The GKR Protocol Revisited: Nearly Optimal Prover-Complexity For Polynomial-Time Wiring Algorithms and For Primality Testing in $n^{\frac{1}{2}+o(1)}$ Rounds

Erich L. Kaltofen
Dept. of Mathematics, NCSU and Dept. of Computer Science, Duke University
Raleigh, Durham, North Carolina, USA

ABSTRACT

The proof-of-work interactive protocol by Shafi Goldwasser, Yael T. Kalai and Guy N. Rothblum (GKR) [STOC 2008, JACM 2015] certifies the execution of an algorithm via the evaluation of a corresponding boolean or arithmetic circuit whose structure is known to the verifier by circuit wiring algorithms that define the uniformity of the circuit. Here we study protocols whose prover time- and space-complexities are within a poly-logarithmic factor of the time- and space-complexity of the algorithm; we call those protocols 'prover-nearly-optimal.' We show that the uniformity assumptions can be relaxed from LOGSPACE to polynomial-time in the bit-lengths of the labels which enumerate the nodes in the circuit. Our protocol applies GKR recursively to the arising sumcheck problems on each level of the circuit whose values are verified, and deploys any of the prover-nearly-optimal versions of GKR on the constructed sorting/prefix circuits with log-depth wiring functions.

The verifier time-complexity of GKR grows linearly in the depth of the circuit. For deep circuits such as the Miller-Rabin integer primality test of an n-bit integer, the large number of rounds may interfere with soundness guarantees after the application of the Fiat-Shamir heuristic. We re-arrange the circuit evaluation problem by the baby-steps/giant-steps method to achieve a depth of $n^{1/2+o(1)}$, at prover cost $n^{2+o(1)}$ bit complexity and communication and verifier cost $n^{3/2+o(1)}$.

CCS CONCEPTS

• **Theory of computation → Cryptographic protocols**.

KEYWORDS

cloud computing; primality testing; proof-of-work certificate;

ACM Reference Format:
Erich L. Kaltofen. 2022. The GKR Protocol Revisited: Nearly Optimal Prover-Complexity For Polynomial-Time Wiring Algorithms and For Primality Testing in $\mathbf{n}^{\frac{1}{2}+\mathbf{o(1)}}$ Rounds . In *Proceedings of the 47th ACM Symposium on Symbolic and Algebraic Computation (ISSAC '22), July 4–7, 2022, Lille, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3476446.3536183

ISSAC '22, July 4–7, 2022, Villeneuve-d'Ascq, France.

ACM ISBN 978-1-4503-8688-3/22/07…$15.00
https://doi.org/10.1145/3476446.3536183

1 INTRODUCTION

The delegation of computation to a compute service possibly in the Internet cloud today is a universal approach for carrying out massive computations. A landmark example is the determination that a maximum of 20 face moves are required to solve Rubik's Cube, which was done in a Google data center https://www.cube20.org/. The interactive proving methodology and the breakthrough PCP Theorem allow a verifier with moderate compute power to check the computation of the prover. The 2008 GKR protocol by Shafi Goldwasser, Yael T. Kalai and Guy N. Rothblum [15, 16] reduces the number of rounds to within a poly-logarithmic factor of the depth of the boolean circuit that can be associated with the algorithm for the computation. The bare-bones version of GKR protocol can be realized for a prover that runs within a poly-logarithmic factor of the time-complexity of the computation—we call such a protocol *prover-nearly-optimal*—which is probabilistically checked by the verifier in time circuit-depth × (log circuit-size)$^{O(1)}$ + input-size × (loglog circuit-size)$^{O(1)}$ [7]. More recently, protocols have been proposed that are zero-knowledge and whose prover-time-complexity is within a constant factor of the size of the circuit that represents the computation on the given input; see [26, 27].

The GKR proof-of-work protocol [16] works in two stages. It first takes an algorithm represented by a Turing machine and an input and on the prover side constructs a boolean circuit of poly-logarithmic depth that represents the Turing machine execution on the given input. It is assumed that the Turing machine on an input of n symbols from a fixed alphabet will hold $O(\log n)$ symbols on its auxiliary tapes. In a second stage, an interactive proof protocol is designed for (fan-in $\leq$ 2) boolean circuits that are constructed by local circuit wiring algorithms which are known to both the prover and the verifier. One assumes that the circuit feeds data from each level to the next and does not skip over levels, which can be achieved by inserting passthru nodes. The circuit has numbers assigned for each node on the ℓ-th Level as values $\in \{0,1\}^{s_\ell}$, where $s_\ell = O(\log \text{circuit-width})$. One constructs a constant number of local circuit wiring algorithms isor$(\ell,\alpha),\ldots,$ src1(ℓ,α), src2(ℓ,α), which compute which arithmetic is done in the node labelled α on Level ℓ and what are the labels on the previous level of the two arguments that feed into the node. The GKR interactive *bare-bones protocol* [15] is implemented for wiring algorithms whose associated boolean circuits have depth $O(\log s_\ell)$, that is $O(\text{loglog circuit-width})$. The verifier during the protocol has to check the evaluation of those boolean circuits on verifier-selected random values in finite extension fields. In place of the wiring algorithms for the Turing machine execution, Goldreich [13] uses the circuit for a universal Turing machine that interprets the Turing machine on the

input. The universal Turing machine circuit is again bare-bones protocol verifiable.

The verifier time-complexity in the GKR bare-bones protocol depends linearly on the depth of the circuit whose evaluation is checked. Therefore, the verifier performs efficiently for problems that belong to the complexity class *NC*. The verifier has to scan each input bit and therefore has time-complexity that depends linearly on the input size. The transition from Turing machine algorithm and universal Turing machine circuit to the bare-bones GKR protocol incurs a time penalty for the prover: its time-complexity grows polynomially in the Turing machine time complexity. The exponent in the prover-time-complexity is related to the big-Oh constant of the $O(\log n)$ space complexity of the Turing machine.

Instead, one can consider the circuit wiring algorithms as the computational algorithmic model. In addition, aside from using finite algebraic extension fields for boolean arithmetic during the protocol, one can use arithmetic circuits with residues modulo a prime number as values [25]. We will use hybrid circuits that perform both boolean and finite field arithmetic. The circuit wiring algorithms are public, and for $O(\log s_\ell)$-depth wiring algorithms the prover time-complexity can be reduced to nearly linear [6, 7, 26].

In [11] we have extended the circuit wiring model to which a bare-bones interactive protocol applies. We give a proof-of-work protocol for a circuit evaluation with local wiring functions that are of time-complexity $s_\ell^{O(1)}$, that is, polynomial in the number of bits of the node labels. We note that the model relaxation makes it much easier to program an algorithm via circuit wiring algorithms. However, in [11] we increased the prover time-complexity to cubic in the circuit size; see Section 2.2. Here we present a prover-nearly-optimal implementation of our extended bare-bones protocol, that is, in prover time-complexity that is within a poly-logarithmic factor of the circuit size. In fact, the circuit wiring functions need not be polynomial-time, they can be polynomial-depth [11]; see Section 5. But then the prover time-complexity grows again polynomially. We note that our wiring algorithms are assumed to be of $(\log \text{circuit-size})^{O(1)}$ time complexity. The classical notion of polynomial-time-uniformity of boolean circuits allows a time complexity of $(\text{circuit-size}(n))^{O(1)}$ for computing the circuit for n inputs (cf. [1]), but then the prover space complexity and communication complexity for the circuit construction could be the same, in the worst case.

We also show by a check-pointing technique from automatic differentiation [17] that the prover can be realized to have a space-complexity that is within a log-depth factor of the width of the circuit, while remaining prover-nearly-optimal; see Section 2.4. We note that in [26] similar space savings were implemented by the same check-pointing idea.

The trade-offs between the uniformity that the verifier uses and the work that the prover performs are subtle: in [24] the number of rounds in the interactive protocol is reduced to a constant while incrementing the prover time-complexity polynomially. All our protocols [8–10] for linear algebra are, however, prover-nearly-optimal, some are prover-optimal. We note that we call a protocol optimal if the prover complexity is $T(n) + o(T(n))$, where $T(n)$ is the cost of the algorithm without a proof-of-work certificate on input size n. In the literature protocols of complexity $O(T(n))$ have also been termed optimal, and protocols whose prover-time-complexity is $T(n)^{O(1)}$ are termed doubly-efficient.

The transition from algorithm to certified circuit evaluation can be adapted to proof-of-work, that is, the circuit depth can be reduced by the prover communicating intermediate results to the verifier. We demonstrate the technique for proof-of-work of computing $P = A^E \bmod N$, where A, E, and N are n-bit integers, which constitutes the key substep in the Miller-Rabin randomized primality test. Because repeated squaring requires $\Omega(n)$ modular multiplication, the boolean circuit for P has depth $O(n(\log n)^2)$. The GKR/RRR complexity measures are given in Rows 1a and 1b of Table 1. Our yet-to-be-achieved goal is to have a primality test whose prover time-complexity is $n^2(\log n)^{O(1)}$, whose verifier time-complexity is $n(\log n)^{O(1)}$, and which has $O(1)$ rounds of interaction. Row 2a of Table 1 refers to the protocol of Figure 4, which is based on modulo N squareroots and uses a private coin. Here we show that GKR can be adapted by the baby-steps/giant-steps method to the complexity measures of Row 2b of the Table 1. Note that the Agarwal-Kayal-Saxena (AKS) deterministic algorithm does not have a fast version of $n^{2+o(1)}$ bit complexity (cf. [2]); otherwise AKS could be used in our discussion instead.

