+1} in $g(X)$.

SOUNDNESS: assume that there exists an adversary \mathcal{A} that succeeds in breaking the soundness of the argument from Sect. 9. We show how to construct an adversary \mathcal{B} , which interacts with \mathcal{A} and breaks the $(d+1, 2d+3)$ -PDH assumption.

Let $\text{UC}_n : \{0, 1\}^n \rightarrow \{0, 1\}$ be the universal circuit for circuits of size C_{bnd} , where $|C| = n$, which has a polynomial circuit checker $c_{\mathcal{A}}(\text{UC}_n)$ of degree d . Suppose that \mathcal{B} receives a $(d+1, 2d+3)$ -PDH challenge

$$ch := ([1], [\sigma], \dots, [\sigma^d], [\sigma^{d+2}], \dots, [\sigma^{2d+3}]) .$$

\mathcal{B} computes UC_n and associated parameters.

He generates a random $\alpha \leftarrow \mathbb{F}$. He generates β_v, β_w , and γ indirectly in terms of their representations over the power basis $\{\sigma^j\}$, so that he can generate the CRS despite only knowing these values implicitly.

Let $\mathcal{P}_v := \{v_i(X) : i \in [m]\} \cup \{z(X)\}$. To generate β_v , \mathcal{B} generates a random key for ML^- ,

$$\mathbf{K}_v = (K_{v0}, \dots, K_{v, d+1}) \leftarrow \mathbb{V}(\mathcal{P}_v) .$$

Recall that $\text{ML}_{\mathbf{K}_v}^-(\mathbf{f}) = \mathbf{K}_v \cdot \mathbf{f} = \sum_{i=0}^{d+1} K_{vi} f_i$. If \mathbf{x} is the coefficient vector of the polynomial $x(X) \in \mathbb{F}[X]^{\leq d+1}$ and $\mathbf{k}^{rev} = (k_{d+1}, \dots, k_0)$ (the reversal of $\mathbf{k} = (k_0, \dots, k_{d+1})$) is the coefficient vector of the polynomial $k^{rev}(X) = X^{d+1} \cdot k(1/X)$, then clearly $\text{ML}_{\mathbf{k}^{rev}}^-(\mathbf{x}) = \text{cf}(k^{rev}(X)x(X))$.

Denoting $\sigma^* = (\sigma^{d+1}, \sigma^d, \dots, 1)$, $\text{ML}_{\mathbf{K}_v}^-(\sigma^*) = \mathbf{K}_v \cdot \sigma^* = \sum_{i=0}^{d+1} K_{vi} \cdot \sigma^{d+1-i}$. \mathcal{B} implicitly sets

$$\beta_v = \beta_v(\sigma) := \text{ML}_{\mathbf{K}_v}^-(\sigma^*) = \mathbf{K}_v \cdot \sigma^* . \quad (2)$$

(Thus, β_v depends on $c_A(\text{UC}_n)$.) Note that \mathcal{B} cannot create encoding of β_v . We will deal with this issue later. Now, for *any* polynomial $g \in \mathbb{F}[X]^{\leq d+1}$,

$$\begin{aligned} \text{cf}(\beta_v g(\sigma)) &= \text{cf}((\mathbf{K}_v \cdot \sigma^*) \cdot g(\sigma)) = \text{cf}\left(\sum_{i=0}^{d+1} K_{vi} \sigma^{d+1-i}\right) \cdot \left(\sum_{i=0}^{d+1} g_i \sigma^i\right) = \sum_{i=0}^{d+1} K_{vi} g_i = \mathbf{K}_v \cdot \mathbf{g} \\ &= \text{ML}_{\mathbf{K}_v}^-(\mathbf{g}) . \end{aligned}$$

Clearly, $\deg \beta_v(\sigma) \leq d+1$. Assume $p(X) \in \mathcal{P}_v$. Then, $\deg(\beta_v \cdot p(\sigma)) \leq (d+1) + d = 2d+1$. Since $\mathbf{K}_v \in \mathbb{V}(\mathcal{P}_v)$, $\text{cf}(\beta_v p(\sigma)) = \text{ML}_{\mathbf{K}_v}^-(\mathbf{p}) = 0$. Thus, \mathcal{B} can, given $(\llbracket \sigma^i \rrbracket)_{i \in [2d+1] \setminus \{d+1\}} \subset ch$, generate encodings $\llbracket \beta_v p(\sigma) \rrbracket$ for $p(X) \in \mathcal{P}_v$.

