SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption

Yael Tauman Kalai
Microsoft Research and MIT

Alex Lombardi*
Simons Institute and UC Berkeley

Vinod Vaikuntanathan†
MIT

October 26, 2022

Abstract

We construct succinct non-interactive arguments (SNARGs) for bounded-depth computations assuming that the decisional Diffie-Hellman (DDH) problem is sub-exponentially hard. This is the first construction of such SNARGs from a Diffie-Hellman assumption. Our SNARG is also *unambiguous*: for every (true) statement x, it is computationally hard to find any accepting proof for x *other than* the proof produced by the prescribed prover strategy.

We obtain our result by showing how to instantiate the Fiat-Shamir heuristic, under DDH, for a variant of the Goldwasser-Kalai-Rothblum (GKR) interactive proof system. Our new technical contributions are (1) giving a TC^0 circuit family for finding roots of cubic polynomials over a special family of characteristic 2 fields (Healy-Viola, STACS '06) and (2) constructing a variant of the GKR protocol whose invocations of the sumcheck protocol (Lund-Fortnow-Karloff-Nisan, STOC '90) only involve degree 3 polynomials over said fields.

Along the way, since we can instantiate Fiat-Shamir for certain variants of the sumcheck protocol, we also show the existence of (sub-exponentially) computationally hard problems in the complexity class PPAD, assuming the sub-exponential hardness of DDH. Previous PPAD hardness results all required either bilinear maps or the learning with errors assumption.

*This research was conducted in part while the author was at MIT, where he was supported by a Charles M. Vest fellowship and the grants of the third author.

†This research was supported in part by DARPA under Agreement No. HR00112020023, a grant from the MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

Contents

1 Introduction 1

1.1 Technical Overview 2

1.1.1 The Circuit Complexity of Root-Finding 3

1.1.2 PPAD hardness with degree 2 sumchecks 4

1.1.3 SNARGs via degree 3 sumchecks 5

1.1.4 Cubic root finding: proving Lemma 1.3 6

1.2 Related Work 7

2 Preliminaries 8

2.1 Cryptographic Groups 8

2.2 Lossy Trapdoor Functions 9

2.3 Correlation-Intractable Hash Families 11

2.4 Lossy CI Hash Functions 12

2.5 SNARGs for Bounded Depth Computations 12

3 Root-Finding in TC^0 13

3.1 Basic Finite Field Operations 13

3.2 Finding roots of $\mathbb{K}$-quadratics in $\mathbb{L}$ 16

3.3 Finding roots of cubics in $\mathbb{K}$ 17

4 PPAD-Hardness from Subexponential DDH 18

5 Delegation for Bounded Depth Computations from Subexponential DDH 20

5.1 Variable-Extended Formulations for Boolean Functions 20

5.2 A GKR protocol with degree 3 sumcheck polynomials 21

1 Introduction

Succinct non-interactive arguments (SNARGs) [Mic94] are short, easy to verify, and computationally sound proofs that a statement x belongs to a potentially complex language L. In their strongest form, one could hope for a non-interactive[1] argument system for any L decidable in non-deterministic time $T(n)$ with proof size $\mathsf{poly}(\lambda, \log T)$ and verification time $\mathsf{poly}(\lambda, \log T) + \tilde{O}(n)$, where $n = |x|$ and λ is a security parameter. In the random oracle model, such arguments are known to exist [Mic94], but there are significant concerns about whether it is possible to construct them based on falsifiable and preferably standard computational assumptions [Bar01, GK03, GW11, BBH+19].

In this work, we consider SNARGs for a *restricted* class of languages: those computable by *logspace-uniform* circuit families of bounded depth D (and arbitrary polynomial size S). These were first studied in the interactive setting by Goldwasser, Kalai, and Rothblum [GKR08], who constructed (statistically sound) interactive proofs of size $D \cdot \mathsf{poly}\log S$ and verification time $D \cdot \mathsf{poly}\log S + \tilde{O}(n)$. Recently, a work of Jawale, Kalai, Khurana, and Zhang [JKKZ21] (building upon [CCH+19]) showed how to convert this proof system into a SNARG by instantiating the Fiat-Shamir heuristic [FS87, BR93] for the [GKR08] protocol in the standard model [CCH+19]. The [JKKZ21] SNARG is proved secure under the sub-exponential hardness of the learning with errors (LWE) assumption. Aside from [JKKZ21], SNARGs for (even unbounded depth) deterministic computation are now known from bilinear maps [KPY19, WW22, KLVW22], the polynomial hardness of LWE [CJJ22], and a combination of the decisional Diffie-Hellman (DDH) and Quadratic Residuosity (QR) assumptions [HJKS22, KLVW22].

SNARGs from DDH. In this work, we ask if SNARGs can be built from the DDH assumption *alone*. We answer this question in the affirmative for SNARGs for bounded-depth computation:

Theorem 1.1. *Assuming the sub-exponential hardness of* DDH*, there exist* SNARGs *for logspace-uniform depth D computation. The* SNARGs *have proof size* $\mathsf{poly}(\lambda) \cdot D$ *and verification time* $\mathsf{poly}(\lambda)(D + n)$.

Moreover, our SNARGs are *unambiguous* [RRR16, CHK+19a], which means that they satisfy a form of soundness even for *true* statements: for $x \in L$, it is computationally hard to find an accepting proof for x *other than* the honestly generated proof guaranteed to exist by completeness. Unambiguous SNARGs were previously constructed in [KPY20, JKKZ21] but only known using either bilinear maps or LWE.

On a slightly more technical level, we show that (unambiguous) SNARGs for bounded depth can be built from a weaker generic primitive than was known before: (lossy) correlation-intractable hash functions [CGH98, CCH+19, JKKZ21] supporting the complexity class TC^0. Previous work relied on a stronger form of correlation intractability (CI) supporting functions in P (or, implicitly, NC). Since CI for TC^0 was constructed based on DDH in [JJ21], this essentially implies Theorem 1.1. We discuss this in more detail in our technical overview (Section 1.1).

[1]As is common, we consider arguments in the common reference string (CRS) model, where the reference string is set up in advance. Throughout this paper, our reference strings will be uniformly random without loss of generality.

PPAD-hardness from DDH. Closely tied to unambiguous SNARGs is the problem of showing *cryptographic hardness* in the complexity class PPAD [Pap94, CHK+19a]. PPAD is a complexity class arising from computational game theory that famously includes finding a Nash equilibrium of bimatrix games as a complete problem [DGP09, CDT09]. The work of [CHK+19a] showed that instantiating the Fiat-Shamir heuristic for (many variants of) the *sumcheck* protocol [LFKN90] suffices to establish PPAD hardness.

In this work, towards instantiating Fiat-Shamir for the [GKR08] protocol, we show how to instantiate Fiat-Shamir for variants of the sumcheck protocol that can be plugged into the [CHK+19a] framework. Thus, we obtain PPAD-hardness from the sub-exponential DDH assumption.

Theorem 1.2. *Assuming that* DDH *is sub-exponentially hard, the complexity class* PPAD *contains problems that are sub-exponentially hard on average.*

Again, we prove Theorem 1.2 by showing that lossy correlation intractable hash functions for TC^0 suffice to construct the non-interactive sumcheck protocol.

In the rest of this introduction, we give a brief overview of our techniques for proving Theorem 1.2 and Theorem 1.1.

1.1 Technical Overview

We begin by discussing our contributions regarding applying the Fiat-Shamir heuristic to the sumcheck protocol [LFKN90]. To do so, we first recall the sumcheck protocol.

The Sumcheck Protocol. In the sumcheck protocol, the prover and verifier begin with a degree[2] d polynomial $f(x_1, \ldots, x_n)$ in n variables over some finite field $\mathbb{F}$. The prover wants to convince the verifier of the value of the *sum* $y = \sum_{a \in \{0,1\}^n} f(a)$, where the sum is taken over $\mathbb{F}$. This is accomplished by the following interactive protocol:

- The prover sends the univariate polynomial $g(x) = \sum_{a_2, \ldots, a_n \in \{0,1\}} f(x, a_2, \ldots, a_n)$.
- The verifier checks that $g(0) + g(1) = y$. If so, it samples a uniformly random $\beta \leftarrow \mathbb{F}$ and sends β to the prover.
- The prover and verifier recursively execute the sumcheck protocol with respect to the polynomial $f_\beta(x_2, \ldots, x_n) = f(\beta, x_2, \ldots, x_n)$ and value $y_\beta = g(\beta)$.
- In the base case, the verifier simply evaluates $f(\beta_1, \ldots, \beta_n)$, which it can do given a circuit for f.

As shown in [CCH+19, JKKZ21], this protocol satisfies what is called (unambiguous) *round-by-round soundness*. Concretely, what this means (for this protocol) is that once the polynomial g is fixed by the prover, if g is *not* the correct "partial sum" polynomial, then with high probability over the choice of β, the prover and verifier will recurse on a false statement (also note that if g is the correct polynomial but the statement (f, y) is false, then the verifier will immediately reject).

[2]Here and below, by "degree" we refer to individual degree: a multivariate polynomial has individual degree $\leq d$ if it has degree $\leq d$ in each variable.

Removing Interaction via Fiat-Shamir. In this work, we want a *non-interactive* variant of the sumcheck protocol, which we obtain by instantiating the Fiat-Shamir heuristic [FS87, BR93] for the sumcheck protocol. Concretely, this means that we will have n hash functions $h_1, \ldots, h_n$ sampled from a hash family $\mathcal{H}$, and the ith verifier challenge β_i is instead computed as a hash $h_i(f, y, g_1, \beta_1, \ldots, g_i)$ of the transcript so far.

Following the *bad challenge function* paradigm of [CCH+19], we call a challenge β_i *bad* for $(f, y, g_1, \beta_1, \ldots, g_i)$ if (1) g_i is *not* the correct polynomial $g_i^* = \sum_{a_{i+1},\ldots,a_n} f(\beta_1, \ldots, \beta_{i-1}, x, a_2, \ldots, a_n)$ and (2) the resulting recursive claim $(f_{\beta_1,\ldots,\beta_i}, g_i(\beta_i))$ is *true*. In the case of the sumcheck protocol, the set of all bad β is precisely the set of *roots* of the polynomial $g_i(x) - g_i^*(x)$. Note that there are at most d such roots, as $g_i(x) - g_i^*(x)$ is a nonzero polynomial of degree at most d. Thus, letting $F_1^{(i)}, \ldots, F_d^{(i)}$ denote functions where $F_j^{(i)}$ maps $(f, \beta_1, \ldots, \beta_{i-1}, g_i)$ to the jth root of $g_i - g_i^*$ in $\mathbb{F}$ (if one exists), we know by [CCH+19] that the resulting non-interactive protocol is sound if each h_i is correlation-intractable (CI) [CGH98, CCH+19] for $F_1^{(i)}, \ldots, F_d^{(i)}$.