Table 1: Proof-of-Work primality protocol complexities

Problem	*Nr.*	*prover time*	*verifier time*	*Nr. of rounds*	*Commun. size*	*Reference*
Primality/compositeness of n-bit integers	1a	$n^{2+o(1)}$	$n^{1+o(1)}$	$n^{1+o(1)}$	$n^{1+o(1)}$	[16] on Miller-Rabin
	1b	$n^{O(1)}$	$n^{1+o(1)}$	$O(1)$	$n^{1+o(1)}$	[24] on Miller-Rabin
	1c	$n^{2+o(1)}$	$n^{1+o(1)}$	0	$n^{1+o(1)}$	Fiat-Shamir heuristic applied to 1a. Soundness unproven [5, 14].
	1d	$n^{O(1)}$	$n^{1+o(1)}$	0	$n^{1+o(1)}$	[19] applied to 1a.
Primality of n-bit integers	2a	$n^{2+o(1)}$	$n^{1+o(1)}$	$O(1)$	$O(n)$	Figure 4; uses private coin
Primality/compositeness of n-bit integers	2b	$n^{2+o(1)}$	$n^{3/2+o(1)}$	$n^{1/2+o(1)}$	$O(n^{3/2})$	This paper, Section 3 for public coin

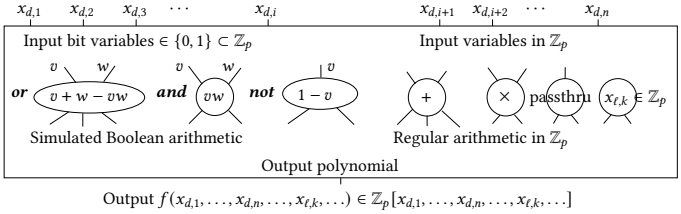
For interactive probabilistic proof protocols with a public randomization coin, whose outcome both the verifier and the prover can see, interaction can be replaced by a cryptographic hash function in the Fiat-Shamir [12] heuristic, which yields a proof of primality/compositeness which can be verified at a later time and whose soundness is justified by the hardness of predicting the hash values. Note that for protocols where interaction is replaced by the Fiat-Shamir idea, a provable soundness requires a special design for the hash functions and cryptographic hardness assumptions [5, 19, 23]. For Google's Rubik's Cube computation mentioned above, such a proof-of-work certificate could have been produced. Row 1c puts Fiat-Shamir in practice by placing the previous hash value as an additional argument to the next hash operation, and may incur a loss of soundness [14], which is mitigated by our protocol of fewer rounds. The papers [5, 19, 23] prove the hardness of predicting the hash values; Row 1d is for the hash functions in [19].

2 THE GOLDWASSER-KALAI-ROTHBLUM INTERACTIVE BARE-BONES PROTOCOL

We briefly describe the bare-bones protocol in [15, 16] (see also [11, 25]). The protocol certifies a computation on a *hybrid* algebraic circuit that performs arithmetic in $\mathbb{Z}_p$ for a prime number p. Figure 1 exhibits the possible operations on Level ℓ of the circuit.

We shall designate the level of the output node, which computes the final value f, as Level 0 and the input values $x_{d,1}, \ldots, x_{d,n}$ as Level d, where d is the depth of the circuit. In our circuits values at Level $(\ell+1)$ get passed on to Level ℓ, and one requires "passthru" nodes to skip over a level. Constants/additional inputs $x_{\ell,k}$ can be introduced on any level before they are used on the next level, and need not be passed thru from input nodes $x_{d,\nu}$. The circuit simulates the boolean operations by arithmetic in $\mathbb{Z}_p$ as shown. However, the GKR protocol itself evaluates the circuit at residues $\neq 0, 1$ for the boolean input variables, which is referred as extension [15]. Proof-of-work is possible on purely algebraic circuits [11, 25], which speeds the protocol because then arithmetic in $\mathbb{Z}_p$ needs not be performed via boolean operations. Our circuit model is hybrid in that it can perform boolean arithmetic when the boolean nodes have $0, 1 \in \mathbb{Z}_p$ as inputs, but the circuit also performs $\mathbb{Z}_p$ arithmetic in nodes which are designated as field arithmetic nodes [11]. Inside a hybrid circuit the binary values $b_\nu, \ldots, b_0$ for the digits of an integer can be converted to a residue $(b_0 + 2b_1 + \cdots + 2^\nu b_\nu \bmod p) \in \mathbb{Z}_p$ and $\mathbb{Z}_p$-field arithmetic can be done from that point on. The residue cannot be converted back to binary representation in subsequent levels, because the magnitudes of the residues cannot be tested by $<$ with arithmetic alone. But tests for $\neq 0$ and modular divisions are possible on the field arithmetic side to some extent: see [11, Remark 2.5, Section 3].

Figure 1: Bounded fan-in hybrid algebraic extension circuit



Output $f(x_{d,1}, \ldots, x_{d,n}, \ldots, x_{\ell,k}, \ldots) \in \mathbb{Z}_p[x_{d,1}, \ldots, x_{d,n}, \ldots, x_{\ell,k}, \ldots]$

For arguments $v, w \in \{0, 1\} \subseteq \mathbb{Z}_p$, the following field operations simulate the boolean functions:

and	***or***	***not***	***equiv***
vw	$v + w - vw$	$1 - v$	$1 + 2vw - v - w$

(1)

In particular, two binary integers $x_0 + 2x_1 + \cdots + 2^{\nu-1}x_{\nu-1}$ and $x'_0 + 2x'_1 + \cdots + 2^{\nu-1}x'_{\nu-1}$ are equal if

$$\mathrm{equal}(\vec{x}, \vec{x}\,') = \prod_{i=0}^{\nu-1} \boldsymbol{equiv}(x_i, x'_i) = \prod_{i=0}^{\nu-1}(1 + 2x_i x'_i - x_i - x'_i), \quad (2)$$

which is a hybrid circuit of degree ν in $x_0, \ldots, x_\nu$ and in $x'_0, \ldots, x'_\nu$.

The width of the circuit on Level ℓ, designated by $\mathrm{width}(\ell)$ is the number of nodes on Level ℓ. Each node on Level ℓ is assigned a bit vector label $\alpha \in \{0, 1\}^{s_\ell}$ where $s_\ell = O(\log_2(\mathrm{width}(\ell)))$. The circuit is known to the verifier by its local wiring algorithms: isor, isand, isnot, …, isplus, isminus, istimes, ispassthru, isconst, src1, src2. For example, $\mathrm{isor}(\ell, \alpha)$ is = 1 if the node labelled α on Level ℓ is a boolean ***or*** node, and it is = 0 otherwise. The functions $\mathrm{src1}(\ell, \alpha) = \beta$ and $\mathrm{src2}(\ell, \alpha) = \gamma$ if the nodes labelled β and γ on Level $(\ell+1)$ feed their values into node α on Level ℓ. The wiring algorithms return the value 0 if the bit-pattern of α does not represent a node in the circuit. The GKR bare-bones protocol requires that the verifier can evaluate the wiring algorithms, that is, the circuit is uniform. In [11] polynomial-time bit complexity in $s_{\max} \stackrel{\mathrm{def}}{=} \max_\ell \log(\mathrm{width}(\ell))$ is shown to suffice. More precisely, we assume:

Assumption 2.1. *the verifier (and prover) can execute the wiring algorithms for $\ell = 0, \ldots, d-1$ (d being the depth of the circuit) to produce boolean circuits for* $\mathrm{isor}(\ell, \alpha), \ldots$ *of size* $s_{\max}^{O(1)}$, *all in total time* $d s_{\max}^{O(1)}$.

Our uniformity requirement extends the log-space uniformity of the complexity class *NC* [16, Theorems 1.1 and 4.7], but even polynomial-time uniformity can be further relaxed (see Section 5).

The GKR bare-bones protocol uses a value-lookup polynomial for each Level ℓ

$$V_0 = w_0,\ V_\ell(z) \stackrel{\mathrm{def}}{=} \sum_{\alpha \in \{0,1\}^{s_\ell}} \mathrm{equal}(z, \alpha) w_{\ell,\alpha},$$
$$z = (z_1, \ldots, z_{s_\ell}),\ V_\ell(a) = w_{\ell,a},\ 1 \le \ell \le d, \quad (3)$$

where $w_{\ell,\alpha} \in \mathbb{Z}_p$ is the value of the circuit at the node on Level ℓ labelled α when evaluated on a given list of inputs. Again, if α is not the bit pattern of a node on Level ℓ, $w_{\ell,\alpha} \stackrel{\mathrm{def}}{=} 0$. For all node labels $a \in \{0, 1\}^{s_\ell}$ we have $V_\ell(a) = w_{\ell,a}$. The protocol certifies $V_\ell(\rho^{[\ell]})$ from a certified $V_{\ell+1}(\rho^{[\ell+1]})$ for computed values $\rho^{[\ell]} \in \mathbb{Z}_p^{s_\ell}$ and $\rho^{[\ell+1]} \in \mathbb{Z}_p^{s_{\ell+1}}$, where $\rho^{[\ell+1]}$ is verifier-selected after $\rho^{[\ell]}$. In (3) one substitutes

$$w_{\ell,\alpha} = W_\ell(\alpha) + \mathrm{isconst}(\ell, \alpha) w_{\ell,\alpha}^{[\mathrm{const}]} \quad (4)$$

with

$$W_\ell(\alpha) \stackrel{\mathrm{def}}{=} \mathrm{isand}(\ell, \alpha) V_{\ell+1}(\mathrm{src1}(\ell, \alpha))\, V_{\ell+1}(\mathrm{src2}(\ell, \alpha))$$
$$+ \mathrm{isnot}(\ell, \alpha)\big(1 - V_{\ell+1}(\mathrm{src1}(\ell, \alpha))\big) + \cdots$$
$$+ \mathrm{ispassthru}(\ell, \alpha)\, V_{\ell+1}(\mathrm{src1}(\ell, \alpha)), \quad (5)$$

where $w_{\ell,\alpha}^{[\mathrm{const}]} \in \mathbb{Z}_p$ are the input values assigned to $x_{\ell,k}$ whose nodes have label α, with $w_{\ell,\alpha}^{[\mathrm{const}]} \stackrel{\mathrm{def}}{=} 0$ for all nodes with $\mathrm{isconst}(\ell, \alpha) = 0$. If all wiring function circuits have depth $O(\log s_{\max})$, one can perform an ($s_{\max}^{O(1)}$-verifier-time) sumcheck protocol [22] to verify the value

$$V_\ell^{[-\mathrm{const}]}(\rho^{[\ell]}) \stackrel{\mathrm{def}}{=} \sum_{\alpha \in \{0,1\}^{s_\ell}} \mathrm{equal}(\rho^{[\ell]}, \alpha)\, W_\ell(\alpha) \quad (6)$$
$$= V_\ell(\rho^{[\ell]}) - \sum_{\alpha \in \{0,1\}^{s_\ell}} \mathrm{equal}(\rho^{[\ell]}, \alpha)\, \mathrm{isconst}(\ell, \alpha) w_{\ell,\alpha}^{[\mathrm{const}]}.$$