The generation of β_w is analogous. Let $\mathcal{P}_w := \{w_i(X) : i \in [m]\} \cup \{z(X)\}$. Set $\beta_w \leftarrow \text{ML}_{\mathbf{K}_w}^-(\sigma^*)$, where $\mathbf{K}_w \leftarrow \mathbb{V}(\mathcal{P}_w)$. Given $(\llbracket \sigma^i \rrbracket)_{i \in [2d+1] \setminus \{d+1\}} \subset ch$, \mathcal{B} generates encodings $\llbracket \beta_w p(\sigma) \rrbracket$ for $p(X) \in \mathcal{P}_w$. Clearly, β_v and β_w generated in this way have appropriately uniform distribution.

The reason why vcrs contains $\llbracket \gamma \rrbracket$, $\llbracket \beta_v \gamma \rrbracket$, and $\llbracket \beta_w \gamma \rrbracket$ instead of just $\llbracket \beta_v \rrbracket$ and $\llbracket \beta_w \rrbracket$ is that with high probability $\text{cf}(\mathbf{K}_v \cdot \sigma^*) \neq 0$, and thus $\beta_v = \text{ML}_{\mathbf{K}_v}^-(\sigma^*(X))$ likely has a non-zero coefficient for σ^{d+1} , making it impossible for \mathcal{B} to generate an encoding of it. To alleviate this problem, we let \mathcal{B} to generate γ^* uniformly from \mathbb{F} . He then implicitly sets $\llbracket \gamma \rrbracket \leftarrow \gamma^* \llbracket \sigma^{d+2} \rrbracket$. Clearly, γ has uniform distribution over \mathbb{F} . Since $d+2 \in [2d+1] \setminus \{d+1\}$, \mathcal{B} can generate an encoding of γ from $\llbracket \sigma^{d+2} \rrbracket \in ch$. Moreover \mathcal{B} can generate encodings of $\beta_v \gamma = \text{ML}_{\mathbf{K}_v}^-(\sigma^*) \cdot \gamma^* \sigma^{d+2}$ and $\beta_w \gamma = \text{ML}_{\mathbf{K}_w}^-(\sigma^*) \cdot \gamma^* \sigma^{d+2}$, since these two terms have a zero coefficient for σ^{d+1} and, due to $\deg \sigma^* \leq d+1$ are of degree at most $\deg \sigma^* + (d+2) \leq (d+1) + (d+2) = 2d+3$.

Since $(\llbracket \sigma^i \rrbracket)_{i \in [d+1] \setminus \{d+1\}} \in ch$, \mathcal{B} can compute the rest of the crs and vcrs as in Sect. 9. \mathcal{B} provides crs and vcrs to \mathcal{A} .

Assume that $\mathcal{A}(\text{crs})$ generates an argument π of a false statement C that passes the verification. From the verification equations and the fact that the image of the encoding is verifiable, the argument must have the form $\pi = (\llbracket V \rrbracket, \llbracket W \rrbracket, \llbracket H \rrbracket, \llbracket \alpha V \rrbracket, \llbracket \alpha W \rrbracket, \llbracket \alpha H \rrbracket, \pi_y = \llbracket \beta_v V + \beta_w W \rrbracket)$.

The CRS crs received by \mathcal{A} is a valid input $(c; z)$ of the d -PKE assumption: it consists of $c = (\{\llbracket \sigma^j \rrbracket, \llbracket \alpha \sigma^j \rrbracket\}_{j \in [0, d]})$ and $z = (\text{crs}, \text{vcrs}) \setminus c$, where the auxiliary information z is independent of α . By using the extractor $\mathcal{E}_{\mathcal{A}}$ of the d -PKE assumption, since $\mathcal{A}(c; z)$ produces $(\llbracket V \rrbracket, \llbracket \alpha V \rrbracket)$, \mathcal{B} obtains a degree- d polynomial $v^*(X)$ (with $v^*(X) = v(X) + r_v z(X)$ if the prover is honest), such that $V = v^*(\sigma)$. Similarly, he obtains degree- d polynomials $w^*(X)$ (with $w^*(X) = w(X) + r_w z(X)$ if the prover is honest) and $h^*(X)$ (with $h^*(X) = h(X) + r_v \cdot (w_0(X) + w(X)) + r_w \cdot (v_0(X) + v(X)) + r_v r_w z(X)$ if the prover is honest). Since π verifies, we have that

- $(v_0(\sigma) + v^*(\sigma)) \cdot (w_0(\sigma) + w^*(\sigma)) = h^*(\sigma) \cdot z(\sigma)$, and
- the last term of the proof properly encodes $\beta_v v(\sigma) + \beta_w w(\sigma)$.

Since π is an argument for a false statement, Lem. 2 (more precisely, its polynomial reinterpretation) implies that at least one of the following two cases must hold:

- Case 1: $(v_0(X) + v^*(X)) \cdot (w_0(X) + w^*(X)) \neq h^*(X) \cdot z(X)$.
- Case 2: Either $v^*(X) \notin \text{span}(\{v_i(X) : i \in [m]\} \cup \{z(X)\})$ or $w^*(X) \notin \text{span}(\{w_i(X) : i \in [m]\} \cup \{z(X)\})$.