Recall that a hash function family $\mathcal{H}$ is CI for a relation R (generalizing the case of a function f) if given $h \leftarrow \mathcal{H}$ it is computationally hard to find an input α such that $(\alpha, h(\alpha)) \in R$. There has been much recent progress on constructing CI hash functions based on standard cryptographic assumptions (e.g. [CCH+19, PS19, BKM20, JJ21, HLR21]). The construction relevant to us in this work is that of [JJ21], which built a CI hash function family supporting functions computable in the complexity class TC^0 (constant-depth threshold circuits) based on sub-exponential DDH.

Thus, in order to use the [JJ21] hash function family, we ask: **what is the computational complexity of the bad challenge functions $F_j^{(i)}$?**

Naively, it is not even clear that the $F_j^{(i)}$ functions are in P, because even computing the polynomial g_i^* (as a function of $f, \beta_1, \ldots, \beta_{i-1}$) seems to require time 2^{n-i}. However, following [JKKZ21], if the functions $h_1, \ldots, h_{i-1}$ are *lossy* [PW08], we can *guess* the challenges $\beta_1, \ldots, \beta_{i-1}$ in advance and non-uniformly hard-wire the polynomial g_i^* in our security reduction (incurring a sub-exponential security loss from guessing $\beta_1, \ldots, \beta_{i-1}$). That is, we will actually define each h_i to be the *composition* of a [JJ21] hash function with a lossy trapdoor function family (LTDF). The resulting composition will still be CI for TC^0 circuits provided that *inversion* of a LTDF can be computed in TC^0, which we observe (see Section 2.2) is possible for a simple modification of standard constructions [PW08, FGK+10].

1.1.1 The Circuit Complexity of Root-Finding

Finally, we arrive at the first of two main challenges in this work. With the setup so far, we have reduced the problem to achieving correlation intractability for a circuit class that has the power to find roots of univariate polynomials over a field $\mathbb{F}$ (and some additional TC^0 operations). If root-finding over finite fields were known to be in TC^0, we would be done! Unfortunately, standard root-finding algorithms [Ber70, Rab80, CZ81] are *not* known to be implementable in TC^0. Indeed, it is clear that some care is required: if p is a large (size 2^λ) prime, finding roots of even *degree* 1 polynomials over $\mathbb{F}_p$ is at least as hard as computing mod p inverses $a \mapsto a^{-1} \pmod{p}$, which is not known to be in TC^0.

Thus, it is clear that to have any hope of TC^0 root-finding, one must carefully choose the field $\mathbb{F}$. In this work, we make use of a special characteristic 2 field ensemble $\mathbb{K} = \{\mathbb{K}_n\}$ studied by Healy and Viola, over which many field operations (including the inversion map $a \mapsto a^{-1}$) are known to be in TC^0 [HV06]. In this work, we show:

Lemma 1.3 (informal, see Theorem 3.8)**.** *There is a* TC^0 *circuit family that finds all roots of* cubic *($d = 3$) univariate polynomials over the [HV06] field ensemble.*

We emphasize that the algorithm in Lemma 1.3 only finds roots that lie in the ground field $\mathbb{K}$, *not* (necessarily) roots that lie in extensions[3] of $\mathbb{K}$. Combining Lemma 1.3 with our discussion so far, we obtain the following result:

Theorem 1.4 (informal)**.** *Fiat-Shamir for* degree 3 *sumcheck protocols over the [HV06] field ensemble is sound using hash functions that are Lossy CI for* TC^0*, and is therefore instantiable under sub-exponential* DDH*.*

That is, by carefully instantiating the field ensemble and designing a special-purpose root-finding algorithm, we show how to use the [JJ21] hash function family to achieve a non-interactive sumcheck for *degree three* polynomials. This is a very limited form of non-interactive sumcheck, but we next show how to leverage this limited form of sumcheck to prove Theorems 1.1 and 1.2. Finally, at the end of this overview we sketch a proof of Lemma 1.3, which is one of our main technical contributions.

1.1.2 PPAD hardness with degree 2 sumchecks

First, we show that PPAD-hardness can be established making use of our non-interactive sumcheck protocol for polynomials of degree as low as 2!

To employ the framework of [CHK$^+$19a], all that we require is that our sumcheck protocol can be used to prove membership in a NP-hard language. In [CHK$^+$19a] this is accomplished by using sumcheck over a *large* characteristic field as an argument system for #SAT.

Since our non-interactive sumcheck works over a characterestic 2 field, we instead start with $\oplus$SAT, the computational problem of counting the *parity* of the number of satisfying assignments of a SAT formula. This problem is also NP-hard. Given such a SAT formula ϕ, this parity can then be expressed as a sumcheck problem over $\mathbb{K}$:

$$\oplus\mathsf{SAT}(\phi) = \sum_{a_1,\ldots,a_n \in \{0,1\}} \phi(a_1, \ldots, a_n).$$

Moreover, ϕ can be arithmetized so that it is represented by a polynomial-size arithmetic circuit over $\mathbb{F}_2 \subset \mathbb{K}$. In order for our non-interactive sumcheck protocol to be applicable, we would need the individual degree of this arithmetization to be at most 3. Given that we are doing a "standard" arithmetization, what is the individual degree of ϕ? If ϕ is a CNF, this is nothing more than the maximum number of clauses that an individual variable appears in.

Conveniently, it is known that $\oplus$SAT on arbitrary formulas reduces to $\oplus$SAT on CNFs (which are not 3-regular) where each variable occurs in at most three clauses [Tov84]. This suffices to establish Theorem 1.2 by invoking Theorem 1.4 and [CHK$^+$19a] with respect to a degree 3 sumcheck protocol.

[3]In fact, our algorithm finds all roots that lie in the unique degree 2 extension of $\mathbb{K}$ but not its algebraic closure.

Variable-Extended Formulations. While we have already proved Theorem 1.2, we will give a slightly different second proof, since this will be a crucial step in proving Theorem 1.1. Specifically, we will prove Theorem 1.2 by invoking a degree 2 (rather than 3) sumcheck protocol. To do this, we start with the sumcheck problem above with respect to (the standard poly-size arithmetization of) ϕ. The individual degree of ϕ may be very large; nevertheless, we will show that $\oplus\mathsf{SAT}(\phi)$ can be expressed as a *different* degree 2 sumcheck.

Concretely, we introduce new variables $y_1, \ldots, y_m$, where m is the number of wires in the NAND-circuit computing ϕ. Consider the polynomial f in $n+m$ variables defined as

$$f(x_1, \ldots, x_n, y_1, \ldots, y_m) = y_m \prod_{(i,j,k)\in\mathsf{Gates}(\phi)} (y_i + y_j y_k) \prod_{i=1}^{n} \left(x_i \prod_{j\in S_i} y_j + (1-x_i) \prod_{j\in S_i} (1-y_j) \right),$$

where for every $i \in [n]$, $S_i \subset [m]$ is defined to be the subset of leaf vertices in ϕ that are assigned the input x_i. In words, f (arithmetically) computes an AND of polynomials indicating that each NAND gate is computed correctly and polynomials indicating that the leaf variables were all assigned with respect to a consistent $x \in \{0,1\}^n$. Thus, for boolean inputs, $f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is either zero or equal to $\phi(x_1, \ldots, x_n)$ (which happens for one "consistent" assignment to y). Therefore,

$$\sum_{a_1,\ldots,a_n\in\{0,1\}} \phi(a_1, \ldots, a_n) = \sum_{\substack{a_1,\ldots,a_n\in\{0,1\}\\ b_1,\ldots,b_m\in\{0,1\}}} f(a_1, \ldots, a_n, b_1, \ldots, b_m),$$

so computing $\oplus\mathsf{SAT}(\phi)$ reduces to an f-sumcheck. Finally, note that f has individual degree 2! Indeed, it is linear in the x_i, quadratic in the non-leaf y_j (as they are each used in two gates of ϕ), and quadratic in the leaf y_j (each is used in one gate of ϕ and has a linear dependence in the "input encoding" part of f).

In general, we say that the above transformation produces a "variable-extended formulation" of a boolean formula ϕ (see Definition 5.2), and this is a key step in proving Theorem 1.1.

1.1.3 SNARGs via degree 3 sumchecks

Armed with our new approach of constructing variable-extended formulations of sumcheck polynomials, we proceed to sketch our proof of Theorem 1.1. We prove Theorem 1.1 by choosing a suitable variant of the [GKR08] protocol, modifying it to rely only on degree 3 sumchecks, and then (essentially[4]) applying Theorem 1.4.

At a high level, the [GKR08] protocol[5] proves that $C(x) = y$ for a logspace-uniform depth D, size S circuit C by iteratively producing pairs of claims about a multilinear extension encoding of each *layer* L_i of the computation tableau of the circuit (when evaluated on input x). That is, each $L_i = L_i(x) \in \{0,1\}^S$ is a string containing the value of all level i vertices in the evaluation of $C(x)$, and L_i is interpreted as a *function* $\ell_i : \{0,1\}^{\log S} \to \{0,1\}$, which can then be extended to a multilinear function $\hat{\ell}_i : \mathbb{K}^{\log S} \to \mathbb{K}$. The GKR protocol begins with an evaluation claim about $\hat{\ell}_D$

[4]The variant of [GKR08] that we use actually runs pairs of sumcheck protocols in parallel with shared verifier randomness (as is done in [JKKZ21]), but this detail does not substantially change the proof.

[5]The variant of [GKR08] that we begin with is due to [JKKZ21]. We then modify it further in this paper.

(the end of the computation) and ends with a pair of evaluation claims about $\hat{\ell}_0$; since the input layer of C has only width n (rather than, say, S/D) $\hat{\ell}_0$ can be evaluated in $O(n)$ field operations and thus can be checked by the verifier.

The key step is to understand how to *reduce* claims about $\hat{\ell}_i$ to claims about $\hat{\ell}_{i-1}$; this boils down to the "sumcheck-friendly" equation

$$\ell_i(a) = \sum_{b,c \in \{0,1\}^{\log S}} \left[\chi^{(i)}_{\text{add}}(a,b,c)\Big(\ell_{i-1}(b) + \ell_{i-1}(c)\Big) + \chi^{(i)}_{\text{mult}}(a,b,c)\Big(\ell_{i-1}(b)\ell_{i-1}(c)\Big) \right],$$

where $\chi_{\text{add}}, \chi_{\text{mult}}$ are the *gate indicator functions* that take as input three wire labels (a,b,c) for the circuit and indicates whether an addition (respectively, multiplication) gate is present at these three wires. This equation can then be extended multilinearly to a similar equation relating $\hat{\ell}_i$ to $\hat{\ell}_{i-1}$.