The definition (6) requires that $\mathrm{src2}(\ell, \alpha)$ is defined for single-argument nodes, e.g., when $\mathrm{isnot}(\ell, \alpha) = 1$. For such nodes, the local wiring algorithms compute $\mathrm{src2}(\ell, \alpha) = (1, \ldots, 1) \in \{1\}^{s_{\ell+1}}$. The arising interpolants in the sumcheck protocol then have degrees of magnitude $s_{\max}^{O(1)}$. The sumcheck interactions take prover time $O(s_{\max}^{C_1} 2^{s_\ell + s_{\ell+1}})$ for each Level ℓ, where the factor $2^{s_{\ell+1}}$ is from the evaluations of $V_{\ell+1}(\mathrm{src1}(\ell, a))$ and $V_{\ell+1}(\mathrm{src2}(\ell, a))$ (3), and verifier time $O(s_{\max}^{C_2})$ for constants C_1, C_2. For the final verification of the sumcheck protocol, the verifier needs to check $\mathrm{equal}(\rho^{[\ell]}, r^{[\ell]}) \times W_\ell(r^{[\ell]})$ for a vector $r^{[\ell]} \in \mathbb{Z}_p^{s_\ell}$ of random residues. For that, the verifier requires the two values $V_{\ell+1}(\mathrm{src1}(\ell, r^{[\ell]}))$ and $V_{\ell+1}(\mathrm{src2}(\ell, r^{[\ell]}))$ and a 2-to-1 subprotocol is used. The prover interpolates and commits

$$\chi_{\ell+1}(t) \stackrel{\mathrm{def}}{=} V_{\ell+1}\big(t\ \mathrm{src1}(\ell, r^{[\ell]}) + (1-t)\ \mathrm{src2}(\ell, r^{[\ell]})\big) \quad (7)$$

in $\mathbb{Z}_p[t]$ (of degree $\le s_{\ell+1}$) before the verifier asks for certification of $V_{\ell+1}(\rho^{[\ell+1]})$ for $\rho^{[\ell+1]} = r^*\,\mathrm{src1}(\ell, r^{[\ell]}) + (1-r^*)\mathrm{src2}(\ell, r^{[\ell]})$ for a random residue $r^* \in \mathbb{Z}_p$. The protocol proceeds to Level $(\ell+1)$. After completion on Level $(\ell+1)$, the verifier checks $\chi_{\ell+1}(r^*) = V_{\ell+1}(\rho^{[\ell+1]})$ and $W_\ell(r^{[\ell]})$, the latter using the values $\chi_{\ell+1}(0)$ and $\chi_{\ell+1}(1)$ (7), and certifies $V_\ell(\rho^{[\ell]}) = V_\ell^{[-\mathrm{const}]}(\rho^{[\ell]}) + \sum_\alpha \mathrm{equal}(\rho^{[\ell]}, \alpha)\mathrm{isconst}(\ell,\alpha) w_{\ell,\alpha}^{[\mathrm{const}]}$. Note that the computation of $\sum_\alpha$ by the verifier constitutes an input/constant scan discussed in the next paragraph. Alternatively, instead of certifying $V_{\ell+1}(\rho^{[\ell+1]})$ on Level $(\ell+1)$, the protocol on Level $(\ell+1)$ certifies the linear combination $r^* V_{\ell+1}(\mathrm{src1}(\ell, r^{[\ell]})) + r^{**} V_{\ell+1}(\mathrm{src2}(\ell, r^{[\ell]}))$ for verifier-selected random $r^*, r^{**} \in \mathbb{Z}_p$ after the values $V_{\ell+1}(\mathrm{src1}(\ell, r^{[\ell]}))$ and $V_{\ell+1}(\mathrm{src2}(\ell, r^{[\ell]}))$ were committed by the prover [6].

Note that on Level $\ell = d$, $\sum_\alpha \mathrm{equal}(\rho^{[d]}, \alpha)\mathrm{isconst}(d,\alpha) w_{d,\alpha}^{[\mathrm{const}]}$ constitutes the end of the GKR protocol, when $\mathrm{isconst} = 1$ and $w_{d,\alpha}^{[\mathrm{const}]}$ are values for the inputs $x_{d,k}$. The verifier computations $\sum_\alpha \mathrm{equal}(\rho^{[\ell]}, \alpha)\mathrm{isconst}(\ell,\alpha) w_{\ell,\alpha}^{[\mathrm{const}]}$ are the input/constant "scans" which are required by the GKR protocol. Note that instead of an input/constant, $w_{\ell,\alpha}^{[\mathrm{const}]}$ could be given to the verifier and prover by a circuit with inputs α and depth $O(\log s_{\max})$. We will use such inputs below: the decoder circuits (15). Last, by (2) the list of values $\mathrm{equal}(\rho^{[\ell]}, \alpha)$ for all α can be computed a multiplication tree in linear time in the number of α's, counting operations in $\mathbb{Z}_p$.

2.1 Prover-Near-Optimality for Log-Depth Uniformity

The described protocol above has prover cost $\mathcal{S}^2(\log \mathcal{S})^{O(1)}$, where $\mathcal{S}$ is the size of the circuit whose evaluation is being certified. In [7] the prover cost for the arising sumcheck protocols is shown to be $\mathcal{S}(\log \mathcal{S})^{O(1)}$. Because we need the result in Section 2.3 below, we give a brief explanation. Prover-efficient protocols for the sumchecks for (6) are studied extensively, improving the prover-time-complexity to $O(\mathcal{S})$ by using the multi-linearity of the value-lookup function (3); we refer to [26, 27] and the references there. However, our solution for polynomial-time local wiring algorithms in Section 2.3 has the $(\log \mathcal{S})^{O(1)}$ factor even if the protocols in this section have $O(\mathcal{S})$ prover-time-complexity. Therefore we limit our brief discussion to the simpler earlier protocol.

First, we re-write the local wiring functions src1 and src2 as:

$$\left.\begin{aligned} &\mathrm{is\_src1}(\ell,\xi,\eta) \stackrel{\mathrm{def}}{=} \mathrm{equal}(\mathrm{src1}(\ell,\xi),\eta),\\ &\mathrm{is\_src2}(\ell,\xi,\eta) \stackrel{\mathrm{def}}{=} \mathrm{equal}(\mathrm{src2}(\ell,\xi),\eta)\\ &\qquad\qquad \in \mathbb{Z}_p[\xi_1,\ldots,\xi_{s_\ell},\eta_1,\ldots,\eta_{s_{\ell+1}}]. \end{aligned}\right\} \tag{8}$$

For instance, $\mathrm{is\_src1}(\ell,\alpha,\beta)$ is $= 1$ if the node labelled α on Level ℓ takes its first value from the node labelled β at Level $(\ell+1)$; for all other node label pairs (α,γ) with $\gamma \ne \beta$ the function returns 0. The function $W_\ell(\alpha)$ in (5) is re-written as $W_\ell(\alpha) = \sum_{\beta\in\{0,1\}^{s_{\ell+1}}} \sum_{\gamma\in\{0,1\}^{s_{\ell+1}}} F_\ell(\alpha,\beta,\gamma)$ with

$$F_\ell(\alpha,\beta,\gamma) \stackrel{\mathrm{def}}{=} \mathrm{isand}(\ell,\alpha)\mathrm{is\_src1}(\ell,\alpha,\beta)\mathrm{is\_src2}(\ell,\alpha,\gamma)V_{\ell+1}(\beta)V_{\ell+1}(\gamma)$$
$$+\,\mathrm{isnot}(\ell,\alpha)\,\mathrm{is\_src1}(\ell,\alpha,\beta)\gamma_1\cdots\gamma_{s_{\ell+1}}\big(1 - V_{\ell+1}(\beta)\big)\big)$$
$$+\cdots+\mathrm{ispassthru}(\ell,\alpha)\,\mathrm{is\_src1}(\ell,\alpha,\beta)\gamma_1\cdots\gamma_{s_{\ell+1}}V_{\ell+1}(\beta).^1 \tag{9}$$

The term $\gamma_1\cdots\gamma_{s_{\ell+1}}$ guarantees that only one γ in the second sum is selected for nodes with one input; the term acts as a stand-in second source. The sumcheck protocol (6) at Level ℓ is now a triple sum $\sum_\alpha\sum_\beta\sum_\gamma \mathrm{equal}(\rho^{[\ell]},\alpha)F_\ell(\alpha,\beta,\gamma)$. The prover in the sumcheck protocol computes sums $\sum_{\beta^{[\mathrm{rem}]}}\sum_{\gamma^{[\mathrm{rem}]}}\sum_{\alpha^{[\mathrm{rem}]}}$ where the missing bits in $\beta^{[\mathrm{rem}]}$, $\gamma^{[\mathrm{rem}]}$ and $\alpha^{[\mathrm{rem}]}$ are evaluated at field elements, both random elements chosen by the verifier and interpolation arguments by the prover to turn one bit into a variable. The protocol processes the bits of β and γ before α. We shall consider the step in the sumcheck protocol when j bits in β have been set to values $r = (r_1,\ldots,r_j) \in \mathbb{Z}_p^j$, which are fixed elements. The prover has to compute for the plus-nodes on Level ℓ, for instance, the sum

$$\sum_{\beta^{[\mathrm{rem}]}=(r,\beta'):\ \beta'\in\{0,1\}^{s_{\ell+1}-j}}\ \sum_{\gamma\in\{0,1\}^{s_{\ell+1}}}\ \sum_{\mathrm{isplus}(\ell,\alpha)=1:\ \alpha\in\{0,1\}^{s_\ell}} \mathrm{equal}(\rho^{[\ell]},\alpha)\mathrm{is\_src1}(\ell,\alpha,\beta^{[\mathrm{rem}]})\,\mathrm{is\_src2}(\ell,\alpha,\gamma)\times (V_{\ell+1}(\beta^{[\mathrm{rem}]}) + V_{\ell+1}(\gamma)), \tag{10}$$

where the only value for γ in the middle sum for which $\mathrm{is\_src2}(\ell,\alpha,\gamma) \ne 0$ is $\gamma = \mathrm{src2}(\alpha)$, when $\mathrm{is\_src2}(\ell,\alpha,\mathrm{src2}(\alpha)) = 1$. The only $\beta^{[\mathrm{rem}]}$ for which $\mathrm{is\_src1}(\ell,\alpha,\beta^{[\mathrm{rem}]})$ can be $\ne 0$ must have $\beta_{j+1}^{[\mathrm{rem}]} = \beta'_1 = \mathrm{src1}(\ell,\alpha)_{j+1},\ldots,\beta_{s_{\ell+1}}^{[\mathrm{rem}]} = \beta'_{s_{\ell+1}-j} = \mathrm{src1}(\ell,\alpha)_{s_{\ell+1}}$ (2). The sum therefore reduces to

$$\sum_{\mathrm{isplus}(\ell,\alpha)=1:\ \alpha\in\{0,1\}^{s_\ell}} \mathrm{equal}(\rho^{[\ell]},\alpha)\times \mathrm{is\_src1}(\ell,\alpha,\ (r_1,\ldots,r_j,\mathrm{src1}(\alpha)_{j+1},\ldots,\mathrm{src1}(\alpha)_{s_{\ell+1}})\ )\times \big(V_{\ell+1}(r_1,\ldots,r_j,\mathrm{src1}(\alpha)_{j+1},\ldots,\mathrm{src1}(\alpha)_{s_{\ell+1}}) + V_{\ell+1}(\mathrm{src2}(\alpha))\big). \tag{11}$$

The prover has all $V_{\ell+1}(\mathrm{src2}(\alpha)) = w_{\ell+1,\mathrm{src2}(\alpha)}$, the values at node $\mathrm{src2}(\alpha)$ at Level $\ell+1$. The prover pre-computes $V_{\ell+1}(r,\beta')$ for all $\beta' \in \{0,1\}^{s_{\ell+1}-j}$, $2^{s_{\ell+1}-j}$ values in total. For each β' by (3) $V_{\ell+1}(r,\beta') = \sum_\eta \mathrm{equal}(\,(r,\beta'),\,\eta)\,w_{\ell+1,\eta}$, where by (2) at most 2^j terms have $\mathrm{equal}((r,\beta'),\eta) \ne 0$, namely for those η whose last $s_{\ell+1}-j$ bits agree with those of β'. The cost to the prover to compute all $V_{\ell+1}(r,\beta')$ is thus $O(2^{s_{\ell+1}})$ operations in $\mathbb{Z}_p$. With them, the prover then can compute all sums like (11) in $2^{s_\ell} s_{\max}^{O(1)}$ operations. Note that for a given β' the values $V_{\ell+1}(r,\beta')$ may be used arbitrarily often, as the fan-out of the circuit is unbounded. The prover uses a log-depth search tree data structure for the look-up. The bits of γ are processed the same way.