We recall that $z(X)$ is a mapping of the all-zero vector $(0, \dots, 0)$, and thus when applying Lem. 2, we can omit mentioning $z(X)$. We now show that in either case, \mathcal{B} can solve the $(d+1, 2d+3)$ -PDH problem.

Suppose that Case 1 holds. Then

$$f(X) := (v_0(X) + v^*(X)) \cdot (w_0(X) + w^*(X)) - h^*(X) \cdot z(X)$$

is a non-zero polynomial of degree $\leq 2d$ having σ as a root. \mathcal{B} uses an efficient polynomial factorization algorithm to find $\leq 2d$ roots σ_i^* of $f(X)$ over \mathbb{F} , and then finds by exhaustive search an index i such that $\llbracket \sigma_i^* \rrbracket = \llbracket \sigma \rrbracket$. Given σ , he can also compute $\llbracket \sigma^{d+1} \rrbracket$. Thus, he has broken the $(d+1, 2d+3)$ -PDH assumption.⁵

⁵ We remark that [GGPR12] used a different proof technique here that did not require the use of polynomial factorization, but resulted in the $(\lambda+1, 2\lambda)$ -PDH assumption for $\lambda \geq 2d-1$. We could use the same technique, but we think that weakening the assumption is worth the extra step in reduction.

Suppose that Case 2 holds. W.l.o.g., suppose that $v^*(X)$ cannot be expressed as a linear combination of $\mathcal{P}_v := \{v_i(X) : i \in [m]\} \cup \{z(X)\}$. (We also note that $z(X)$ is an interpolation of the all-zero vector. The only information that \mathcal{E}_A has about K_v is

- (i) that $\mathbf{K}_v \in \mathbb{V}(\mathcal{P}_v)$ and thus $\text{ML}_{\mathbf{K}_v}^-(p) = 0$ for $p(X) \in \mathcal{P}_v$, and
- (ii) the value $\text{ML}_{\mathbf{K}_v}^-(\sigma^*) = \beta_v$.

By Lemma 3, since $v^*(X) \notin \text{span } \mathcal{P}_v$, $v^*(X) \neq \sigma^*(X)$, and $\mathbf{K}_v \in \mathbb{V}(\mathcal{P}_v)$, the value $\text{ML}_{\mathbf{K}_v}^-(\mathbf{v}^*) = \text{cf}(\beta_v v^*(\sigma))$ is uniformly random. Thus, the coefficient of σ^{d+1} in $\beta_v v^*(\sigma) + \beta_w w^*(\sigma)$ is uniformly random, regardless of the choice of $w^*(X)$. With probability $1 - 1/\mathbb{F}$, this coefficient is non-zero. Assume now that this is the case. If it is non-zero, due to Eq. (2) (and the choice of β_w), π_y encodes an element $y(\sigma) := \beta_v v^*(\sigma) + \beta_w w^*(\sigma)$. Since $\deg \beta_v \leq d+1$, $\deg y(\sigma) = \deg \beta_v + d \leq (d+1) + d = 2d+1 \leq 2d+3$, and, with probability $1 - (d-1)/q$, $y(\sigma)$ has a non-zero coefficient for σ^{d+1} . Since all coefficients of y are known to \mathcal{B} (for example, $\beta_v g(\sigma) = (\sum_{i=0}^{d+1} K_{vi} \sigma^{d+1-i}) \cdot (\sum_{i=0}^{d+1} g_i \sigma^i)$, and thus \mathcal{B} can compute the coefficients of $\beta_v v^*(\sigma)$ from \mathbf{K}_v and \mathbf{v}^*), he can subtract off encodings of multiples of the other powers of σ (given in the $(d+1, 2d+3)$ -PDH instance) to obtain an encoding of a non-zero multiple of σ^{d+1} , from which \mathcal{B} can obtain an encoding of σ^{d+1} , solving the $(d+1, 2d+3)$ -PDH problem.

ARGUMENT OF KNOWLEDGE: The argument of knowledge property follows from the extraction of the polynomials $v^*(X)$, $w^*(X)$ and $h^*(X)$, as described above, and Lemma 2. \square

Acknowledgements. We would like to thank Andris Ambainis, Aleksandrs Belovs and Vitaly Skachek for useful discussions. The author was supported by the Estonian Research Council, and European Union through the European Regional Development Fund.