Given this summary of the GKR protocol, the key question for us is as follows: what is the degree of the sumcheck polynomials? By inspection, it turns out that this degree is one more than the degree of the arithmetizations of $\chi_{\text{add}}, \chi_{\text{mult}}$. Naively, their degree may be up to $O(\log S)$ (i.e., the number of leaves in the boolean formulas for $\chi_{\text{add}}, \chi_{\text{mult}}$), but by using *variable-extended formulations* of these polynomials, we can reduce this degree to 2 (at the cost of adding $O(\log S)$ auxiliary variables to the sumcheck). Note that it is crucial for *prover efficiency* that we only add $O(\log S)$ (rather than $\text{poly} \log S$) new variables, as the prover's running time is exponential in this number of variables. We show that an appropriate extended formulation exists making use of an analysis due to Goldreich [Gol18] of $\chi_{\text{add}}, \chi_{\text{mult}}$.

In total, this results in a GKR protocol variant relying on degree 3 sumchecks over $\mathbb{K}$, and thus we can instantiate Fiat-Shamir for this protocol based on sub-exponential DDH.

1.1.4 Cubic root finding: proving Lemma 1.3

Finally, we sketch a proof of Lemma 1.3, which states that roots of cubic polynomials over Healy–Viola fields $\mathbb{K}$ can be computed in TC^0. We will *not* resort to general-purpose root-finding algorithms [Ber70, Rab80, CZ81] (which all have a high-depth iterative nature) but instead turn to *explicit formulae* for roots of low degree polynomials. We show that these explicit formulae can be converted into low-depth algorithms.

First, let us consider the degenerate cases of linear and quadratic polynomials.

- Root-finding for linear polynomials is equivalent to solving a linear equation over $\mathbb{K}$, which reduces to addition, multiplication, and inversion over $\mathbb{K}$. These operations were shown to be in TC^0 in [HV06].
- Since $\mathbb{K}$ has characteristic 2, root-finding for quadratic polynomials reduces to finding roots of polynomials of the form $x^2 + c$ and $x^2 + x + c$.[6] Then:

[6]This follows from the fact that $az^2 + bz + c = 0 \iff (a/b \cdot z)^2 + (a/b \cdot z) + a/b^2 \cdot c = 0$.

– The polynomial $x^2 + c$ always has a double root of $c^{|\mathbb{K}|/2}$,[7] which can be computed in TC^0 via low-depth exponentiation [HV06].

– The polynomial $x^2 + x + c$ has roots given by an explicit formula as a function of c related to its orbit $\{c, c^2, c^4, \ldots, c^{2^{n-1}}\}$ under the Frobenius map $\alpha \mapsto \alpha^2$. The form depends on the order of two dividing $\log|\mathbb{K}|$ (which turns out to be 1 for the Healy–Viola fields) and is given implicitly in our proof of Theorem 3.6.

Passing to a quadratic extension of $\mathbb{K}$. We note that [CJJ21] also proves that quadratic root-finding in $\mathbb{K}$ is in TC^0 with a different approach; however, we give a more powerful algorithm that actually finds roots of this polynomial in an explicit quadratic extension $\mathbb{L} \supset \mathbb{K}$ (even when no roots in $\mathbb{K}$ exist). This more powerful algorithm is necessary to prove the cubic case of Lemma 1.3.

In order for this to make sense, we must be able to construct $\mathbb{L}$ in a way so that operations in $\mathbb{L}$ are similarly efficient to operations in $\mathbb{K}$. Fortunately, we are able to do this with a careful construction, noting that one way to construct a quadratic extension of $\mathbb{K}$ is to add to it a solution to the equation $x^2 + x + \omega = 0$, where ω is an explicit cube root of unity in $\mathbb{K}$. Since ω alone generates a constant-size field, this greatly simplifies the problem of giving efficient algorithms for operations over $\mathbb{L}$. We show in Theorem 3.4 that all of the relevant field operations on $\mathbb{L}$ are in TC^0, which requires opening up the [HV06] construction and extending their analysis to $\mathbb{L}$.

The Cubic Case. Finally, we compute roots of *cubic* polynomials over $\mathbb{K}$ using an algorithmic variant of the cubic formula [Lag70] over characteristic 2 fields. At a high level, the characteristic 2 cubic formula reduces computing roots of a cubic polynomial (given its coefficients), modulo basic operations, to the following tasks:

1. Finding roots of a related *quadratic* polynomial defined over $\mathbb{K}$.
2. Computing the *cube root* map $\alpha \mapsto \alpha^{1/3}$ (modulo cube roots of unity).
3. Solving a constant-size linear system involving these cube roots.

In Section 3, we show that all of these procedures are computable in TC^0 and thus roots of all cubic polynomials can be computed in TC^0.

One important subtlety is that the roots computed in (1) above are not necessarily in $\mathbb{K}$, but in the quadratic extension $\mathbb{L}$; relatedly, (2) requires computing cube roots of elements of $\mathbb{L}$. One must be careful to argue that (in our setting) the cubic formula algorithm does *not* have to pass into a degree 6 (or degree 3) extension of $\mathbb{K}$, which we have not explicitly constructed.

For a full explanation/proof of Lemma 1.3, we refer the reader to Section 3 (Theorem 3.8).

1.2 Related Work

Fiat-Shamir in the Standard Model. Over the last few years, a line of work (including [CCR16, KRR17, CCRR18, HL18, CCH+19, PS19, BKM20, JJ21, HLR21, CJJ21, CJJ22, HJKS22]) has studied the instantiability of the Fiat-Shamir heuristic using concrete, efficiently computable hash

[7] This is the case since in characteristic 2 fields, $-\alpha = \alpha$ for all α.

function families. Starting with the work of [CCH+19], there have been instantiations based on standard cryptographic assumptions (initially the learning with errors assumption [CCH+19, PS19]). The work of [JJ21] constructed NIZKs for NP under the sub-exponential DDH assumption by building a hash family that is correlation-intractable for TC^0 functions from sub-exponential DDH. Beginning with the works of [CCH+19, JKKZ21], applying Fiat-Shamir to the [GKR08] protocol (to construct SNARGs in the standard model) has been explicitly studied, including a construction based on sub-exponential LWE [JKKZ21]. Finally, more recently the Fiat-Shamir approach has been used to build succinct *batch arguments* for NP [CJJ21, CJJ22, HJKS22] which in turn imply SNARGs for P [CJJ22, KVZ21].

SNARGs without Fiat-Shamir. There have additionally been constructions of SNARGs for P that do not rely on the Fiat-Shamir heuristic [KPY19, GZ21, WW22, KLVW22], all of which currently rely on some form of cryptographic bilinear maps.

Cryptographic Hardness of PPAD. Establishing hardness in PPAD based on cryptographic assumptions has also received considerable attention, including [BPR15, GPS16, CHK+19a, CHK+19b, EFKP20, LV20, KPY20, BCH+22]. Previously, PPAD-hardness was known under the following sets of assumptions:

- Polynomially secure functional encryption [BPR15, GPS16], which can be built by a particular combination of three concrete assumptions [JLS21],
- Super-polynomial hardness of a falsifiable assumption on bilinear maps [KPY20],
- The sub-exponential LWE assumption [JKKZ21], and
- A combination of (polynomially-secure) LWE and the (polynomial) hardness of iterated squaring modulo a composite [BCH+22].

2 Preliminaries

2.1 Cryptographic Groups

Let $\mathbb{G} = \{\mathbb{G}_\lambda\}$ be a group ensemble, indexed by a security parameter λ, such that group operations (and testing equality) can be computed in time $\mathsf{poly}(\lambda)$.

Definition 2.1 (Decisional Diffie-Hellman Assumption). *We say that the decisional Diffie-Hellman (DDH) assumption with time T and advantage μ holds for $\mathbb{G}$ if any $T(\lambda)$-time algorithm $\mathcal{A}(\cdot)$ has advantage at most μ in distinguishing a random "DDH-tuple" (g, g^x, g^y, g^{xy}) from a tuple (g, g^x, g^y, g^z) (for uniformly random x, y, z).*

In this paper, we work exclusively with cryptographic groups satisfying the following conditions:[8]

[8]These groups will be used to instantiate the lossy trapdoor function component of a lossy CI hash family; the CI component does not have to satisfy all of these properties (but DDH must still be sub-exponentially hard).

1. *Iterated group multiplication* $g_1, g_2, \ldots, g_t \mapsto \prod_{i=1}^{t} g_i$ can be computed by a $\mathsf{poly}(\lambda, t)$-size, *low-depth* circuit family. As in [JJ21], there are two flavors of results: one requires TC^0 circuits (which exist for, e.g., subgroups of $\mathbb{Z}_q^\times$), while the other requires threshold circuits (with unbounded fanin) of depth $o(\log \lambda)$ (which exist for standard elliptic curve groups [JJ21]). In the latter case, we will use complexity leveraging: re-define the group security parameter to be $\kappa = \mathsf{poly} \log \lambda$, so that DDH remains polynomially hard and iterated multiplication can be computed in depth $o(\log \log \lambda)$.

2. The DDH assumption holds with inverse-subexonential $\mu = 2^{-\lambda^\epsilon}$ for some constant $\epsilon > 0$. If iterated multiplication requires superconstant-depth threshold circuits, then we require the assumption to hold for $T = 2^{\lambda^\epsilon}$ (so that we can complexity leverage as above), while if iterated multiplication has TC^0 circuits, then we only require the assumption to hold for $T = \mathsf{poly}(\lambda)$.

3. Letting M denote a uniformly random $n(\lambda) \times n(\lambda)$ matrix (for $n(\lambda) = \mathsf{poly}(\lambda)$) modulo $N = |\mathbb{G}|$, we have that M is invertible with probability $1 - \mathsf{negl}(\lambda)$. This holds immediately for prime-order groups or groups with order N that have no polynomial-size prime divisors.

As discussed in [JJ21], properties (1) and (2) are satisfied (that is, DDH is plausible) by common examples such as (subgroups of) $\mathbb{Z}_q^\times$ or groups of $\mathbb{F}_q$-points of elliptic curves.