2.2 Polynomial-Time Local Wiring Algorithms

If the wiring function circuits isor, ... on node label inputs of $s_{\max} = O(\log \mathcal{W})$ bit-length have depth $s_{\max}^{O(1)}$, where $\mathcal{W}$ is the maximum width of the original circuit, one cannot perform a sumcheck for $V_\ell^{[-\mathrm{const}]}(\rho^{[\ell]})$ (6,10) directly and maintain $(\log \mathcal{W})^{O(1)}$ verifier-time, because the arising degrees could be exponential in $s_{\max}$. We use the GKR protocol recursively instead. We write

[1] In [16] and later papers, the term $\mathrm{isplus}(\ell,\alpha)\mathrm{is\_src1}(\ell,\alpha,\beta)\mathrm{is\_src2}(\ell,\alpha,\gamma)$, for instance, is lumped into a single function $\widetilde{\mathrm{add}}_\ell(\alpha,\beta,\gamma)$, where the tilde indicates a possible evaluation at values from an algebraic extension of a finite field.

$$G_\ell(\alpha,\beta,\gamma,\lambda_1,\lambda_2) \stackrel{\text{def}}{=} \text{isand}(\ell,\alpha)\,\text{is\_src1}(\alpha,\beta)\,\text{is\_src2}(\alpha,\gamma)\,\lambda_1\lambda_2 + \text{isnot}(\ell,\alpha)\,\text{is\_src1}(\alpha,\beta)\,\gamma_1\ldots\gamma_{s_{\ell+1}}\,(1-\lambda_1) + \cdots + \text{ispassthru}(\ell,\alpha)\,\text{is\_src1}(\alpha,\beta)\,\gamma_1\ldots\gamma_{s_{\ell+1}}\,\lambda_1, \quad (12)$$

and perform a proof-of-work GKR interactive protocol for

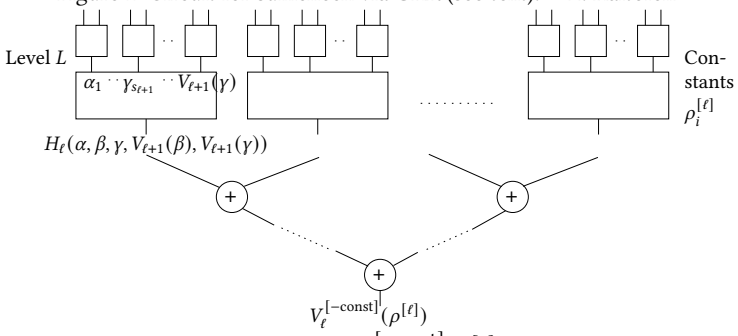
$$V_\ell^{[-\text{const}]}(\rho^{[\ell]}) = \sum_{\alpha\in\{0,1\}^{s_\ell}}\sum_{\beta\in\{0,1\}^{s_{\ell+1}}}\sum_{\gamma\in\{0,1\}^{s_{\ell+1}}}\Big(\text{equal}(\rho^{[\ell]},\alpha)\times G_\ell\big(\,\alpha,\beta,\gamma,\underbrace{V_{\ell+1}(\beta)}_{\lambda_1},\underbrace{V_{\ell+1}(\gamma)}_{\lambda_2}\,\big)\Big). \quad (13)$$

Note that (13) is an equality, where $V_\ell^{[-\text{const}]}(\rho^{[\ell]})$ is defined in (6). The function

$$H_\ell(\underbrace{x_1,\ldots,x_{s_\ell}}_{\xi},\underbrace{x_{s_\ell+1},\ldots,x_{s_\ell+s_{\ell+1}}}_{\eta},\underbrace{x_{s_\ell+s_{\ell+1}+1},\ldots,x_{s_\ell+2s_{\ell+1}}}_{\tau},\lambda_1,\lambda_2) \stackrel{\text{def}}{=} \text{equal}(\rho^{[\ell]},\xi)G_\ell(\,\xi,\eta,\tau,\lambda_1,\lambda_2) \quad (14)$$

under the triple sum in (13) can be represented as a circuit with input bits $\xi_i = \alpha_i$ for $1 \le i \le s_\ell$ and $\eta_j = \beta_j, \tau_j = \gamma_j$ for $1 \le i \le s_{\ell+1}$ and the two input field element values $\lambda_1 = V_{\ell+1}(\beta)$ and $\lambda_2 = V_{\ell+1}(\gamma)$, which depend on the bits of β and γ. In addition, the circuit for H_ℓ accesses a common pool of constants $\rho_i^{[\ell]}$ for $1 \le i \le s_\ell$.

Figure 2: Circuit for sumcheck via GKR (see text). ©E. Kaltofen



The circuit which computes $V_\ell^{[-\text{const}]}(\rho^{[\ell]})$ in (13) is sketched in Figure 2. There are $\sigma_\ell \stackrel{\text{def}}{=} s_\ell + 2s_{\ell+1} + 2$ inputs to each of the $2^{s_\ell+2s_{\ell+1}}$ copies of the circuit for H_ℓ, one for each value of (α,β,γ). Note that σ_ℓ is the number of variables of H_ℓ in (14). Each input is itself a "decoder" [11] circuit, each with $\sigma_\ell + \sigma'_\ell$ input bits $x_1,\ldots,x_{s_\ell+2s_{\ell+1}+2}, x'_1,\ldots,x'_{\sigma'_\ell}$, where $\sigma'_\ell = \lfloor\log_2(\sigma_\ell-1)\rfloor + 1$ represents the number of bits in an integer $\bar\theta = \sum_{k=0}^{\sigma'_\ell}\theta_k 2^k$ which indicates which of the σ_ℓ values are to be output by the decoder. The node labels for the output nodes of the decoder circuits on Level L, which are the inputs for the H's, are $(\alpha,\beta,\gamma,\nu)$ for $0 \le \bar\nu \le \sigma_\ell - 1$. We therefore have the following value look-up function for Level L:

$$V_L(z_1,\ldots,z_{s_\ell+2s_{\ell+1}},z'_1,\ldots,z'_{\sigma'_\ell}) = \sum_{\alpha\in\{0,1\}^{s_\ell}}\sum_{\beta\in\{0,1\}^{s_{\ell+1}}}\sum_{\gamma\in\{0,1\}^{s_{\ell+1}}} \text{equal}(z,(\alpha,\beta,\gamma))\Big(\big(\sum_{\bar\theta=0}^{\sigma_\ell-3}\text{equal}(z',\underbrace{\text{binary-digits}(\bar\theta)}_{\theta})\;(\alpha,\beta,\gamma)_{\bar\theta+1}\big) + \text{equal}(z',\text{binary-digits}(\sigma_\ell-2))\;V_{\ell+1}(\beta) + \text{equal}(z',\text{binary-digits}(\sigma_\ell-1))\;V_{\ell+1}(\gamma)\Big), \quad (15)$$

where $(\alpha,\beta,\gamma)_{\bar\theta+1}$ is the $(\bar\theta+1)$'st entry in the $(s_\ell+2s_{\ell+1})$-dimensional vector (α,β,γ). The function computes

$$V_L(a,b,c,\nu) = \begin{cases} a_i, 1\le i\le s_\ell & \text{for } 0 \le \bar\nu < s_\ell, \\ b_j, 1\le j\le s_{\ell+1} & \text{for } s_\ell \le \bar\nu < s_\ell + s_{\ell+1}, \\ c_k, 1\le k\le s_{\ell+1} & \text{for } s_\ell + s_{\ell+1} \le \bar\nu < s_\ell + 2s_{\ell+1}, \\ V_{\ell+1}(b) & \text{for } \bar\nu = s_\ell + 2s_{\ell+1} = \sigma_\ell - 2, \\ V_{\ell+1}(c) & \text{for } \bar\nu = s_\ell + 2s_{\ell+1} + 1 = \sigma_\ell - 1, \\ 0 & \text{for } \bar\nu \ge s_\ell + 2s_{\ell+1} + 2 = \sigma_\ell \\ & \text{(invalid label)}, \end{cases} \quad (16)$$

which are the required inputs for the circuits for H_ℓ ($\bar\nu$ denotes the integer corresponding to digits in the bit-vector ν). When the GKR protocol for the circuit in Figure 2, which is the sumcheck of Level ℓ in the original circuit evaluation proof-of-work problem, reaches Level L, $V_L(\rho,\rho')$ in (15) needs to be certified for $\rho \in \mathbb{Z}_p^{s_\ell+2s_{\ell+1}}$ and $\rho' \in \mathbb{Z}_p^{\sigma'_\ell}$. Here a standard sumcheck protocol can be used, because the degrees of the function under the triple sum in the entries of α,β,γ are of order $O(s_{\max})$, which is true for $V_{\ell+1}(\beta)$ and $V_{\ell+1}(\gamma)$ by their definitions (3). Again, the verifier at the conclusion of the sumcheck protocol for $V_L(\rho,\rho')$ checks the function under the triple sum for random field elements substituted for the bits in (α,β,γ), and therefore needs the values $V_{\ell+1}(r^{[1]})$ and $V_{\ell+1}(r^{[2]})$, which is accomplished by the 2-1 protocol (7) described above. The protocol then can proceed to Level $(\ell+1)$ in the original circuit.