References

- Amb10. Andris Ambainis. New Developments in Quantum Algorithms. In Petr Hlinený and Antonín Kucera, editors, *MFCS 2010*, volume 6281 of *LNCS*, pages 1–11, Brno, Czech Republic, August 23–27, 2010. Springer, Heidelberg. 1, 2
- BCCT12. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, And Back Again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349, Cambridge, MA, USA, January 8–10, 2012. ACM Press. 1
- BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data. In ?, editor, *STOC 2013*, pages ?–?, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. 1, 8
- Bel12a. Aleksandrs Belovs. Span-Program-Based Quantum Algorithm for the Rank Problem. Technical Report arXiv:1103.0842, arXiv.org, March 4, 2012. Available from <http://arxiv.org/abs/1103.0842>. 2
- Bel12b. Aleksandrs Belovs. Span Programs for Functions with Constant-Sized 1-Certificates. In Howard J. Karloff and Toniann Pitassi, editors, *STOC 2012*, pages 77–84, New York, NY, USA, May 19–22, 2012. ACM Press. 2
- BF01. Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, Santa Barbara, USA, August 19–23, 2001. Springer, Heidelberg. 9
- BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-Interactive Zero-Knowledge and Its Applications. In *STOC 1988*, pages 103–112, Chicago, Illinois, USA, May 2–4, 1988. ACM Press. 1
- BGP95. Amos Beimel, Anna Gál, and Mike Paterson. Lower Bounds for Monotone Span Programs. In *FOCS 1995*, pages 674–681, Milwaukee, Wisconsin, USA, October 23–25 1995. IEEE. 2
- BW03. Amos Beimel and Enav Weinreb. Separating the Power of Monotone Span Programs over Different Fields. In *FOCS 2003*, pages 428–437, Cambridge, MA, USA, October, 11–14 2003. IEEE, IEEE Computer Society Press. 2
- CF02. Ronald Cramer and Serge Fehr. Optimal Black-Box Secret Sharing over Arbitrary Abelian Groups. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 272–287, Santa Barbara, USA, August 18–22, 2002. Springer, Heidelberg. 2

- CLZ12. Rafik Chaabouni, Helger Lipmaa, and Bingsheng Zhang. A Non-Interactive Range Proof with Constant Communication. In Angelos Keromytis, editor, *FC 2012*, volume 7397 of *LNCS*, pages 179–199, Bonaire, The Netherlands, February 27–March 2, 2012. Springer, Heidelberg. 1
- CW79. L. Lawrence Carter and Mark N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979. 1, 10
- DL08. Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP Proofs from an Extractability Assumption. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Computability in Europe, CIE 2008*, volume 5028 of *LNCS*, pages 175–185, Athens, Greece, June 15–20, 2008. Springer, Heidelberg. 1
- DN00. Cynthia Dwork and Moni Naor. Zaps and Their Applications. In *FOCS 2000*, pages 283–293, Redondo Beach, California, USA, November 12–14, 2000. IEEE Computer Society Press. 1
- Elk11. Michael Elkin. An Improved Construction of Progression-Free Sets. *Israeli Journal of Mathematics*, 184:93–128, 2011. 1
- ET36. Paul Erdős and Paul Turán. On Some Sequences of Integers. *Journal of the London Mathematical Society*, 11(4):261–263, 1936. 1
- FLZ12. Prastudy Fauzi, Helger Lipmaa, and Bingsheng Zhang. New Non-Interactive Zero-Knowledge Subset Sum, Decision Knapsack and Range Arguments. Technical Report 2012/548, International Association for Cryptologic Research, September 19, 2012. Available at <http://eprint.iacr.org/2012/548>. 1
- Gál01. Anna Gál. A Characterization of Span Program Size and Improved Lower Bounds for Monotone Span Programs. *Computational Complexity*, 10(4):277–296, 2001. 1, 2
- GG03. Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2 edition, July 3, 2003. 1, D, 1, 2, 3, D, 4, 5
- GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482, Santa Barbara, California, USA, August 15–19, 2010. Springer, Heidelberg. 1
- GGPR12. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. Technical Report 2012/215, International Association for Cryptologic Research, April 19, 2012. Available at <http://eprint.iacr.org/2012/215>, last retrieved version from June 18, 2012. 1, 1, 1, 2, 4, 5, 6.1, 6.1, 8, 7, 2, 7, 8, 8, 9, 8, 8, 3, 10, 5
- GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume ? of *LNCS*, pages ?–?, Athens, Greece, April 26–30, 2013. Springer, Heidelberg. 1
- GMS74. Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes Which Detect Deception. *Bell System Technical Journal*, 53:405–424, 1974. 1, 10
- Gro10. Jens Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg. 1, 8
- GS08. Jens Groth and Amit Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. In Nigel Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg. 1
- GW11. Craig Gentry and Daniel Wichs. Separating Succinct Non-Interactive Arguments from All Falsifiable Assumptions. In Salil Vadhan, editor, *STOC 2011*, pages 99–108, San Jose, California, USA, June 6–8, 2011. ACM Press. 1
- HKP84. H. James Hoover, Maria M. Klawe, and Nicholas Pippenger. Bounding Fan-out in Logical Networks. *Journal of the ACM*, 31(1):13–18, 1984. 1, 4, B
- Jou00. Antoine Joux. A One-Round Protocol for Tripartite Diffie-Hellman. In Wieb Bosma, editor, *The Algorithmic Number Theory Symposium*, volume 1838 of *LNCS*, pages 385–394, Leiden, The Netherlands, 2–7 June 2000. Springer, Heidelberg. ISBN 3-540-67695-3. 9
- Juk12. Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 2 of *Algorithms and Combinatorics*. Springer, January 2012. 2
- KW93. Mauricio Karchmer and Avi Wigderson. On Span Programs. In *Structure in Complexity Theory Conference 1993*, pages 102–111, San Diego, CA, USA, May 18–21, 1993. IEEE Computer Society Press. 2, 2
- Lip12. Helger Lipmaa. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189, Taormina, Italy, March 18–21, 2012. Springer, Heidelberg. 1, 8
- LZ12. Helger Lipmaa and Bingsheng Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In Ivan Visconti and Roberto De Prisco, editors, *SCN 2012*, volume 7485 of *LNCS*, pages 477–502, Amalfi, Italy, September 5–7, 2012. Springer, Heidelberg. 1