2.2 Lossy Trapdoor Functions

Lossy trapdoor functions were first defined and constructed in an influential work of Peikert and Waters [PW08]. Loosely speaking, a lossy trapdoor function family contains two types of functions: injective ones and lossy ones, such that one cannot distinguish between a random injective function in the family and a random lossy function in the family. An injective function can be generated together with a trapdoor, which allows one to efficiently invert the function, whereas a lossy function "loses" most information about its preimage, since its image is much smaller than its domain.

Definition 2.2 ((T, ω)-Lossy Trapdoor Family)**.** *A quadruple* $(\mathsf{InjGen}, \mathsf{LossyGen}, \mathsf{Eval}, \mathsf{Inv})$ *of* PPT *algorithms is said to be a* (T, ω)*-lossy trapdoor function family if there exist polynomials* $n = n(\lambda)$*,* $n' = n'(\lambda)$*,* $s = s(\lambda)$ *and* $t = t(\lambda)$ *for which the following syntax and properties are satisfied:*

- ***Syntax.***
  - $\mathsf{InjGen}(1^\lambda)$ *takes as input a security parameter* 1^λ *and outputs a pair* $(\mathsf{k}, \mathsf{td})$*, where* $\mathsf{k} \in \{0,1\}^s$ *is a key corresponding to an injective function and* $\mathsf{td} \in \{0,1\}^t$ *is a corresponding trapdoor.*
  - $\mathsf{LossyGen}(1^\lambda)$ *takes as input a security parameter* 1^λ *and outputs a key* $\mathsf{k} \in \{0,1\}^s$ *corresponding to a lossy function.*
  - $\mathsf{Eval}(\mathsf{k}, x)$ *takes as input a key* $\mathsf{k} \in \{0,1\}^s$ *and an element* $x \in \{0,1\}^n$ *and outputs an element* $y \in \{0,1\}^{n'}$*.*
  - $\mathsf{Inv}(\mathsf{k}, \mathsf{td}, y)$ *takes as input a key* $\mathsf{k} \in \{0,1\}^s$*, a trapdoor* $\mathsf{td} \in \{0,1\}^t$*, and an element* $y \in \{0,1\}^{n'}$*, and outputs an element* $x \in \{0,1\}^n \cup \{\perp\}$*.*

- ***Properties.*** *The following properties hold:*
  - ***Injective Mode.*** *For every* $\lambda \in \mathbb{N}$ *and every* $\mathsf{k} \in \mathsf{InjGen}(1^\lambda)$ *the function* $\mathsf{Eval}(\mathsf{k}, \cdot)$ *is injective. Furthermore, for every* $x \in \{0,1\}^{n(\lambda)}$, $\Pr[\mathsf{Inv}(\mathsf{k}, \mathsf{td}, \mathsf{Eval}(\mathsf{k}, x)) = x] = 1$.[9]
  - ω***-Lossiness.*** *For every* $\lambda \in \mathbb{N}$ *and every* $\mathsf{k} \in \mathsf{LossyGen}(1^\lambda)$ *the function* $\mathsf{Eval}(\mathsf{k}, \cdot)$ *has an image of size* $2^{n(\lambda)-\omega(\lambda)}$.
  - T***-Security.*** *There exists a negligible function* μ *such that for every* $\mathsf{poly}(T)$*-size adversary* $\mathcal{A}$ *and for every* $\lambda \in \mathbb{N}$,

$$\left| \Pr_{\mathsf{k} \leftarrow \mathcal{G}.\mathsf{LossyGen}(1^\lambda)}[\mathcal{A}(\mathsf{k}) = 1] - \Pr_{\mathsf{k} \leftarrow \mathcal{G}.\mathsf{InjGen}(1^\lambda)}[\mathcal{A}(\mathsf{k}) = 1] \right| = \mu(T(\lambda))$$

Theorem 2.3. *[PW08, FGK+10]. Assuming the sub-exponential hardness of* DDH*, for every constant* $0 < \delta < 1$ *and every polynomial* $n(\lambda)$*, there exists a constant* $0 < \epsilon < 1$ *for which there exists a* (T, ω)*-lossy trapdoor function family for* $\omega(\lambda) = n(\lambda) - \lambda^\delta$ *and* $T = 2^{\lambda^\epsilon}$.

Moreover, after a td*-independent preprocessing step, inversion of this function family has threshold circuits of depth* $O(d)$*, provided that large fan-in multiplication over the* DDH *group can be computed in depth* d.

Proof. We slightly modify the construction due to [FGK+10] in order to obtain TC^0 inversion:

- The public key is of the form g^M, where g is a generator for an order p group where DDH is hard and M is a $k \times k$ matrix with entries in $\{0, 1, \ldots, p-1\}$. In *injective mode*, M is a uniformly random invertible matrix. In *lossy mode*, M is a uniformly random rank 1 matrix. Injective and lossy modes are computationally indistinguishable under the DDH assumption.
- The input x is an element of $\{0,1\}^n$. To evaluate the function, one computes $f_{g^M}(x) = g^{Mx}$ by evaluating the matrix-vector product "in the exponent."
- The *trapdoor* in injective mode is as follows:

$$\mathsf{td} = \left[a_{ij}\right]_{ij} = M^{-1},$$

  To invert, we compute $f_{\mathsf{td}}^{-1}(g^z) = g^{M^{-1}z}$, where the matrix-vector product

$$M^{-1}z = \left(\sum_j a_{ij} z_j \right)_i$$

  is computed by *exponentiating* $g^{z_j} \mapsto g^{a_{ij}z_j}$ and then computing k different k-fold products. Algorithmically, this is done as follows:
  - First compute $g_{j,\ell} = g^{2^\ell z_j}$ for all $0 \le j \le \log p$. This does not require td and is thus considered a preprocessing step.

[9]Following [JKKZ21], we require perfect correctness for simplicity only.

  – Given $\{g_{j,\ell}\}, \mathsf{td} = [a_{ij}]_{ij}$, compute (for all i, in parallel)

$$g^{x_i} = \prod_{j,\ell} g_{j,\ell}^{a_{ij}[\ell]},$$

  where $a_{ij}[\ell]$ denotes the ℓth bit of a_{ij}. x_i can then be recovered by checking whether the group element is $\mathsf{id}_{\mathbb{G}}$ or g. Since this online step consists of many parallel iterated product operations, its threshold circuit depth essentially matches that of iterated group multiplication.

- Finally, we observe that in lossy mode, f_{g^M} maps a domain of size p^k to a range of size p, thus achieving the desired amount of lossiness for $p = 2^\lambda$ and $k = \lambda^{1/\delta}$. □

2.3 Correlation-Intractable Hash Families

In this section, we recall the notion of a correlation-intractable (CI) hash family originally defined in [CGH98]. We start by recalling the notion of a hash family.

Definition 2.4. *A hash family $\mathcal{H}$ is associated with two algorithms $(\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$, and parameters $n = n(\lambda)$ and $m = m(\lambda)$, such that:*

- *$\mathcal{H}.\mathsf{Gen}$ is a PPT algorithm that takes as input a security parameter 1^λ and outputs a key k.*
- *$\mathcal{H}.\mathsf{Hash}$ is a polynomial time computable (deterministic) algorithm that takes as input a key $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$ and an element $x \in \{0,1\}^{n(\lambda)}$ and outputs an element $y \in \{0,1\}^{m(\lambda)}$.*

In what follows when we refer to a hash family, we usually do not mention the parameters n and m explicitly.

Definition 2.5 (T-Correlation Intractable [CGH98])**.** *A hash family $\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ is said to be T-correlation intractable (T-CI) for a function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ if the following two properties hold:*

- *For every $\lambda \in \mathbb{N}$, every $f \in \mathcal{F}_\lambda$, and every $k \in \mathcal{H}.\mathsf{Gen}(1^\lambda)$, the functions f and $\mathcal{H}.\mathsf{Hash}(k,\cdot)$ have the same domain and the same co-domain.*
- *For every $\mathsf{poly}(T)$-size $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and every $f \in \mathcal{F}_\lambda$,*

$$\Pr_{\substack{k\leftarrow\mathcal{H}.\mathsf{Gen}(1^\lambda)\\ x\leftarrow\mathcal{A}(k)}}[\mathcal{H}.\mathsf{Hash}(k,x) = f(x)] = \mu(T(\lambda)).$$

Theorem 2.6. *[JJ21] Assuming sub-exponential hardness of DDH against polynomial-time attackers, there exists a constant $\epsilon > 0$ and a T-correlation intractable hash family for TC^0, for $T = T(\lambda) = 2^{\lambda^\epsilon}$.*

2.4 Lossy CI Hash Functions

In this section we recall the notion of a *lossy* CI hash family, originally defined in [JKKZ21].

Definition 2.7 ((T, T', ω)-Lossy CI). *A hash family*

$$\mathcal{H} = (\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{LossyGen}, \mathcal{H}.\mathsf{Hash})$$

is said to be (T, T', ω)-lossy CI for a function family $\mathcal{F}$ if the following holds:

- *$(\mathcal{H}.\mathsf{Gen}, \mathcal{H}.\mathsf{Hash})$ is a T-CI hash family for $\mathcal{F}$ (Definition 2.5).*
- *The additional key generation algorithm $\mathcal{H}.\mathsf{LossyGen}$ takes as input a security parameter λ and outputs hash key k, such that the following two properties hold:*
  - ***T'-Key Indistinguishability.*** *For every $\mathrm{poly}(T')$-size adversary $\mathcal{A}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$*

$$\left| \Pr_{k \leftarrow \mathcal{H}.\mathsf{LossyGen}(1^\lambda)}[\mathcal{A}(k) = 1] - \Pr_{k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)}[\mathcal{A}(k) = 1] \right| = \mu(T'(\lambda)).$$

  - ***ω-Lossiness.*** *For every $\lambda \in \mathbb{N}$ and every $k \in \mathcal{H}.\mathsf{LossyGen}(1^\lambda)$, denoting by $n = n(\lambda)$ the length of elements in the domain of $\mathcal{H}.\mathsf{Hash}(k, \cdot)$,*

$$|\{\mathcal{H}.\mathsf{Hash}(k, x)\}_{x \in \{0,1\}^{n(\lambda)}}| \leq 2^{n(\lambda) - \omega(\lambda)}.$$

Theorem 2.8. *There exists a (T, T', ω)-lossy CI hash family for $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ (Definition 2.5) assuming the existence of the following primitives:*