We finally explain that the circuit wiring functions for the circuit of Figure 2 can be computed from the circuits for $\text{isor}(\ell,\alpha),\ldots$ in such a way that their degrees are $O(\log(s_{\max}))$; note that $s_{\max}$ denotes the maximum number of digits in the node labels in the original hybrid circuit. The node labels in each of the $2^{s_\ell+2s_{\ell+1}}$ copies for H_ℓ (14,12) are all prefixed by those (α,β,γ) for which they compute the value. In each box the nodes then have the same label suffixes $\bar\alpha$, which are of magnitude $s_{\max}^{O(1)}$ because the circuit for H_ℓ has by our uniformity assumption size $s_{\max}^{O(1)}$. We denote by $\bar s_{\max}$ the number of bits in those labels, which is $\bar s_{\max} = O(\log(s_{\max}))$. The prover and verifier compute the maps for the wiring functions from the circuit for H_ℓ: for instance, $\overline{\text{isor}}(\bar k,\bar\alpha)$ maps to the values of isor for each node labelled $\bar\alpha$ at Level $\bar k$ in the circuit for H_ℓ. Now if the local Level $\bar k$ in the $2^{s_\ell+2s_{\ell+1}}$ boxes for H_ℓ in Figure 2 corresponds to global Level k in the entire circuit of Figure 2, we have the corresponding wiring function

$$\text{isor}_{\text{Figure 2 circuit for Level }\ell}(k,\alpha,\beta,\gamma,\bar\alpha) = \sum_{\theta\in\{0,1\}^{\bar s_{\max}}}\text{equal}(\bar\alpha,\theta)\overline{\text{isor}}(\bar k,\theta). \quad (17)$$

By (2), the degree in $\bar\alpha$ in (17) is $\bar s_{\max}$, that is, $O(\log(s_{\max}))$. We have the following for the protocol in [11].

Theorem 2.1. *Our modified GKR bare-bones protocol under the polynomial-time uniformity Assumption 2.1 for the circuit C of size $\mathcal{S}$,*

width $\mathcal{W}$ and depth d for which proof-of-work certification is performed, uses $d\,(\log \mathcal{S})^{O(1)}$ rounds, and in terms of arithmetic operations in $\mathbb{Z}_p$ has verifier time-complexity $O(d\,(\log \mathcal{S})^{O(1)} + n_{\text{tot}})$, where n_{tot} is the the number of inputs/constants to the circuit, which is n in Figure 1 plus the number of the $x_{\ell,k}$'s for $\ell < d$. The prover time-complexity is $d\,\mathcal{W}^3(\log \mathcal{S})^{O(1)}$ using the sumchecks of Section 2.1. The protocol is Monte-Carlo randomized and requires $p = d\,(\log \mathcal{S})^{O(1)}$ for a soundness probability $\geq 1/2$. Specifically, for a boolean circuit with n input and constant bits, whose size is $\mathcal{S} = n^{O(1)}$ and whose depth is $d(n)$ and whose wiring functions have bit-time-complexity $(\log n)^{O(1)}$, the protocol has prover bit-time-complexity $d(n)\,\mathcal{S}^3\,(\log n)^{O(1)}$, verifier time-complexity $O(d(n)\log(n)^{C_1} + n(\text{loglog}\, n)^{C_2})$ for constants C_1, C_2, and $d\,(\log n)^{O(1)}$ rounds.

In Section 2.3 below we show how to modify the circuit in Figure 2 so that the prover time-complexity reduces to $\mathcal{S}(\log \mathcal{S})^{O(1)}$; however, the log-depth wiring circuits $\overline{\text{isor}}, \ldots$ will be much more complicated.

2.3 Prover-Near-Optimality for Polynomial-Time Local Wiring Algorithms

We now show how to perform all sumchecks (6), namely,

$$\sum_{\alpha \in \{0,1\}^{s_\ell}} \text{equal}(\rho^{[\ell]}, \alpha)\, W_\ell(\alpha), \quad W_\ell(\alpha) \text{ defined in (5)}, \tag{18}$$

in $\mathcal{S}(\log \mathcal{S})^{O(1)}$ prover time and $d\,(\log \mathcal{S})^{O(1)}$ verifier time counting operations in $\mathbb{Z}_p$, for all Levels ℓ in total. We assume circuits for the local circuit wiring functions isor, ... are of size $(\log \mathcal{S})^{O(1)}$ and can be computed by the prover and verifier for each level in $(\log \mathcal{S})^{O(1)}$ time; here $\mathcal{S}$ is the size of the original circuit whose evaluation is to be certified, and d is its depth.

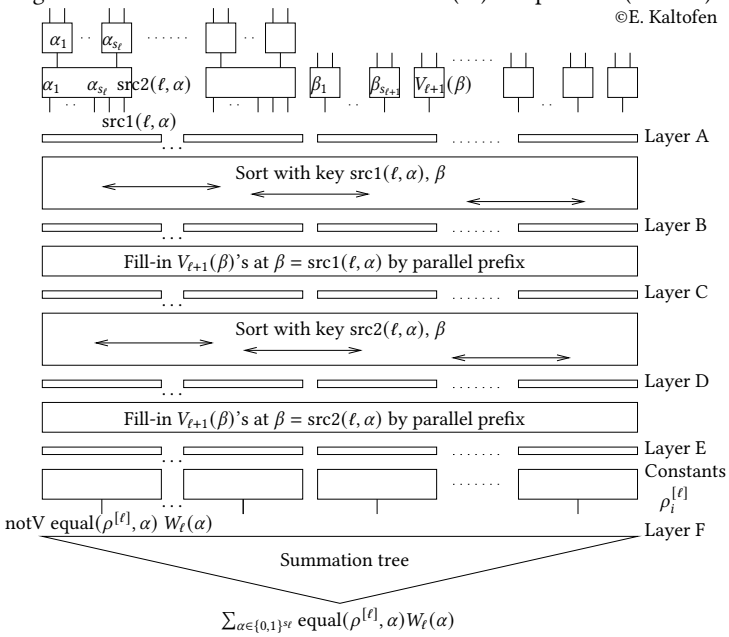
We first briefly describe a summation-tree circuit of size $\mathcal{S}^2 \times (\log \mathcal{S})^{O(1)}$ for (18) for a given Level ℓ. For each node label $\alpha \in \{0,1\}^{s_\ell}$ we compute in parallel circuits from the decoder-supplied inputs $\alpha_1, \ldots, \alpha_{s_\ell}$ (cf. (15)) the values for $\text{src1}(\ell, \alpha)$ and $\text{src2}(\ell, \alpha)$. Each of those circuit boxes (cf. Figure 2) has size $s_\ell^{O(1)}$ by assumption, and outputs the block of bits $\alpha_1, \ldots, \alpha_{s_\ell}, \text{src1}(\ell, \alpha), \text{src2}(\ell, \alpha)$. On that output level we also have decoders for $\beta_1, \ldots, \beta_{s_{\ell+1}}, V_{\ell+1}(\beta)$ for all $\beta \in \{0,1\}^{s_{\ell+1}}$. Next, we compute for each α the two values

$$V_{\ell+1}(\text{src1}(\ell,\alpha)) = \sum_{\beta \in \{0,1\}^{s_{\ell+1}}} \text{equal}(\text{src1}(\ell,\alpha), \beta)\, V_{\ell+1}(\beta),$$
$$V_{\ell+1}(\text{src2}(\ell,\alpha)) = \sum_{\beta \in \{0,1\}^{s_{\ell+1}}} \text{equal}(\text{src2}(\ell,\alpha), \beta)\, V_{\ell+1}(\beta), \tag{19}$$

using two summation trees. Note that the bits for β needed for the equal's are available from the decoders that produce the $\beta_1, \ldots, \beta_{s_{\ell+1}}$, $V_{\ell+1}(\beta)$ blocks of values. The $\overline{\text{src2}}$ local circuit wiring functions, which are determined on that level by the placement of those blocks, locate those bits. The corresponding circuits essentially match sub-blocks of bits and have depth $O(\log s_{\max})$. With the values (19) the circuit computes in parallel for each α the terms $\text{equal}(\rho^{[\ell]}, \alpha) \times W_\ell(\alpha)$ in the sum (18) and then sums up those values in a final summation tree (cf. Figure 2).

The soft-quadratic size solution described in the previous paragraph replaces the dictionary lookup of the values $V_{\ell+1}(r_1, ..., r_j, \text{src1}(\alpha)_{j+1}, ..., \text{src1}(\alpha)_{s_{\ell+1}})$ and $V_{\ell+1}(\text{src2}(\alpha))$ of (11), which are executed by the prover in Section 2.1, by the linear-size search (19) done on the circuit after computing the src1 and src2 values. Note, however, that ultimately dictionary look-ups are still executed by the prover, because each level of the soft-quadratic size circuit for (18) needs to be processed by the GKR protocol of Section 2.1.

Figure 3: Prover's soft-linear time sumcheck (18) computation (see text).



We can avoid the quadratic size blow-up by using sorting networks to locate the values of β matching to $\text{src1}(\alpha)$ and $\text{src2}(\alpha)$. In Figure 3 we display the overall layout of the circuit for (18). On Layer A we have computed blocks of values (notV, addr0, addr1, addr2, V1, V2). If the bit notV = 1, the block is equal $(1, \alpha, \text{src1}(\alpha), \text{src2}(\alpha), 0, 0)$ and if notV = 0, the block is equal $(0, 0, \ldots, 0, \beta, \beta, V_{\ell+1}(\beta), V_{\ell+1}(\beta))$. Note that node values can be duplicated on a subsequent level as passthru nodes with the same src1. Next, the circuit sorts the blocks according to the key (addr1, notV) values. We can use a Batcher sorting network of size $O(\,(2^{s_\ell} + 2^{s_{\ell+1}})\, s_{\max}^2)$ and depth $O(s_{\max}^2)$, counting comparators. The local circuit wiring functions in the Batcher sorting network use simple integer arithmetic on the node labels and are of depth $O(\log s_{\max})$. At Layer B each block $(0, 0 \ldots, 0, \beta, \beta, V_{\ell+1}(\beta), V_{\ell+1}(\beta))$ is followed by blocks $(1, \alpha, \text{src1}(\alpha), \text{src2}(\alpha), 0, 0)$ with $\text{src1}(\alpha) = \beta$, which are followed by $(0, 0 \ldots, 0, \beta+1, \beta+1, V_{\ell+1}(\beta+1), V_{\ell+1}(\beta+1))$, where $\beta+1$ is the binary representation of β incremented by 1. At Layer C the blocks with notV = 1 have their V1 nodes filled-in by $V_{\ell+1}(\beta) = V_{\ell+1}(\text{src1}(\alpha))$ from the closest block to the left with notV = 0. The fill-in is performed on a parallel prefix circuit. An additional bit 'is_filled' is added to each block at Layer B, and initialized to is_filled = 0 if notV = 1 and is_filled = 1 if notV = 0. The parallel prefix uses the following binary operation on a pair of blocks:

$$\begin{aligned}&(\text{notV}, \text{addr0}, \text{addr1}, \text{addr2}, \text{V1}, \text{V2}, \text{is\_filled}) \circ \\ &\qquad (\text{notV}', \text{addr0}', \text{addr1}', \text{addr2}', \text{V1}', \text{V2}', \text{is\_filled}') \\ &= (\text{notV}', \text{addr0}', \text{addr1}', \text{addr2}', \text{V1}^{[\text{new}]}, \text{V2}', \text{is\_filled}^{[\text{new}]}),\end{aligned} \tag{20}$$

where $\mathrm{V1}^{[\mathrm{new}]} = \mathrm{V1}, \mathrm{is\_filled}^{[\mathrm{new}]} = \mathrm{is\_filled}$ if $\mathrm{is\_filled}' = 0$ and $\mathrm{V1}^{[\mathrm{new}]} = \mathrm{V1}', \mathrm{is\_filled}^{[\mathrm{new}]} = 1 = \mathrm{is\_filled}'$ if $\mathrm{is\_filled}' = 1$. The operation $\circ$ is associative and is familiar from carry look-ahead addition/pointer jumping. Again, the parallel prefix circuit has a regular layout and local circuit wiring functions of depth $O(\log s_{\max})$. The size of the parallel prefix circuit is $O(2^{s_\ell} + 2^{s_{\ell+1}})$ and the depth is $O(s_{\max})$, counting $\circ$'s.