- Mic94. Silvio Micali. CS Proofs. In Shafi Goldwasser, editor, *FOCS 1994*, pages 436–453, Santa Fe, New Mexico,, USA, November 20–22, 1994. IEEE, IEEE Computer Society Press. 1
- Pip80. Nicholas Pippenger. On the Evaluation of Powers and Monomials. *SIAM Journal of Computing*, 9(2):230–250, 1980. 8
- Rei11. Ben Reichardt. Reflections for Quantum Query Algorithms. In Dana Randall, editor, *SODA 2011*, pages 560–569, San Francisco, California, USA, January 23–25, 2011. SIAM. 1, 2
- Rot06. Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. 6.1, A
- RS60. Irving S. Reed and Gustave Solomon. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. 1, 6.1, A
- RS08. Ben W. Reichardt and Robert Spalek. Span-Program-Based Quantum Algorithm for Evaluating Formulas. In Richard E. Ladner and Cynthia Dwork, editors, *STOC 2008*, pages 103–112, Victoria, BC, Canada, May 17–20 2008. ACM Press. 2
- SOK00. Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems Based on Pairing. In *SCIS 2000*, Okinawa, Japan, 2000. 9
- Val76. Leslie G. Valiant. Universal Circuits (Preliminary Report). In *STOC 1976*, pages 196–203, Hershey, Pennsylvania, USA, May 3–5, 1976. ACM. 1, 9
- WC81. Mark N. Wegman and Larry Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. 1, 10

A Preliminaries: Coding Theory

Consider a finite field \mathbb{F} of cardinality q . A linear code C of length n and rank k is a linear subspace with dimension k of the vector space \mathbb{F}^n . The vectors of C are called codewords. The size of a code is the number of codewords, q^k . The distance between two codewords is the Hamming distance between them. The distance d of a linear code is the minimum distance between distinct codewords. A linear code of length n , dimension k , and distance d is called an $[n, k, d]$ code.

The code c may be represented as the span of a minimal set (basis) of codewords. The generating matrix G of a linear code consists of the basis vectors. If $G = (I_k|A)$, where I_k is the $k \times k$ identity matrix and A is some $k \times (n - k)$ matrix, then G is said to be in the standard form. In this case, the code is said to be systematic.

Let \mathbb{F} be a finite field. Pick distinct elements $\{\alpha_1, \dots, \alpha_n\}$ (also called *evaluation points*) from \mathbb{F} and choose n and k such that $k \leq n \leq q$. We define an encoding function for Reed-Solomon code $RS : \mathbb{F}^k \rightarrow \mathbb{F}^n$ as follows. A message $\mathbf{m} = (m_0, \dots, m_{k-1})$ with $m_i \in \mathbb{F}$ is mapped to a degree $k - 1$ polynomial, $\mathbf{m} \mapsto f_{\mathbf{m}}(X)$, where $f_{\mathbf{m}}(X) = \sum_{i=0}^{k-1} m_i X^i$. The encoding of \mathbf{m} is the evaluation of $f_{\mathbf{m}}$ at all the α_i 's, $RS(\mathbf{m}) = (f_{\mathbf{m}}(\alpha_1), \dots, f_{\mathbf{m}}(\alpha_n))$. We call this image Reed-Solomon code or RS code. A common special case is $n = q - 1$ with the set of evaluation points being $\mathbb{F}^* := \mathbb{F} \setminus \{0\}$.