- *A (T', ω)-lossy trapdoor function family $\mathcal{G}$ (Definition 2.2), such that for every $\lambda \in \mathbb{N}$, $f \in \mathcal{F}_\lambda$, and $\mathsf{k} \in \mathcal{G}.\mathsf{Gen}(1^\lambda)$, the domain of $\mathcal{G}.\mathsf{Eval}(\mathsf{k}, \cdot)$ is equal to the domain of f.*
- *A T-CI hash family $\mathcal{H}$ (Definition 2.5) for the function family $\mathcal{F}'$, where the family $\mathcal{F}' = \{\mathcal{F}'_\lambda\}_{\lambda \in \mathbb{N}}$ is defined as follows: for each $\lambda \in \mathbb{N}$, $f' \in \mathcal{F}'_\lambda$ if and only if there exists $f \in \mathcal{F}_\lambda$, and $(\mathsf{k}, \mathsf{td}) \in \mathcal{G}.\mathsf{Gen}(1^\lambda)$ such that $f'_\lambda(\cdot) = f_\lambda(\mathcal{G}.\mathsf{Inv}(\mathsf{k}, \mathsf{td}, \cdot))$. In fact, this holds even when $\mathcal{G}.\mathsf{Inv}(\mathsf{k}, \mathsf{td}, \cdot)$ is replaced by the* online *phase of an offline/online (with respect to* td*) algorithm for $\mathcal{G}.\mathsf{Inv}$.*

2.5 SNARGs for Bounded Depth Computations

In this section we recall the main theorem from [JKKZ21], which claims that (a variant of) the GKR protocol has a standard model Fiat-Shamir instantiation. The GKR protocol considered in [JKKZ21], as well as the one considered in this work, is slightly different from the original protocol proposed in [GKR08], and we elaborate on this protocol in Section 5.2. In what follows, when we refer to the GKR protocol we refer to the protocol from [JKKZ21].

The GKR protocol is a publicly verifiable interactive proof for proving the correctness of log-space uniform bounded depth computations. Let C be a log-space uniform circuit of depth d and size s. The GKR protocol for proving that $C(x) = 1$ for a given input $x \in \{0,1\}^n$, consists of $d = d(n)$

sub-protocols. Each sub-protocol is a sum-check protocol with $\log s$ variables over a finite field $\mathbb{F}$ of size $\mathsf{poly}(|C|)$. In [GKR08] and [JKKZ21] the finite field $\mathbb{F}$ is taken to be an extension of $\mathsf{GF}[2]$. In this work we take a particular field of size 2^λ, for which computing roots of a degree-3 univariate polynomial can be done in TC^0. See Section 3 for details.

In what follows, for any field ensemble $\mathbb{F} = \mathbb{F}_n$ and any $c = c_n \in \mathbb{N}$ we let $\mathsf{GKR}_{\mathbb{F},c}$ denote an instantiation of the GKR protocol with the field $\mathbb{F}$ and where the degree of each variable in the underlying sum-check protocols inside the GKR protocol is bounded by c. We let $\mathcal{F} = \mathcal{F}_{\mathbb{F},c}$ be the function family where each $f \in \mathcal{F}$ has a degree c univariate polynomial $p : \mathbb{F} \to \mathbb{F}$ hardwired into it. It takes as input a degree c polynomial p' specified by $c + 1$ elements in $\mathbb{F}$, and it outputs a root of $p - p'$ (which is an element in $\mathbb{F}$).

Theorem 2.9. *[JKKZ21] Fix any field ensemble $\mathbb{F} = \mathbb{F}_n$ and any $c = c_n \in \mathbb{N}$. Let ℓ denote the number of rounds in each sum-check protocol in $\mathsf{GKR}_{\mathbb{F},c}$. Fix any $T'(\lambda) \geq \lambda$. Assume there exists a constant $\epsilon > 0$ for which there exists a (T, T', ω)-lossy* CI *hash family for the function family $\mathcal{F}_{\mathbb{F},c}$, with $T(\lambda) = 2^{\ell \cdot \lambda^\epsilon}$ and $\omega(\lambda) = n(\lambda) - \lambda^\epsilon$. Then there exists a hash family $\mathcal{H}$ such that applying the Fiat-Shamir heuristic to the $\mathsf{GKR}_{\mathbb{F},c}$ protocol with the hash family $\mathcal{H}$ results with a T'-sound* SNARG *scheme.*

In Section 5 we show that any log-space uniform computation has a $\mathsf{GKR}_{\mathbb{F},c}$ protocol with $\mathbb{F}$ being any finite field ensemble of size $|\mathbb{F}| = 2^\lambda$ and with $c = 3$. Moreover, in Section 3 we show that computing a root of a degree 3 univariate polynomial over a specific finite field ensemble $\mathbb{F}$ (constructed by Healy and Viola [HV06]) can be done in TC^0. This, together with Theorem 2.9 and Theorems 2.3 and 2.6, yields our SNARG construction (Theorem 5.1).

3 Root-Finding in TC^0

In this section, we recall the finite field ensemble constructed by Healy and Viola [HV06], who show that their fields admit TC^0 circuits for many basic finite field operations (addition, pairwise multiplication, large fan-in multiplication, and exponentiation). We construct explicit degree-2 extensions of all finite fields in this ensemble and prove that the same basic operations in the field extensions have TC^0 circuits. Finally, we show that there are TC^0 circuits finding all roots of a given quadratic or cubic equation in the original field ensemble of [HV06].

The results of this section will be used in later subsections to instantiate the Fiat-Shamir transform and show PPAD-hardness (Section 4) and delegation for bounded-depth computations (Section 5).

3.1 Basic Finite Field Operations

Following Healy and Viola [HV06], we define the following field ensemble $\{\mathbb{K}_n\}_{n=2\cdot 3^\ell}$.

Definition 3.1 (Healy-Viola Fields). *The Healy-Viola (HV) field $\mathbb{K}_n$, which is an extension of $\mathbb{F}_2$ of degree $n = 2 \cdot 3^\ell$, is defined to be the polynomial ring $\mathbb{F}_2[x]/(x^{2\cdot 3^\ell} + x^{3^\ell} + 1)$.*

Theorem 3.2 (Healy-Viola [HV06]). *The field ensemble $\{\mathbb{K}_n\}$ admits a polynomial-size*[10] TC^0 *circuit family for the following operations:*

- *Addition:* $(\alpha_1, \dots, \alpha_t) \mapsto \sum_{i=1}^{t} \alpha_i$ *over* $\mathbb{K}$*.*
- *(Large fan-in) Multiplication:* $(\alpha_1, \dots, \alpha_t) \mapsto \prod_{i=1}^{t} \alpha_i$ *over* $\mathbb{K}$*.*
- *Exponentiation:* $(\alpha, T) \mapsto \alpha^T$ *over* $\mathbb{K}$*. The* TC^0 *circuit size is* $\mathsf{poly}(n, \log T)$*.*

In this work, we need to extend Theorem 3.2 to hold over not just $\mathbb{K}$ but a degree-2 extension $\mathbb{L}/\mathbb{K}$.

Definition 3.3 (Degree-2 field extension of HV fields). *The degree-2 field extension* $\{\mathbb{L}_n\}_{n=2\cdot 3^\ell}$ *of* $\mathbb{K}_n$ *is defined to be the polynomial ring* $\mathbb{L} = \mathbb{K}[y]/(y^2 + y + \omega)$*, where* $\omega = x^{3^\ell} \in \mathbb{K}$*.*

We first show that the polynomial $y^2 + y + \omega$ is irreducible over $\mathbb{K}$ (so that $\mathbb{L}$ is in fact a field), which follows by the following standard algebraic argument. Since the polynomial has degree 2, it suffices to show that all the roots of $y^2 + y + \omega$ in a fixed algebraic closure $\overline{\mathbb{K}} = \overline{\mathbb{F}}_2$ are not in $\mathbb{K}$. We do so by arguing that, on the one hand, any root of $y^2 + y + \omega$ in the algebraic closure $\overline{\mathbb{K}}$ has degree exactly 4 over $\mathbb{F}_2$,[11] and on the other hand, $\mathbb{K}$ does not contain any degree-4 field elements. The latter follows from the fact that $\deg(\mathbb{K}) = 2 \cdot 3^\ell$ is not divisible by 4, so it does not contain a subfield of degree 4 over $\mathbb{F}_2$.[12]

It remains to argue that any root of $y^2 + y + \omega$ in the algebraic closure $\overline{\mathbb{K}}$ has degree exactly 4 over $\mathbb{F}_2$. This holds by the following analysis: we know that $\omega^2 + \omega + 1 = 0$ over $\mathbb{K}$ (but $\omega \notin \mathbb{F}_2$), so $\mathbb{F}_2[\omega]$ has degree 2 over $\mathbb{F}_2$. Moreover, $y^2 + y + \omega$ is irreducible over $\mathbb{F}_2[\omega] \simeq \mathbb{F}_4$. Thus, any root of $y^2 + y + \omega$ lies in $\mathbb{F}_{16}$ (realized as a degree 2 extension of $\mathbb{F}_2[\omega]$) but not $\mathbb{F}_4$.

Having established that $\mathbb{L}$ is well-defined, we proceed to generalize Theorem 3.2.

Theorem 3.4. *The field ensemble* $\{\mathbb{L}_n\}$ *admits a polynomial-size* TC^0 *circuit family for the following operations:*

- *Addition:* $(\alpha_1, \dots, \alpha_t) \mapsto \sum_{i=1}^{t} \alpha_i$ *over* $\mathbb{L}$*.*
- *(Large fan-in) Multiplication:* $(\alpha_1, \dots, \alpha_t) \mapsto \prod_{i=1}^{t} \alpha_i$ *over* $\mathbb{L}$*.*
- *Exponentiation:* $(\alpha, T) \mapsto \alpha^T$ *over* $\mathbb{L}$*. The* TC^0 *circuit size is* $\mathsf{poly}(n, \log T)$*.*

Theorem 3.4 follows by a very similar approach as the proof of Theorem 3.2, making use of some additional properties of $\mathbb{L}$.

Proof. Note that $\alpha \in \mathbb{L}$ is given as an explicit bivariate polynomial $\alpha_0(x) + \alpha_1(x)y$ for $\alpha_0(x), \alpha_1(x) \in \mathbb{K}$. An $\mathsf{AC}^0[\oplus] \subseteq \mathsf{TC}^0$ circuit family for addition then follows immediately by component-wise

[10] As usual, the circuit size will be polynomial in the description length of its input, which will be at least n as a single field element is an n-bit string.

[11] An element α in a field extension $\mathbb{K}$ of $\mathbb{F}_2$ is said to have degree d if d is the minimal degree of a nonzero polynomial p over $\mathbb{F}_2$ such that $p(\alpha) = 0$ (over $\mathbb{K}$).