The sorting and fill-in is repeated with the key (β, notV) and $(\mathrm{src2}(\alpha), \mathrm{notV})$ in the blocks in the addr2, V2 positions. At Layer E we have in the blocks with notV = 1 the values $(1, \alpha, \mathrm{src1}(\alpha), \mathrm{src2}(\alpha), V_{\ell+1}(\mathrm{src1}(\alpha)), V_{\ell+1}(\mathrm{src2}(\alpha)))$. We now can for each block compute by parallel circuits the argument $\mathrm{equal}(\rho^{[\ell]}, \alpha) W_\ell(\alpha)$ in the sum (18). The bit notV allows setting the values for the blocks with notV = 0 to zero. By the Assumption 2.1 the circuit of Figure 3 has size $(2^{s_\ell} + 2^{s_{\ell+1}}) s_{\max}^{O(1)}$, depth $s_{\max}^{O(1)}$, and local circuit wiring functions $\overline{\mathrm{src1}}, \ldots$ of depth $O(\log(s_{\max}))$, to which the prover-nearly-optimal bare-bones GKR protocol of Section 2.1 can be applied. The scan across the $(s_{\ell+1}+1)2^{s_{\ell+1}}$ decoder circuits for the β_i and $V_{\ell+1}(\beta)$ values before Layer A (cf. (15)) is a sumcheck protocol, which can be performed in prover-time-complexity $2^{s_{\ell+1}} s_{\max}^{O(1)}$ by the algorithm of Section 2.1 (cf. (10)). The protocol ends in the verification of a single $V_{\ell+1}(r_1, \ldots, r_{s_{\ell+1}})$ on the next level of the original circuit, and no 2-1 protocol (7) is required. Of course, the 2-1 protocols are needed in the GKR verification of the circuit of Figure 3. The sumcheck scan across the decoder circuits for the α_i-bits at the top of the circuit in Figure 3 ends by the verifier computing the value of the decoder circuit at a random value in $\mathbb{Z}_p^{s_\ell + s'_\ell}$, where $s_\ell + s'_\ell$ with $s'_\ell = \lfloor \log(s_\ell - 1) \rfloor + 1$ is the number of input bits to the decoders. Our main result can be stated as follows.

Theorem 2.2. *In Theorem 2.1 one may replace the prover time-complexity by $\mathcal{S}(\log \mathcal{S})^{O(1)}$ while all other complexity measures remain the same with different $O(1)$ and C_1, C_2 constants.*

2.4 Prover-Nearly-Optimal Space Complexity

We finally analyze the space complexity for the prover. The circuit size is $\mathcal{S}$, the depth d and the width $\mathcal{W}$. The prover can, at the cost of an additional $O(\log(d))$ factor in time-complexity, reduce the intermediate space to $O(\mathcal{W} \log(d))$. The technique is well-known in the reverse mode of automatic differentiation [17]. We sketch the technique. The idea is to only store ("checkpoint") the values of each node at Level $\lfloor d/2 \rfloor$, where d is the depth of the entire circuit, from which values at the Output-Level 0 to Level $\lfloor d/2 \rfloor - 1$ are recursively determined. The values at Level $(\lfloor d/2 \rfloor + 1)$ to the Input-Level d are then freshly computed from the inputs, again storing only values at half the depths. The strategy stores the values for at most $O(\log(d))$ different checkpoint levels, and each value is re-computed at most $\log(d)$ times. The latter can be shown as follows: re-computation of a value at Level ℓ_2, which is not a checkpoint, is triggered if the values at Level $(\ell_1 + 1)$ are needed, when the values at the nearest-towards-output checkpoint Level ℓ_1 were used in the sum-check protocol for Level $(\ell_1 - 1)$ and afterwards their storage released. During re-computation of the values at Level $(\ell_3 - 1), \ldots$, Level $\ell_2, \ldots$, Level $(\ell_1 + 1)$, where Level ℓ_3 is the nearest checkpoint towards input, the values at Level ℓ_{half}, $\ell_{\mathrm{half}} = \lfloor (\ell_1 + \ell_3)/2 \rfloor$, Level $\lfloor (\ell_1 + \ell_{\mathrm{half}})/2 \rfloor, \ldots$ are stored. Therefore, the depth $(\ell_3 - \ell_1)$, which is the distance of the checkpoint levels between which the unstored Level ℓ_2 lies, is at least halved every time a value on Level ℓ_2 is re-computed.

3 PROOF OF PRIMALITY

In Figure 4 we describe an interactive protocol that certifies an n-bit integer N to be a prime number. The protocol has $n^2 (\log n)^{O(1)}$ prover time-complexity, $n (\log n)^{O(1)}$ verifier time-complexity using the Tonelli-Shanks Algorithm and in C rounds achieves failure probability $\leq 1/2^C$, because for composite $p = q_1 q_2$ with $\mathrm{GCD}(q_1, q_2) = 1$ there are at least 4 squareroots by the Chinese Remainder Theorem. The check that $\sqrt[k]{p} \notin \mathbb{Z}$ for $k \geq 2$ can be done by the algorithms in [4, 21] in $n (\log n)^{O(1)}$ bit operations. Since the randomly sampled r_i, which are uniformly and independently distributed (u.i.d.), must remain unknown to the prover, the Fiat-Shamir [12] heuristic cannot remove interaction. That would be quite important, so that a proof of primality can be stored for verification at a future time.

Figure 4: Protocol for primality with a private verifier coin

Prover	*Verifier*
	$p \geq 3$ a prime number
	Check that $\sqrt[k]{p} \notin \mathbb{Z}$ for all k=2, ..., $\lfloor \log_2(p) \rfloor$
for $i = 1, 2, \ldots, 100$ **do**	
Compute b_i with $\xleftarrow{a_i}$	$a_i = (r_i^2 \bmod p), r_i \xleftarrow{\text{u.i.d.}} \mathbb{Z}_p$ privately
$b_i^2 \equiv a_i \pmod p$ $\xrightarrow{b_i}$	$b_i \stackrel{?}{=} (\pm r_i \bmod p)$

We now realize the Miller-Rabin primality/compositeness test by baby-steps/giant-steps for a protocol of the complexity measures in Line 2a of Table 1, which uses a public coin. The Miller-Rabin primality test for an integer N with $n = \lfloor \log_2(N) \rfloor + 1$ binary digits computes $A^E \bmod N$ for random residues $A \in \mathbb{Z}_N$ with $\mathrm{GCD}(A, N) = 1$ and exponents $E \leq N - 1$. The boolean circuit for those modular exponentiations has size $\mathcal{S} = O(n^2 \log(n) \log\log(n))$ with Schönhage-Strassen integer multiplication and depth $d = O(n \log(n)^2)$ with repeated modular squaring and Cook's long division algorithm. The GKR proof-of-work protocol has prover time-complexity $n^2 (\log n)^{O(1)}$, verifier time-complexity $n (\log n)^{O(1)}$ and $n (\log n)^{O(1)}$ rounds. We show how to reduce the number of rounds to $\sqrt{n} (\log n)^{O(1)}$ by increasing the verifier time-complexity to $n^{3/2} (\log\log n)^{O(1)}$. The prover time-complexity stays $n^2 (\log n)^{O(1)}$. Both the prover's and the verifier's space complexity is $O(n^{3/2})$.

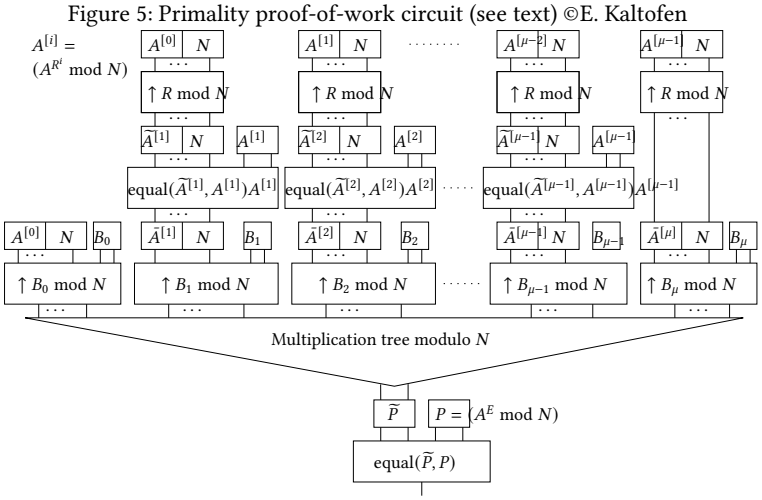
We write $E = b_0 + b_1 2 + \cdots + b_{n-1} 2^{n-1}$ with $b_i \in \{0, 1\}$ in radix $R = 2^\lambda$,

$$E = \sum_{i=0}^{\mu} B_i R^i,\ R = 2^\lambda,\ 0 \leq B_i \leq R - 1,$$
$$\lambda = \lceil \sqrt{n-1} \rceil,\ \mu = \lfloor \sqrt{n-1} \rfloor,\ \lambda\mu + \lambda - 1 \geq n - 1. \quad (21)$$

Figure 5 shows the layout of the "baby steps/giant steps" circuit that tests the value $P = (A^E \bmod N)$ communicated by the prover to the verifier before the GKR protocol for correctness. The prover also communicates the residues $A^{[i]} = (A^{R^i} \bmod N)$ for $1 \leq i \leq \mu - 1$. The initial residue is $A^{[0]} = A$, which for the Miller-Rabin test is a random residue chosen by the verifier. The μ parallel $(\uparrow R \bmod N)$-sub-circuits compute $\widetilde{A}^{[i+1]} = ((A^{[i]})^R \bmod N)$ for $0 \leq i \leq$

$\mu - 1$ by λ squarings modulo N. The equal$(\widetilde{A}^{[i]}, A^{[i]})$ on the next circuit layer check in parallel the computed values against the committed values, except for $(A^{[\mu-1]})^R \bmod N$. The residue $\bar{A}^{[i]} = 0$ if the check fails, which causes the computed $\widetilde{P} = 0$ and the comparison to P produce 0, namely failure. If all committed powers $A^{[i]} \equiv A^{R^i} \equiv A^{2^{i\lambda}} \pmod N$ then $\bar{A}^{[i]} = (A^{R^i} \bmod N)$ for all $1 \le i \le \mu - 1$.