The singleton bound states that for any $[n, k, d]_q$ code, $k \leq n - d + 1$. A code is a maximum distance separable (MDS) code when it meets the singleton bound, that is, $k = n - d + 1$. Every binary systematic MDS code is either $\{0\}$, parity-check, repetition, or \mathbb{F}^d (see [Rot06, Chapter 11]). The Reed-Solomon codes [RS60] meet the singleton bound, that is, satisfy $k = n - d + 1$ (but have the unfortunate property that $q \geq n$; this property is fine for our application).

B Proof of Thm. 1 (Parameters of Aggregate Gate Checker)

Proof. As in [HKP84], we reduce the fan-out of every gate i with $\deg^+(i) > 2$ by connecting the output wire of i to an inverted binary tree of new $\deg^+(i) - 2$ fork gates.

Assume that the fan-in of all gates is bounded by t_0 , the fan-out is bounded by t , and that n_0/n_1 is the number of inputs/outputs of the circuit; here $t_0 = t = 2$ and $n_1 = 1$. We use the technique of Hoover, Klawe

and Pippenger [HKP84] to upper bound the number of added fork gates by

$$\begin{aligned}
\sum_{\deg^+(i) > t} (\lceil (\deg^+(i) - 1)t^* \rceil - 1) &\leq \sum_{\deg^+(i) > 0} ((\deg^+(i) - 1)t^* - t^*) \\
&\leq \sum_{\deg^+(i) > 0} (\deg^+(i) - 1)t^* - t^* \\
&= t^* \cdot \left(\sum_{\deg^+(i) > 0} \deg^+(i) - \sum_{\deg^+(i) > 0} 1 \right) - t^* \\
&\leq t^* \cdot (t_0 \cdot (n - n_0) - (n - n_1)) - t^* \\
&= (t_0 - 1)t^* \cdot n + (n_1 - 1 - t_0 \cdot n_0)t^* .
\end{aligned}$$

Thus, this operation adds

$$(t_0 - 1)t^* \cdot n + (n_1 - 1 - t_0 \cdot n_0)t^* = t^* \cdot n - 2t^*n_0$$

gates to the circuit. (The last equality is true only when $t_0 = 2$ and $n_1 = 1$, but we assume that the circuit has fan-in ≤ 2 anyway.) Thus, the total number of the gates in C_{bnd} will be $(1 + t^*) \cdot n - 2t^*n_0$.

Recall from Sect. 3 that $SP(c_{\check{Y}}^t)$ has size $m_{c_{\check{Y}}^t} = 2t + 2$ and dimension $d_{c_{\check{Y}}^t} = t + 1$. The size of the span program for the aggregate gate checker function for f_{bnd} is thus at most $m^* \cdot (n - n_0) + m_{c_{\check{Y}}^t} \cdot (n - 2n_0)t^*$, where $m^* = 6$ is an upper bound on the size of the span programs for the individual gates (NAND, AND, OR, XOR, NOT, and fork), thus at most

$$6 \cdot (n - n_0) + 2(t + 1) \cdot (n - 2n_0)t^* = (8 + 4t^*) \cdot n - (10 + 8t^*) \cdot n_0$$

by using $SP(c_{\check{\Lambda}})$ and $SP(c_{\check{Y}}^t)$ from Sect. 3. Analogously, the dimension of the span program is at most

$$3 \cdot (n - n_0) + (t + 1) \cdot (n - 2n_0)t^* = (4 + 2t^*) \cdot n - (5 + 4t^*) \cdot n_0 .$$

Clearly, each vector in the span program for **agc** (aside from the target vector) has only a small constant number of non-zero coefficients, since the vectors in the span program for **agc** are inherited from the small span programs for the individual gates of C_{bnd} . More precisely, the number of non-zero entries of the aggregate gate checker is

$$\begin{aligned}
&\text{supp}(SP(c_{\check{\Lambda}})) \cdot (n - n_0) + \text{supp}(SP(c_{\check{Y}}^t)) \cdot (n - 2n_0)t^* \\
&= 7 \cdot (n - n_0) + (2t + 2) \cdot (n - 2n_0)t^* = (9 + 4t^*) \cdot n - (11 + 8t^*) \cdot n_0 .
\end{aligned}$$

□

C Proof of Thm. 4 (Parameters of Aggregate Wire Checker)

Proof. Let $|C_{\text{bnd}}|$ be the circuit after we have applied Thm. 1 to it. Let again $t^* = 1/(t - 1)$. First, note that in the original circuit, for every gate $i \in [n - 1]$, the wire checker corresponding to its output wire has $D(\eta) = \deg^+(i) + 1$. If $\deg^+(i) > t$, Thm. 1 builds a t -ary inverse tree on top of the i th node. This tree adds $\lceil (\deg^+(i) - 1)t^* \rceil - 1$ new vertices. Let $\Gamma(i)$ be the set of vertices induced by the original vertex i (including i itself), then $|\Gamma(i)| = \lceil (\deg^+(i) - 1)t^* \rceil$. Every node of $\Gamma(i)$ corresponds to one wire checker which is induced by the vertex i in circuit C_{bnd} .