[12] $\mathbb{F}_{p^d}$ is a subfield of $\mathbb{F}_{p^n}$ if and only if $d \mid n$.

addition. Additionally, note that since

$$\begin{aligned}\Big(\alpha_0(x)+\alpha_1(x)y\Big)&\Big(\beta_0(x)+\beta_0(x)y\Big)\\ &=\alpha_0(x)\beta_0(x)+\Big(\alpha_1(x)\beta_0(x)+\alpha_0(x)\beta_1(x)\Big)y+\alpha_1(x)\beta_1(x)y^2\\ &=\alpha_0(x)\beta_0(x)+\Big(\alpha_1(x)\beta_0(x)+\alpha_0(x)\beta_1(x)\Big)y+\alpha_1(x)\beta_1(x)(y+\omega)\\ &=\Big(\alpha_0(x)\beta_0(x)+\alpha_1(x)\beta_1(x)\omega\Big)+\Big(\alpha_1(x)\beta_0(x)+\alpha_0(x)\beta_1(x)+\alpha_1(x)\beta_1(x)\Big)y,\end{aligned}$$

an $\mathsf{AC}^0[\oplus] \subseteq \mathsf{TC}^0$ circuit for pairwise multiplication over $\mathbb{L}$ follows from the analogous circuits over $\mathbb{K}$.

Next, we consider large fan-in multiplication. Suppose we are given t field elements $\alpha^{(1)}, \ldots, \alpha^{(t)} \in \mathbb{L}_n$ and we want to compute $\prod \alpha^{(i)} \in \mathbb{L}_n$. To do this, we view each $\alpha^{(i)}$ as a bivariate polynomial over $\mathbb{Z}$, and compute (in TC^0) the bivariate polynomial representation of $\prod \alpha^{(i)}$. [HAB02] argues that the analogous product for *univariate* polynomials can be done in (uniform) TC^0, but we can see the same holds for our bivariate polynomials via the following reduction:

- Given a bivariate polynomial $\alpha^{(i)}(x, y)$, define the polynomial $\beta^{(i)}(z) = \alpha^{(i)}(z, z^{n\cdot t})$; the coefficients of $\beta^{(i)}$ can be computed with a TC^0 circuit.
- Compute the polynomial $\prod \beta^{(i)} \in \mathbb{Z}[z]$ by invoking [HAB02].
- Map the coefficients of $\prod \beta^{(i)}$ to the coefficients of $\prod \alpha^{(i)}(x, y)$ via the correspondence $z^k \mapsto x^{k \pmod{nt}} \cdot y^{\lfloor k/nt \rfloor}$; this map can also be computed in TC^0.

Finally, we must reduce this bivariate polynomial $\prod \alpha^{(i)}(x, y)$ modulo $(x^{2\cdot 3^\ell} + x^{3^\ell} + 1, y^2 + y + x^{3^\ell})$; this can be done via the following process:

- Reduce each y exponent modulo 15 (since $y^{15} \equiv 1$, as $y \in \mathbb{L}$ is in a degree 4 extension of $\mathbb{F}_2$),
- Reduce each (constant) power of y modulo $(y^2 + y + x^{3^\ell}, x^{2\cdot 3^\ell} + x^{3^\ell} + 1)$,
- Group terms by power of y (either y^0 or y^1), and
- Reduce each y^j coefficient modulo $x^{2\cdot 3^\ell} + x^{3^\ell} + 1$.

This completes the proof that large fan-in multiplication over $\mathbb{L}$ is in TC^0.

Finally, we consider exponentiation $(\alpha, T) \mapsto \alpha^T \in \mathbb{L}$. T is given as input in binary; by invoking a large fan-in multiplication solver, we can reduce to the case where $T = 2^i$ is a power of 2. Now, note that in $\mathbb{L}$, we have

$$\alpha(x,y)^{2^i} = \alpha\Big(x^{2^i}, y^{2^i}\Big) = \alpha\Big(x^{2^i}, y + \sum_{j=0}^{i-1} \omega^{2^j}\Big),$$

where the first equality follows from the fact that our field has characteristic 2, and the second equality uses the defining equation $y^2 + y + \omega = 0$. The field element $g(\omega) = \sum_{j=0}^{i-1} \omega^{2^j} \in \mathbb{K}$ can be computed in $\mathsf{AC}^0[\oplus]$ (e.g. invoking [HV06]), and $\alpha(\cdot, \cdot)$ is linear in its second argument, so we can compute $\alpha^{2^i} \in \mathbb{L}$ by computing each expression $x^{2^i \cdot k}$ for $k \leq n$, which by [HV06] can be done in $\mathsf{AC}^0[\oplus]$, and invoking pairwise field element multiplication and large fan-in addition circuits. □

3.2 Finding roots of $\mathbb{K}$-quadratics in $\mathbb{L}$

In this section, we give a TC^0 circuit family for solving the following computational problem:

Definition 3.5 (($\mathbb{K},\mathbb{L}$) Quadratic Root Finding)**.** *Given a quadratic polynomial $az^2+bz+c \in \mathbb{K}[z]$, find all zeroes of this polynomial in $\mathbb{L}$.*

Theorem 3.6. $(\mathbb{K},\mathbb{L})$ *quadratic root finding admits a TC^0 circuit family.*

Proof. We break into cases.

- If $a = 0$, then this amounts to computing $b^{-1} \in \mathbb{K}$, which can be done because $b^{-1} = b^{2^n-2}$ and exponentiation is in TC^0 (Theorem 3.2).
- If $a \neq 0$ and $b = 0$, then this amounts to inverting a and computing a square root in $\mathbb{K}$, which can be done because $\sqrt{\alpha} = \alpha^{2^{n-1}}$ for $\alpha \in \mathbb{K}$.
- If $a \neq 0$ and $b \neq 0$, then (by invoking standard field operations) this reduces to the case where $a = 1$ and $b = 1$, as

$$az^2 + bz + c = 0 \iff (a/b \cdot z)^2 + (a/b \cdot z) + a/b^2 \cdot c = 0.$$

Thus, for the rest of the proof, we assume that $a = 1$ and $b = 1$. Moreover, it suffices to find a single solution z^* in $\mathbb{L}$, as the other solution will be $z^* + 1$ (since $\mathbb{L}$ has characteristic 2).

Given $z^2 + z + c = 0$, since $n = 2 \cdot 3^\ell$ is 2 mod 4, solving the equation turns out (via standard theory of finite fields, see e.g. [BSS99] Chapter II) to be related to the $\mathbb{F}_4$-trace map

$$\mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) = \sum_{i=0}^{n/2-1} \alpha^{2^{2i}}$$

as follows. First, we note that for any $\alpha \in \mathbb{K}$, $\mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) \in \mathbb{F}_2[\omega]$, as $\mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha)$ is invariant under the map $z \mapsto z^{2^i}$ for all even i. Additionally, the formula above is computable via a TC^0 (in fact, $\mathsf{AC}^0[\oplus]$) circuit family.

Next, we give a TC^0 (in fact, $\mathsf{AC}^0[\oplus]$) circuit that on input $\alpha \in \mathbb{K}$, outputs $\beta \in \mathbb{K}$ such that $\beta^2 + \beta = \alpha + \mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha)$. The circuit simply computes the expression

$$\beta = \sum_{\substack{0 \le i \le n/2-1 \\ i \text{ odd}}} \alpha^{2^{2i}} + \alpha^{2^{2i+1}}.$$

Observe that

$$\beta^2 + \beta = \sum_{\substack{0 \le i \le n/2-1 \\ i \text{ odd}}} \alpha^{2^{2i}} + \alpha^{2^{2i+2}} = \alpha + \mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha),$$

where the last equation uses the fact that $n/2 - 1$ is even.

Finally, in order to solve the equation $z^2 + z + \alpha = 0$, given that we can compute β above, it suffices by additivity to be able to solve the equation $z^2 + z + c = 0$ for $c = \mathrm{Tr}_{\mathbb{K}/\mathbb{F}_4}(\alpha) \in \mathbb{F}_2[\omega]$. But this can be done by lookup table: for $c = 0$ a solution is 0, for $c = 1$ a solution is ω, for $c = \omega$ a solution is y, and for $c = 1 + \omega$ a solution is $\omega + y$. This completes the proof of Theorem 3.6. $\square$

3.3 Finding roots of cubics in $\mathbb{K}$

In this section, we give a TC^0 circuit family for solving the following computational problem:

Definition 3.7 (($\mathbb{K},\mathbb{K}$) Cubic Root Finding)**.** *Given a cubic polynomial $az^3+bz^2+cz+d \in \mathbb{K}[z]$, find all zeroes of this polynomial that lie in $\mathbb{K}$.*

Theorem 3.8. *$(\mathbb{K},\mathbb{K})$-cubic root finding admits a TC^0 circuit family.*

Proof. If $a=0$, then by Theorem 3.6, we know that $(\mathbb{K},\mathbb{L})$-quadratic root finding can be solved in TC^0, and it is easy to check membership in $\mathbb{K}$ (on an input $\alpha \in \mathbb{L}$), so this suffices to solve $(\mathbb{K},\mathbb{K})$-quadratic root finding as well.

Thus, we now assume that $a=1$. Note that we only want to find all roots in $\mathbb{K}$, so we may assume without loss of generality that **there is at least one root in** $\mathbb{K}$ (or else the problem is vacuous). Under this promise, it follows that *all three roots* will lie in $\mathbb{L}$ (since the polynomial factors into linear and quadratic terms over $\mathbb{K}$). Our algorithm will find all three of these roots (and then check membership in $\mathbb{K}$).

We find these roots by invoking (a special case of) a standard characteristic 2 variant of the cubic formula (following e.g. [Lag70]). Namely, letting $\alpha_0,\alpha_1,\alpha_2$ denote the three roots in $\mathbb{L}$, we will find $\alpha_0,\alpha_1,\alpha_2$ by first solving a related quadratic equation with coefficients in $\mathbb{K}$, then taking cube roots (in $\mathbb{L}$), and then solving a linear system over $\mathbb{L}$.

By Vieta's identities, we know that $\hat{\alpha}_0 := \alpha_0+\alpha_1+\alpha_2 = b$. Letting $\omega = x^{3^\ell} \in \mathbb{K}$ so that $\omega^3=1$, we will eventually also compute the linear combinations

$$\hat{\alpha}_1 = \alpha_0 + \omega\alpha_1 + \omega^2\alpha_2,$$

$$\hat{\alpha}_2 = \alpha_0 + \omega^2\alpha_1 + \omega\alpha_2$$

The map $(\alpha_0,\alpha_1,\alpha_2) \mapsto (\hat{\alpha}_0,\hat{\alpha}_1,\hat{\alpha}_2)$ is always (efficiently) invertible over $\mathbb{L}$, so it suffices to compute $\hat{\alpha}_1,\hat{\alpha}_2$. This is sometimes referred to as the "Lagrange resolvent method."