Figure 5: Primality proof-of-work circuit (see text) ©E. Kaltofen



The next layer of $(\uparrow B_i \bmod N)$-sub-circuits computes in parallel $(\bar{A}^{[i]})^{B_i} \bmod N \equiv A^{B_i R^i} \pmod N$ for all $0 \le i \le \mu$, or 0 if the equal failed. In Figure 5, the B_i are inputs to the powering by repeated squarings, although their bits could have been incorporated as constants in each box with the appropriate isconst wiring functions. Finally, the circuit computes $\widetilde{P} = (\prod_{i=0}^{\mu} (\bar{A}^{[i]})^{B_i} \bmod N) \equiv \prod_{i=0}^{\mu} A^{B_i R^i} \pmod N$ and compares the computed value $\widetilde{P}$ with the committed value $P = (A^E \bmod E)$.

Let $M(n) = n \log(n) \operatorname{loglog}(n)$ be the Schönhage-Strassen integer multiplication complexity function (there are smaller theoretical values today [18]). The prover time complexity for computing all bit values in the circuit of Figure 5 inclusive the communicated residues P and $A^{[i]}$ for $1 \le i \le \mu - 1$ is

$$O(\underbrace{\mu M(n) \log(R)}_{\widetilde{A}^{[i]}, i=1,\ldots,\mu} + \underbrace{\mu n}_{\text{equal}} + \underbrace{\mu M(n) \log(\max_i B_i)}_{\uparrow B_i \bmod N, i=0,\ldots,\mu} + \underbrace{\mu M(n)}_{\text{tree product}}), \quad (22)$$

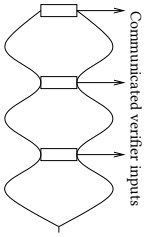
which is $O(nM(n))$; the size of the circuit is also $O(nM(n))$. The depth of the circuit of Figure 5 is $O(\lambda(\log n)^2)$ with $\lambda = O(\sqrt{n})$, where the depth $(\log n)^2$ is from the Cook $O(M(n))$-time division with remainder algorithm using the $O(\log(n))$-deep Schönhage-Strassen multiplication circuit. The GKR protocols to certify the values adds a factor $(\log n)^{O(1)}$ or $O(1)$ to the prover time-complexity. The verifier time-complexity is dominated by the input scans. There are four such scans, the bits of $A^{[i]}$ twice, and the bits of B_i and P. The verifier scan over the $A^{[i]}$ can be reduced to a single scan by passing the values from the Input-Level through to the equal-Level, which makes the circuit wiring functions for the $(\uparrow R \bmod N)$-blocks a bit more complicated, and disallows independent certification of each of the μ powering subcircuits. There are μn bits for all $A^{[i]}$, and the input scans take $n^{3/2}(\operatorname{loglog} n)^{O(1)}$ bit operations, as the modulus p for the field $\mathbb{Z}_p$ over which the GKR protocol is performed is $p = (\log n)^{O(1)}$; the circuit labels have $s_{\max} = O(\log n)$ bits. Note that the protocol without the input scans has verifier time complexity $\sqrt{n}(\log n)^{O(1)}$.

Remark 3.1. If $\operatorname{GCD}(A, N) > 1$ then the prover can take advantage of the GCD-free basis [20] for (A, N): the prover computes and communicates integers $M_1 \ge 2, \ldots, M_k \ge 2$ with $\operatorname{GCD}(M_i, M_j) = 1$ for all $1 \le i < j \le k$ and exponents $\ell_1 \ge 0, \ldots, \ell_k \ge 0$ and $m_1 \ge 0, \ldots, m_k \ge 0$ such that $A = \prod_{i=1}^{k} M_i^{\ell_i}$ and $N = \prod_{i=1}^{k} M_i^{m_i}$, that in $n(\log n)^{O(1)}$ bit operations [3]. The modular power $P = A^E \bmod N$ is then computed by the Chinese remainder algorithm: each residue $A^E \bmod M_i^{m_i}$ for $m_i \ge 1$ is computed by the prover as $M_i^{\ell_i E}(A/M_i^{\ell_i})^E \bmod M_i^{m_i}$. The second power is to a base which is relatively prime to the modulus $M_i^{m_i}$, and the above protocol is used if $\ell_i E < m_i$: $(A^E \bmod M_i^{m_i}) = M_i^{\ell_i E}\big((A/M_i^{\ell_i})^E \bmod M_i^{m_i - \ell_i E}\big)$.

By Section 2.4 for the circuit of Figure 5, which has $\mathcal{W} = n^{3/2} \times (\log n)^{O(1)}$ and $d = n^{1/2}(\log n)^{O(1)}$, the prover space complexity is $n^{3/2}(\log n)^{O(1)}$ while the time complexity stays $n^2(\log n)^{O(1)}$. In that case, namely of verifying $A^E \bmod N$, the prover and verifier can compute a circuit for $A_1 A_2 \bmod N$ and with that circuit derive the circuit wiring functions for the powering circuit of Figure 5. The node values which are stored can be intermediate μ+1 residues of powers modulo N, which further simplifies the protocol.

4 CONCLUSION

Uniformity assumptions on families of boolean circuits that are parameterized with the number n of input bits and which evaluate to functions such as "is a squarefree integer" turn the circuits into algorithms. Proof-of-work of an evaluation at a given input reduces the verifier complexity by assuming efficient algorithms for computing the circuit structure. Because proof-of-work can be applied to the uniformity algorithms themselves, uniformity assumptions can be high in terms of complexity, say uniform of polynomial-depth. Prover-near-optimality can be retained by use of prover-nearly-optimal GKR protocols on the circuits with Batcher sorting networks.



Moreover, the proof-of-work protocol can adapt the circuit evaluation problem to reduce depth of the circuit. The prover can communicate values at intermediate levels to the verifier, which then become inputs to the verifier which are checked against the computed values; cf. the figure on the right. The depth of the proof-of-work circuit then is the maximum depth between the checkpoints.

We have shown how to use the technique for reducing the depth of the Miller-Rabin primality-test circuit of an n-bit integer by a factor of $\sqrt{n}$ while retaining prover-near-optimality.

Our original motivation comes from computations that have relatively narrow width but large depth: the training phase of neural networks. There the weights in the network are adjusted sequentially by testing each item in the training set. Our question is: how does a prover certify that the training was done with all items?

ACKNOWLEDGMENTS

I thank Jean-Guillaume Dumas for discussions on the GKR protocol. I also thank Yael Kalai and the reviewers of two versions of the paper for their comments, in particular on Fiat-Shamir'ing GKR.

This research was supported by the National Science Foundation under Grant CCF-1717100.

Note added on June 5, 2022: Paragraph before Section 2.1: $x_{d,k} \longleftrightarrow x_i$ and y_j.

REFERENCES

[1] P. M. Beame, S. A. Cook, and H. J. Hoover. 1986. Log depth circuits for division and related problems. *SIAM J. Comput.* 15 (1986), 994–1003.

[2] Daniel Bernstein. 2007. Proving Primality in Essentially Quartic Random Time. *Math. Comput.* 76, 257 (July 2007), 389–403. URL: http://cr.yp.to/papers.html.

[3] Daniel J. Bernstein. 2005. Factoring into coprimes in essentially linear time. *J. Algorithms* 54, 1 (2005), 1–30. URL: https://doi.org/10.1016/j.jalgor.2004.04.009.

[4] Daniel J. Bernstein, Hendrik W. Lenstra, Jr., and Jonathan Pila. 2007. Detecting perfect powers by factoring into coprimes. *Math. Comp.* 76 (2007), 385–388.

[5] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. 2019. Fiat-Shamir: from practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, Moses Charikar and Edith Cohen (Eds.). ACM, 1082–1090. URL: https://ia.cr/2018/1004, https://doi.org/10.1145/3313276.3316380.

[6] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. 2017. A Zero Knowledge Sumcheck and its Applications. *Electron. Colloquium Comput. Complex.* 24 (2017), 57. URL: https://eccc.weizmann.ac.il/report/2017/057.

[7] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, Shafi Goldwasser (Ed.). ACM, 90–112. URL: https://doi.org/10.1145/2090236.2090245.

[8] Jean-Guillaume Dumas and Erich L. Kaltofen. 2014. Essentially Optimal Interactive Certificates In Linear Algebra. In *ISSAC 2014 Proc. 39th Internat. Symp. Symbolic Algebraic Comput.*, Katsusuke Nabeshima (Ed.). Association for Computing Machinery, New York, N. Y., 146–153. URL: http://users.cs.duke.edu/~elk27/bibliography/14/DuKa14.pdf.

[9] Jean-Guillaume Dumas, Erich Kaltofen, David Lucas, and Clément Pernet. 2020. Elimination-based certificates for triangular equivalence and rank profiles. *J. Symbolic Comput.* 98 (May–June 2020), 246–269. Special Issue on ISSAC 2017; Mohab Safey El Din, Chee Yap editors. URL: http://users.cs.duke.edu/~elk27/bibliography/18/DKLP18.pdf, https://doi.org/10.1016/j.jsc.2019.07.013.

[10] Jean-Guillaume Dumas, Erich Kaltofen, Emmanuel Thomé, and Gilles Villard. 2016. Linear Time Interactive Certificates for the Minimal Polynomial and the Determinant of a Sparse Matrix. In *ISSAC'16 Proc. 2016 ACM Internat. Symp. Symbolic Algebraic Comput.*, Markus Rosenkranz (Ed.). Association for Computing Machinery, New York, N. Y., 199–206. URL: http://users.cs.duke.edu/~elk27/bibliography/16/DKTV16.pdf.

[11] Jean-Guillaume Dumas, Erich L. Kaltofen, Gilles Villard, and Lihong Zhi. 2017. Polynomial Time Interactive Proofs For Linear Algebra with Exponential Matrix Dimensions And Scalars Given by Polynomial Time Circuits. In *ISSAC '17 Proc. 2017 ACM Internat. Symp. Symbolic Algebraic Comput.*, Michael Burr (Ed.). Association for Computing Machinery, New York, N. Y., 125–132. In memory of Wen-tsun Wu (5/12/1919–5/7/2017). URL: http://users.cs.duke.edu/~elk27/bibliography/17/DKVZ17.pdf.