To compute size and dimension of the part of the aggregate wire checker of the gates induced by a concrete gate i , we note that the corresponding D values have in total

1. $\deg^+(i) + 1$ original vectors (one per each wire labelled by x_i and z in Fig. 4),
2. $2 \cdot \lceil (\deg^+(i) - 1)t^* - 1 \rceil$ new vectors (one per each wire labelled by x or y),

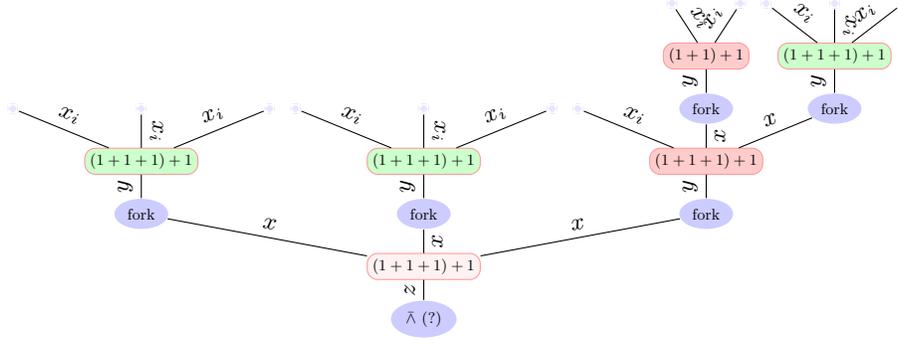


Fig. 4. Added inverted tree with $\deg^+(i) = 12$ and $t = 3$

all together $\sum_{j \in \Gamma(i)} D_j \leq (\deg^+(i) + 1) + 2 \cdot (\lceil (\deg^+(i) - 1)t^* \rceil - 1)$ vectors. When we sum over all nodes of the original circuit and over their induced nodes from all $\Gamma(i)$'s, we get

$$\sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} D_j \leq \sum_{i=1}^{n-1} (\deg^+(i) + 1) + 2 \cdot \sum_{\deg^+(i) > t} (\lceil (\deg^+(i) - 1)t^* \rceil - 1).$$

Now, $\sum_{i=1}^{n-1} (\deg^+(i) + 1) = \sum_{i=n_0+1}^n \deg^-(i) + n - 1 \leq 2(n - n_0) + n - 1 = 3n - 2n_0 - 1$. On the other hand, as in the proof of Thm. 1,

$$\begin{aligned} \sum_{\deg^+(i) > t} (\lceil (\deg^+(i) - 1)t^* \rceil - 1) &\leq \left(\sum_{i=1}^{n-1} (\deg^+(i) - 1)t^* - t^* \right) = t^* \cdot \left(\sum_{i=1}^{n-1} \deg^+(i) - \sum_{i=1}^{n-1} 1 - 1 \right) \\ &= t^* \cdot (2(n - n_0) - (n - 1) - 1) = t^*n - 2t^*n_0. \end{aligned}$$

Thus,

$$\sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} D_j \leq 3n - 2n_0 - 1 + 2(t^*n - 2t^*n_0) = (3 + 2t^*)n - 2(1 + 2t^*)n_0 - 1.$$

The size of the aggregate wire checker is upperbounded by

$$2 \sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} D_j \leq (6 + 4t^*)n - (4 + 8t^*)n_0 - 4.$$

This value is minimal for large t , and maximal when t is small. If $t = 3$, then $\text{size } Q^{\text{awc}} \leq 8n - 8n_0 - 4$.

Similarly,

$$\text{sdim } Q^{\text{awc}} = \sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} (2D_j - 1) \leq (6 + 4t^*)n - (4 + 8t^*)n_0 - 4.$$

(Here, as in the case of degree we have omitted lower order terms.) This value is minimal when t is large. If $t = 3$, then $\text{sdim } Q^{\text{awc}} \leq 8n - 8n_0 - 4$.

Degree:

$$\sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} D_j \leq (3 + 2t^*)n - (2 + 4t^*)n_0 - 1.$$

This value is minimal when t is large. If $t = 3$, then $\text{sdeg } Q^{\text{awc}} \leq 4n - 4n_0 - 2$.