The field elements $\hat{\alpha}_1$ and $\hat{\alpha}_2$ have been carefully chosen to satisfy useful symmetries when $\alpha_0,\alpha_1,\alpha_2$ are permuted as formal variables:

- Under the cyclic permutation $(\alpha_0,\alpha_1,\alpha_2) \mapsto (\alpha_i,\alpha_{i+1},\alpha_{i+2})$, we have that $\hat{\alpha}_1 \mapsto \omega^i\hat{\alpha}_1$ and $\hat{\alpha}_2 \mapsto \omega^{2i}\hat{\alpha}_2$.
- Under the swap permutation $\alpha_i \leftrightarrow \alpha_j$, we have that $\hat{\alpha}_1 \mapsto \omega^{i+j}\hat{\alpha}_2$ and $\hat{\alpha}_2 \mapsto \omega^{2i+2j}\hat{\alpha}_1$.

The symmetries simplify even further if you consider $\hat{\alpha}_1^3$ and $\hat{\alpha}_2^3$ (since $\omega^3=1$): under cyclic permutation, these expressions are *invariant*, while under a swap permutation, they swap!

Thus, $\hat{\alpha}_1^3+\hat{\alpha}_2^3$ and $\hat{\alpha}_1^3\hat{\alpha}_2^3$ are symmetric under all permutations of $(\alpha_0,\alpha_1,\alpha_2)$. The theory of symmetric polynomials therefore tells us that $\hat{\alpha}_1^3+\hat{\alpha}_2^3$ and $\hat{\alpha}_1^3\hat{\alpha}_2^3$ can be expressed in terms of the *elementary* symmetric polynomials in $\alpha_0,\alpha_1,\alpha_2$, which in our case evaluate to none other than b,c, and d by Vieta's identities. Indeed, one can explicitly check that

$$(\hat{\alpha}_1\hat{\alpha}_2)^3 = (b^2+c)^3$$

and

$$\hat{\alpha}_1^3 + \hat{\alpha}_2^3 = bc + d,$$

and thus $(\hat{\alpha}_1^3, \hat{\alpha}_2^3)$ are solutions to the quadratic equation

$$z^2 + (bc + d)z + (b^2 + c)^3 = 0.$$

By Theorem 3.6, we can hence compute $\hat{\alpha}_1^3, \hat{\alpha}_2^3 \in \mathbb{L}$ with a TC^0 circuit. Finally, since $\hat{\alpha}_1, \hat{\alpha}_2 \in \mathbb{L}$, we can find three candidate values for each of $\hat{\alpha}_1, \hat{\alpha}_2$, by computing cube roots over $\mathbb{L}$; this leads to nine possible root sets for our original problem, which can then be individually checked to find the correct roots.

Thus, we have reduced the problem to computing cube roots over $\mathbb{L}$. For this problem, we use a special case of the Adleman-Manders-Miller algorithm [AMM77]. Specifically, we note that $|\mathbb{L}| - 1 = 2^{4 \cdot 3^\ell} - 1$ is congruent to $3^{\ell+1}$ modulo $3^{\ell+2}$. Then, invoking exponentiation[13] in $\mathbb{L}$, on any input $\alpha \in \mathbb{L}$ we can compute

$$\beta = \alpha^{\frac{|\mathbb{L}|-1-3^{\ell+1}}{3^{\ell+2}}} \in \mathbb{L}.$$

Note that

$$\beta^{3^{\ell+2}} = \alpha^{3^{\ell+1}},$$

and thus β^3/α is a $3^{\ell+1}$th root of unity, the set of which is precisely $S = \{1, x, x^2, \ldots, x^{3^{\ell+1}-1}\}$. We can then enumerate (in parallel) over this $\leq n$-size set to determine (the x-exponent of) β^3/α and thus compute a cube root of α provided that a cube root of β^3/α exists (necessarily within S).

Putting everything together, we obtain the desired TC^0 circuit family for $(\mathbb{K}, \mathbb{K})$-cubic root finding. $\square$

4 PPAD-Hardness from Subexponential DDH

In this section, we prove Theorem 1.2, that PPAD is hard under the sub-exponential DDH assumption. We do this by instantiating the Fiat-Shamir heuristic for the sumcheck protocol executed on polynomials of individual degree 2 over the Healy-Viola field ensemble. We prove that Fiat-Shamir for this protocol is sound under DDH by using a lossy CI hash family for TC^0 (Theorem 2.8) and appealing to TC^0 algorithms for quadratic root finding (Theorem 3.6).

Definition 4.1 ($\oplus$3SAT). *A 3CNF formula ϕ is in the language $\oplus$3SAT if the number of satisfying assignments to ϕ is odd.*

Fact 4.2. *If* NP *is hard (on average), then $\oplus$3SAT is hard (on average).*

In particular, if one-way functions exist, then $\oplus$3SAT is hard on average.

Definition 4.3 (Sumcheck Language). *An instance of the sumcheck language consists of an arithmetic circuit f over some field $\mathbb{F}$, along with a target value y. The pair (f, y) is a YES-instance if*

$$\sum_{x \in \{0,1\}^n} f(x_1, \ldots, x_n) = y.$$

[13] The (large) exponent can also be computed in TC^0 [HAB02], or can be nonuniformly hard-wired for simplicity.

In this work, we observe that if $\oplus$3SAT is hard on average, then there is a hard sumcheck problem over $\mathbb{F}_2$ where the individual degree of f is at most two.

Lemma 4.4. *If $\oplus$3SAT is hard-on-average, then sumcheck over $\mathbb{F}_2$ is hard-on-average with respect to a distribution of (f, y) such that the individual degree of f is at most two.*

Proof. We describe a one-to-one reduction mapping $\oplus$3SAT formulas ϕ to sumcheck polynomials f, so that deciding whether $\phi \in \oplus$3SAT maps to checking whether $(f, 1)$ is a valid sumcheck instance.

Suppose that ϕ is an n-variable, m-clause 3CNF:

$$\phi(x_1, \dots, x_n) = \bigwedge_{j=1}^{m} \phi_j(x_{j_1}, x_{j_2}, x_{j_3})$$

where each ϕ_j is an OR of three variables $(x_{j_1}, x_{j_2}, x_{j_3})$ with some negation pattern (contained in the description of ϕ_j). Then, consider the following formula in $3m$ variables:

$$f(z = (z_{j,k})_{j\in[m],k\in\{1,2,3\}}) = \prod_{j=1}^{m} \phi_j(z_{j,1}, z_{j,2}, z_{j,3}) \prod_{i=1}^{n} \left(\prod_{j,k:j_k=i} z_{j,k} + \prod_{j,k:j_k=i} (1 - z_{j,k}) \right),$$

where ϕ_j can be interpreted as a multilinear polynomial in three variables over $\mathbb{F}_2$. We observe that:

- f has individual degree at most 2. This is because the two products are individually multilinear.
- For $z \in \{0,1\}^{3m}$, $f(z) = 1$ if and only if for some $x \in \{0,1\}^n$, $\phi(x) = 1$ and $z_{j,k} = x_{j_k}$ for all (j, k). Otherwise, $f(z) = 0$.

Thus, we see that

$$\sum_{x\in\{0,1\}^n} \phi(x_1, \dots, x_n) = \sum_{y\in\{0,1\}^{3m}} f(y) \pmod 2.$$

This completes the reduction. □

To conclude that PPAD is hard-on-average, we combine Lemma 4.4 with the unambiguous non-interactive argument system for sumcheck from [JKKZ21]. [JKKZ21] implies the following result:

Theorem 4.5 ([JKKZ21], translated)**.** *Let K be a field (ensemble) of size 2^λ. Then, there exists an updatable, unambiguous non-interactive argument system for* Sumcheck$_K$ *for individual degree d polynomials assuming the existence of a hash family $\mathcal{H}$ that is lossy CI (Definition 2.7) for a class of functions that enumerate over all roots of a given univariate degree d polynomial over K.*

By Theorem 2.6, Theorem 2.8, and Theorem 2.3, we know that there exists a lossy CI hash family for TC^0 circuits under sub-exponential DDH. Moreover, letting $\{K_\lambda\}$ denote the field ensemble defined in Definition 3.1, we showed that roots of degree 2 polynomials over K can be enumerated in TC^0. Thus, by Theorem 4.5, we conclude that the claimed argument system exists under sub-exponential DDH.

Finally, it is known that an argument system satisfying the conditions of Theorem 4.5 (along with the hardness of the underlying sumcheck problem) implies the hardness of PPAD [CHK+19a], so this completes the proof of Theorem 1.2.

5 Delegation for Bounded Depth Computations from Subexponential DDH

In this section, we apply and extend our techniques to prove our main theorem on SNARGs for bounded-depth computation.

Theorem 5.1. *Assuming the sub-exponential hardness of the DDH assumption, there exists a* SNARG *for any logspace uniform depth d and size s computation, where the size of the* SNARG *and the* crs *is bounded by $d \cdot \mathsf{poly}(\lambda, \log s)$ and the verification time is $(n + d) \cdot \mathsf{poly}(\lambda, \log s)$, where n is the length of the input.*

Our SNARG is obtained by applying the Fiat-Shamir heuristic to a variant of the GKR protocol, considered in [KPY18, JKKZ21] (building on a simplification of the original GKR protocol due to [Gol18]).

5.1 Variable-Extended Formulations for Boolean Functions

In this section we show how to reduce the degree of any boolean formula down to individual degree at most 2, by adding auxiliary variables. Loosely speaking, this is done by adding a variable corresponding to each wire in the original formula, and computing the original formula by making a series of consistency checks.