[12] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO'86 (Lect. Notes Comput. Sci., Vol. 263)*, A. M. Odlyzko (Ed.). Springer, 186–194. URL: https://link.springer.com/chapter/10.1007/0-387-34799-2_18.

[13] Oded Goldreich. 2018. On Doubly-Efficient Interactive Proof Systems. *Foundations and Trends® in Theoretical Computer Science* 13, 3 (2018), 158–246. URL: http://dx.doi.org/10.1561/0400000084.

[14] Shafi Goldwasser and Yael Tauman Kalai. 2003. On the (In)security of the Fiat-Shamir Paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*. IEEE Computer Society, 102–113. URL: https://eprint.iacr.org/2003/034.pdf, https://doi.org/10.1109/SFCS.2003.1238185.

[15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *Annual ACM Symp. Theory Comput. 2008*, Cynthia Dwork (Ed.). ACM Press, 113–122. URL: https:doi.acm.org/10.1145/1374376.1374396.

[16] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4 (2015), 27:1–27:64. URL: http://doi.acm.org/10.1145/2699436.

[17] A. Griewank. 1992. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods & Software* 1 (1992), 35–54.

[18] David Harvey and Joris van der Hoeven. 2021. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics* 193, 2 (2021), 563 – 617. URL: https://doi.org/10.4007/annals.2021.193.2.4.

[19] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. 2021. SNARGs for Bounded Depth Computations and PPAD Hardness from Sub-Exponential LWE. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing* (Virtual, Italy) *(STOC 2021)*. Association for Computing Machinery, New York, NY, USA, 708–721. URL: https://doi.org/10.1145/3406325.3451055, https://eprint.iacr.org/2020/980.

[20] E. Kaltofen. 1985. Sparse Hensel lifting. In *EUROCAL 85 European Conf. Comput. Algebra Proc. Vol. 2 (Lect. Notes Comput. Sci.)*, B. F. Caviness (Ed.). Springer Verlag, Heidelberg, Germany, 4–17. URL: http://users.cs.duke.edu/~elk27/bibliography/85/Ka85_eurocal.pdf.

[21] Erich Kaltofen and Mark Lavin. 2010. Efficiently Certifying Non-Integer Powers. *Computational Complexity* 19, 3 (Sept. 2010), 355–366. URL: http://users.cs.duke.edu/~elk27/bibliography/09/KaLa09.pdf.

[22] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (Oct. 1992), 859–868. URL: http://doi.acm.org/10.1145/146585.146605.

[23] Chris Peikert and Sina Shiehian. 2019. Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 89–114. URL: https://eprint.iacr.org/2019/158, https://doi.org/10.1007/978-3-030-26948-7_4.

[24] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2016. Constant-Round Interactive Proofs for Delegating Computation. Electronic Colloquium on Comput. Complexity. Report No. 61. URL: https://eccc.weizmann.ac.il/report/2016/061/.

[25] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO '13 (Lect. Notes Comput. Sci., Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, 71–89. URL: = https://doi.org/10.1007/978-3-642-40084-1_5.

[26] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 926–943. URL: https://doi.org/10.1109/SP.2018.00060.

[27] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 733–764. URL: https://doi.org/10.1007/978-3-030-26954-8_24.

5 APPENDIX: BEYOND POLYNOMIAL-TIME CIRCUIT-UNIFORMITY

We briefly discuss the uniformity model in [11]. For a computable function, which is parameterized by the number n of input bits, such as testing if an n-bit integer is a squarefree number, a family of circuits of size $\mathcal{S}(n)$, width $\mathcal{W}(n)$ and depth $d(n)$ that computes the function for each n becomes algorithmic by virtue of the circuit wiring algorithms. We also assume that the used constants, which are bits, are the same for all circuits, which means that there is a fixed number of them. Our protocols in Sections 2.2 and 2.1 have assumed that the prover and verifier can compute $(\log \mathcal{W}(n))^{O(1)}$-size circuits for those functions $\mathrm{isor}_n(\ell, \alpha), \ldots$, where n is now also an input to the functions; note that we have $s_{\max} = O(\log \mathcal{W}(n))$. Then the prover complexity is soft-linear in the size $\mathcal{S}(n)$ of the circuits in the family. The GKR verification of the circuit evaluation of Figure 2 uses the wiring functions (17), where $\overline{\mathrm{isor}}, \ldots$ are the wiring functions of the circuits for H_ℓ (14,12), and are therefore derived from the circuits of the wiring functions $\mathrm{isor}_n(\ell, \xi), \ldots$ of the original family of circuits. In short, $\overline{\mathrm{isor}}, \ldots$ are *the wiring functions of the wiring functions*. We observed in [11] that in (12) the circuits

for isor, ... need only be of depth $s_{\max}^{O(1)}$ and size $2^{O(s_{\max})}$, the latter of which is of order $\mathcal{S}(n)^{O(1)}$, provided their own circuit wiring functions $\overline{\text{isor}}, \ldots$ can be constructed by the prover and verifier and have depth $O(\log(s_{\max}))$. Then in (17) we have $\bar{s}_{\max} = O(s_{\max})$, the degrees of $\overline{\text{isor}}, \ldots$ are of order $s_{\max}^{O(1)}$ and the sumcheck protocol at Level k in Figure 2 can include the bits of θ in (17) [11, Assumption 2.2]. The prover complexity will remain $\mathcal{S}(n)^{O(1)}$ and the verifier complexity $O(d(n)\,(\log \mathcal{S}(n))^{C_3} + n(\text{loglog}\,\mathcal{S}(n))^{C_4})$ for constants C_3, C_4, and there are still $d(n)\,(\log \mathcal{S}(n))^{O(1)}$ rounds.

The process of iterating GKR on the sumchecks need not stop at the $\overline{\text{isor}}, \ldots$ wiring algorithms, that is, the assumption that those have depth $O(\log(s_{\max}))$. If at selected place those $\overline{\text{isor}}, \ldots$ have size and depth $s_{\max}^{O(1)}$, GKR could again be iterated for the arising sumchecks.

6 APPENDIX: NOTATION

This appendix is not included in the ISSAC Proceedings.

Notation (in alphabetic order):	
$\circ$	the associative prefix operator that fills-in the values of $V_{\ell+1}(\beta)$ for matching $\beta = \text{src}i(\ell,\alpha)$ (20)
$\stackrel{?}{=}$	verifier check for equality in interactive protocol
$\stackrel{\text{u.i.d.}}{\longleftarrow}$	randomly uniformly independently distributed sample
A	the base of the modular power
$A^{[i]}$	$= (A^{R^i} \bmod N)$
a, α	the integer label of a circuit node at Level ℓ (3)
$\tilde{\text{add}}_\ell(\alpha,\beta,\gamma)$	GKR's orginal circuit wiring function: see Footnote[1]
***and*,*or*,*not*,...**	the arithmetic extension functions simulating boolean logic (1)
B_i	the digits of the exponent E in radix R
b, β	an integer label of a circuit node at Level $(\ell+1)$ (see text after (8))
$C_1, C_2, \ldots$	constants in complexity estimates
c, γ	an integer label of a circuit node at Level $(\ell+1)$ (see text after (8))
$d, d(n)$	the depth of the circuit
E	the exponent of the modular power
equal	the arithmetic extension function testing if two boolean vectors are the same (2)
$F_\ell(\alpha,\beta,\gamma)$	GKR's orginal sumcheck argument (right factor) (9)
$G_\ell(\alpha,\beta,\gamma,\lambda_1,\lambda_2)$	GKR's original sumcheck argument (right factor), parameterized with values from Level $(\ell+1)$ (12)
$H_\ell(\alpha,\beta,\gamma,$ $V_{\ell+1}(\beta), V_{\ell+1}(\gamma))$	GKR's orginial sumcheck argument as a circuit (14)

Notation continued (in alphabetic order):	
isand, isor, isnot, ..., isconst	the wiring functions testing the arithmetic function of a node (arguments: level ℓ, node label α)
is_src1(ℓ,α,β), is_src2(ℓ,α,β)	the wiring functions testing the labels β,γ of the nodes on Level $(\ell+1)$ if they feed into Node α on Level ℓ (8)
$\overline{\text{isor}}, \ldots$	the wiring functions testing the aritmetic function of a node in the circuit for the sumcheck problem (17)
ℓ	a level in the circuit: $\ell = 0$ is the output level, and $\ell = d$ the input level
λ_1, λ_2	the input variables for the values on the preceeding level in H_ℓ
$M(n)$	the complexity of multiplying two n-bit integers
N	the modulus of the modular power
n	a parameter for the size of the input
p	a prime number
R	the radix of the exponent E
$r^{[\ell]}$	the final random selection in the sumcheck protocol on Level ℓ
$\rho^{[\ell]}$	the argument from the 2-to-1 homotopy passed from Level $(\ell-1)$
$\mathcal{S}, \mathcal{S}(n)$	the size of the circuit
s_ℓ	the number of bits in the node labels on Level ℓ
$s_{\max}$	$= \max_{0\le\ell\le d} s_\ell$
src1(ℓ,α), src2(ℓ,α)	the wiring functions computing the labels of the nodes on Level $(\ell+1)$ that feed into Node α on Level ℓ
$\overline{\text{src1}}, \ldots$	the wiring functions for the feeder node label in the circuit for the sumcheck problem (Figures 2 and 3)
$T(n,\ldots)$	time-complexities of algorithms on inputs of size n
$V_L(z,z')$	the input decoder circuit for $H_\ell(\xi,\eta,\tau,\lambda_1,\lambda_2)$ (15,16)
$V_\ell^{[-\text{const}]}(z)$	the value-lookup polynomial at Level ℓ exclusive input/constant values (6)
$W_\ell(\alpha)$	the polynomial which computes $w_{\ell,\alpha}$ from values on Level $(\ell+1)$ (5)
$\mathcal{W}, \mathcal{W}(n)$	the width of the circuit
$w_{\ell,\alpha}$	the value computed on a given input in Node α at Level ℓ (see text after (3))
$w_{\ell,\alpha}^{[\text{const}]}$	the value of the input variable $x_{\ell,k}$ at Node α on Level ℓ (see text after (5))
$x_{\ell,k}$	the input variables at Level ℓ (see Figure 1)
$\chi_\ell(t)$	the symbolic homotopy (7)
$\mathbb{Z}_p$	the integer residues $\{0,\ldots,p-1\}$ modulo p
z	the variable for the node label in $V_\ell(z)$ (3)