We also need to compute the value $\sum_{j \in \Gamma(i)} D_j^2$. This value is maximized if all “large” D values are concentrated in as few individual span programs as possible. That is,

$$\sum_{j \in \Gamma(i)} D_j^2 \leq (t+1)^2 \lceil (\deg^+(i) - 1)t^* \rceil .$$

Thus,

$$\sum_{i=1}^{n-1} \sum_{j \in \Gamma(i)} D_j^2 \leq (t+1)^2 \sum_{i=1}^{n-1} \lceil (\deg^+(i) - 1)t^* \rceil .$$

But

$$\begin{aligned} \sum_{i=1}^{n-1} \lceil (\deg^+(i) - 1)t^* \rceil &= (n-1) + \sum_{i=1}^{n-1} (\lceil (\deg^+(i) - 1)t^* \rceil - 1) \\ &\leq (n-1) + t^*n - 2t^*n_0 = (1+t^*)n - 2t^*n_0 - 1 . \end{aligned}$$

Thus, the support is upperbounded by

$$4 \sum_i \sum_{j \in \Gamma(i)} D_j^2 \leq 4(t+1)^2 ((1+t^*)n - 2t^*n_0 - 1) .$$

This value is minimized (if n is large) when $t = 2$, then the support is upperbounded by $72n - 72n_0 - 36$. \square

D Preliminaries: Computer Algebra

The following results are more or less directly lifted from [GG03]. We say that a (commutative) ring R supports the FFT if R has a primitive 2^k th root of unity for any $k \in \mathbb{N}$.

Fact 1 (Fast polynomial multiplication, Thm. 8.18 from [GG03]) *Let R be a ring that supports the FFT, and $n = 2^k$ for some $k \in \mathbb{N}$. Let $\omega \in R$ be a primitive n th root of unity. Then convolution in $R[X]/(X^n - 1)$ and multiplication of polynomials $f, g \in R[X]$ with $\deg(fg) < n$ can be performed using $3n \log n$ additions in R , $\frac{3}{2}n \log n + n - 2$ multiplications by powers of ω , n multiplications in R , and n divisions by n , in total $\frac{9}{2}n \log n + O(n)$ arithmetic operations. In particular, polynomials in $R[X]$ of degree less than n can be multiplied with $18n \log n + O(n)$ operations in R .*

Fact 2 (Fast polynomial multiplication in any ring, Thm. 8.23 from [GG03]) *Over any commutative ring R , polynomials of degree less than n can be multiplied using at most $(18 + 72 \log_3 2)n \log n \log \log n + O(n \log n)$ or $63.43 \cdot n \log n \log \log n + O(n \log n)$ arithmetic operations in R .*

In what follows, assume that polynomials in $R[X]$ of degree less than n can be multiplied using at most $M(n)$ operations in R .

Fact 3 (Fast polynomial division with remainder, Thm. 9.6 from [GG03]) *Let D be a ring (commutative, with 1). Division with remainder of a polynomial $a \in D[X]$ of degree $n + m$ by a monic polynomial $b \in D[X]$ of degree n , where $n \geq m \in \mathbb{N}$ can be done using $4M(m) + M(n) + O(n)$ ring operations.*

We recall from [GG03], that one can do this in two steps. First, given some polynomial f , compute — by using Newton iteration (see Thm. 9.4 of [GG03]) — the inverse $\text{Inv}(f, \ell)$ of f modulo 2^ℓ . If $\ell = 2^r$ is a power of two, then this requires at most $3M(\ell) + \ell = O(M(\ell))$ arithmetic operations in D . Second, let

$f^{rev}(X) := X^{\deg f} \cdot f(1/X)$ for any polynomial $f(X)$. Given a and b for monic b , compute q, r such that $a = qb + r$ and $\deg r < \deg b$, as follows:

- 1 **if** $\deg a < \deg b$ **then return** $(q, r) = (0, a)$;
- 2 $m \leftarrow \deg a - \deg b$;
- 3 $c(X) \leftarrow \text{Inv}(b^{rev}(X), m + 1)$;
- 4 $q^*(X) \leftarrow a^{rev}(X) \cdot c(X) \pmod{X^{m+1}}$;
- 5 $q(X) \leftarrow q^{rev}(X)$;
- 6 $r(X) \leftarrow a(X) - b(X)q(X)$;
- 7 **return** (q, r) ;

Fact 4 (Fast multipoint evaluation, Cor. 10.8 from [GG03]) *Evaluation of a polynomial in $R[X]$ of degree less than n at n points in R can be performed using at most $(\frac{11}{2}M(n) + O(n)) \log n$ or $O(M(n) \log n)$ operations in R .*

Fact 5 (Fast interpolation, Cor. 10.12 from [GG03]) *Polynomial interpolation over a (commutative) ring R can be solved by using at most $(\frac{13}{2}M(n) + O(n)) \log n$ or $O(M(n) \log n)$ operations in R .*