Definition 5.2. *Let $f(x_1, \ldots, x_m)$ be a boolean function on m variables. We say that $g(x_1, \ldots, x_m, z_1, \ldots, z_t)$ is a* variable-extended formulation *of f if for every $x \in \{0,1\}^m$, there exists a* unique *$z(x) \in \{0,1\}^t$ such that $g(x, z(x)) = f(x)$, and $g(x, z) = 0$ for all $z \neq z(x)$.*

Lemma 5.3. *Let $f(x_1, \ldots, x_m)$ be a NAND-boolean formula of size s. Then, there exists a variable-extended formulation g of f such that (1) $t = s$, and (2) g can be computed by a $\mathbb{F}_2$-arithmetic circuit of size $O(s)$ that defines a (formal) polynomial of individual degree at most 2.*

Also, the above arithmetic circuit can be constructed in time $\mathsf{poly}(s)$ *given the description of f.*

Proof. We use a similar strategy as in Lemma 4.4. That is, we introduce s new variables $z_1, \ldots, z_s$, one for each wire of the formula computing f. We then define

$$g(x, z) = z_s \prod_{i=1}^{s} g_i(z) \prod_{j=1}^{m} g'_j(x, z),$$

where for every gate (i, j, k) we have $g_i(z) = z_i + z_j z_k$ and for every input index j we have $g'_j(x, z) = x_j \prod_{i \in S_j} z_i + (1 - x_j) \prod_{i \in S_j} (1 - z_i)$, where S_j denotes the set of leaf indices corresponding to x_j. Note that $g(x, z)$ has individual degree 2, since (1) z_s appears only twice, (2) each intermediate (non-output, non-leaf) variable only appears twice because they have fan-in 1 and fan-out 1, and (3) the variables $\{x_j, z_i\}_{j \in [m], i \in S_j}$ have degree at most 2 (they occur at most once in the first product, while the second product is multilinear). □

5.2 A GKR protocol with degree 3 sumcheck polynomials

In this section, we construct a special variant of the [GKR08] interactive proof system for logspace-uniform depth d computation. Our starting point is the GKR protocol variant described in [KPY18, JKKZ21], which makes use of observations from [Gol18] to simplify the protocol. In [JKKZ21], it was shown that the Fiat-Shamir heuristic can be instantiated for this protocol using a hash function that is "lossy correlation-intractable" for circuits that (modulo basic field operations) compute roots of univarite polynomials (of polylogarithmic degree). They show how to construct such a lossy correlation-intractable hash functions from

By using an appropriate *variable-extended formulation* (Lemma 5.3), we will modify the protocol so that every sumcheck sub-protocol uses a polynomial of individual degree *at most* 3. Finally, working over the field ensemble from Definition 3.1 and using the correlation-intractable hash family of [JJ21] (and lossy trapdoor functions from DDH [PW08]), we will deduce Theorem 5.1.

The Protocol. Let $C = \{C_n\}_n$ denote a family of logspace-uniform circuits of depth d and width w. We assume without loss of generality that C has fan-in 2 and consists of addition (mod 2) and multiplication (mod 2) gates. The key objects of interest are the *gate-indicator functions* $\chi^{(i)}_{\text{add}}, \chi^{(i)}_{\text{mult}}$ for each layer (i) of the circuit. $\chi^{(i)}_{\text{add}}$ and $\chi^{(i)}_{\text{mult}}$ take as input three strings $(a, b, c) \in \{0,1\}^{\log w}$ and output whether (a, b, c) is an addition (respectively, multiplication) gate in C.

The protocol is typically defined with respect to particular *low-degree extensions* $\widetilde{\chi}^{(i)}_{\text{add}}, \widetilde{\chi}^{(i)}_{\text{mult}}$ of $\chi_{\text{add}}, \chi_{\text{mult}}$. For our variant, we make use of the following fact shown implicitly in [Gol18]:

Fact 5.4. *Let C' be any family of logspace-uniform circuits of depth d and size s. Then, there exists a family C of logspace-uniform circuits of depth $d \cdot \mathsf{poly}\log(s)$ and size $\mathsf{poly}(s)$ such that:*

- *C computes the same function as C', and*
- *For all i, $\chi^{(i)}_{\text{add}}, \chi^{(i)}_{\text{mult}}$ (for C) are computable by* boolean formulas *of size $O(\log w)$ (i.e., the size is linear in the $\chi_{\text{add}}, \chi_{\text{mult}}$ input length). These formulas can be constructed (by a uniform Turing machine) in time $\mathsf{poly}(\log s)$.*

[Gol18] only explicitly claims that the formulas have size $\mathsf{poly}\log s$, but the construction in [Gol18] Section 3.4.2 actually (specializing to $H = \{0, 1\}$) implies Fact 5.4.

Thus, we assume without loss of generality that C satisfies the conclusion of Fact 5.4. Invoking Lemma 5.3, we conclude that $\chi^{(i)}_{\text{add}}, \chi^{(i)}_{\text{mult}}$ have *variable-extended formulations* $\psi^{(i)}_{\text{add}}, \psi^{(i)}_{\text{mult}} : \{0,1\}^{3\log w + O(\log s)} \to \{0,1\}$ that are computable by $\mathbb{F}_2$-arithmetic circuits of size $O(\log s)$ that define polynomials of individual degree at most 2. We let $\widetilde{\psi}^{(i)}_{\text{add}}, \widetilde{\psi}^{(i)}_{\text{mult}}$ denote the corresponding individual degree 2 polynomials.

We are finally ready to describe the protocol, which will use arithmetic over an extension K of $\mathbb{F}_2$. Our instantiation will use the field ensemble from Definition 3.1.

- The prover and verifier, given the logspace-uniform Turing machine that constructs C, both compute arithmetic circuit descriptions of each $\widetilde{\psi}^{(i)}_{\text{add}}, \widetilde{\psi}^{(i)}_{\text{mult}}$.

- The prover, given the input x and circuit C, computes the following quantities:
  - For every layer i of the circuit, compute the string $L_i = L_i(C,x) \in \{0,1\}^w$ consisting of all wire values in the evaluation $C(x)$ in the ith layer of C.
  - For each such i, define the function $\ell_i : \{0,1\}^{\log w} \to \{0,1\}$ such that $\ell_i(a) = (L_i)_a$, where a is interpreted as an integer between 0 and $w-1$. Implicitly, this defines a *multi-linear extension* $\hat{\ell}_i : K^{\log w} \to K$ of ℓ_i.
- The prover and verifier recursively agree on a *pair* of claims of the form "$\hat{\ell}_i(u_1) = v_1$," "$\hat{\ell}_i(u_2) = v_2$" for $u_1, u_2 \in K^{\log w}$ and $v_1, v_2 \in K$. They do so as follows:
  - The base case is $i = d$, the top (output) layer of C; the claims are (both) that $\hat{\ell}_d(0^{\log w}) = y$ (where allegedly $C(x) = y$).
  - Inductively, suppose that we have two claims "$\hat{\ell}_i(u_1) = v_1$," "$\hat{\ell}_i(u_2) = v_2$" about layer i. The recursion uses the fact that

$$\begin{aligned}\hat{\ell}_i(u) &= \sum_{a \in \{0,1\}^{\log w}} \widehat{EQ}(u,a)\ell_i(a) \\ &= \sum_{a,b,c \in \{0,1\}^{\log w}} \widehat{EQ}(u,a)\Big(\chi^{(i)}_{\text{add}}(a,b,c)\cdot(\ell_{i-1}(b)+\ell_{i-1}(c)) + \chi^{(i)}_{\text{mult}}(a,b,c)\ell_{i-1}(b)\cdot\ell_{i-1}(c)\Big). \\ &= \sum_{\substack{a,b,c \in \{0,1\}^{\log w} \\ z \in \{0,1\}^{O(\log s)}}} \widehat{EQ}(u,a)\Big(\psi^{(i)}_{\text{add}}(a,b,c,z)\cdot(\ell_{i-1}(b)+\ell_{i-1}(c)) + \psi^{(i)}_{\text{mult}}(a,b,c,z)\ell_{i-1}(b)\cdot\ell_{i-1}(c)\Big).\end{aligned}$$

  where $\widehat{EQ}(u,a) := \prod_j (1 + u_j + a_j)$.
  - The prover and verifier then run two simultaneous sumcheck protocols using the polynomials g_{u_1}, g_{u_2}, where

  $$g_u(a,b,c,z) = \widehat{EQ}(u,a)\Big(\widetilde{\psi}^{(i)}_{\text{add}}(a,b,c,z)\cdot(\hat{\ell}_{i-1}(b)+\hat{\ell}_{i-1}(c)) + \widetilde{\psi}^{(i)}_{\text{mult}}(a,b,c,z)\hat{\ell}_{i-1}(b)\cdot\hat{\ell}_{i-1}(c)\Big)$$

  and the claimed outputs v_1, v_2. Importantly, the same verifier randomness is used for these two sumcheck protocols.
  - At the end of the interactive phase of this protocol, the verifier has a tuple of field elements $\boldsymbol{\beta} \in K^{3\log w + O(\log s)}$ and outputs γ_1, γ_2 such that (allegedly) $g_{u_1}(\boldsymbol{\beta}) = \gamma_1$ and $g_{u_2}(\boldsymbol{\beta}) = \gamma_2$. Let u_1', u_2' denote the part of β corresponding to b and c.
  - Finally, the prover sends $v_1' = \hat{\ell}_{i_1}(u_1'), v_2' = \hat{\ell}_{i-1}(u_2')$ to the verifier. Since $\widehat{EQ}, \widetilde{\psi}^{(i)}_{\text{add}}, \widetilde{\psi}^{(i)}_{\text{mult}}$ are all computable in time poly$(\log s)$, the verifier can check that v_1' and v_2' are consistent with the claims output by the sumcheck protocol. This completes the recursive step, which has produced two new claims $(u_1', v_1'), (u_2', v_2')$.
- After this recursive process, the verifier has obtained two final claims "$\hat{\ell}_0(u_1) = v_1$," "$\hat{\ell}_0(u_2) = v_2$" about the multilinear extension $\hat{\ell}_0$. Since $\hat{\ell}_0$ is nothing more than the multilinear extension of the input x (thought of as a function mapping $\{0,1\}^{\log n} \to \{0,1\}$), the verifier can check these two claims (given x) using $O(n)$ field operations.

Crucially, $\widetilde{\psi}_{\text{add}}$ and $\widetilde{\psi}_{\text{mult}}$ have individual degree 2, which implies that every polynomial g_u has individual degree **at most** 3. This is because $\widehat{EQ}(u,a)(\hat{\ell}_{i-1}(b)+\hat{\ell}_{i-1}(c))$ and $\widehat{EQ}(u,a)(\hat{\ell}_{i-1}(b)\hat{\ell}_{i-1}(c))$ are both multilinear polynomials.

This completes our description of our variant of the [GKR08] protocol. By combining Theorems 2.3, 2.6 and 2.9, the fact that this [GKR08] variant runs (pairs of) degree 3 sumchecks, and Theorem 3.8, we conclude Theorem 5.1.

References

[AMM77] Leonard Adleman, Kenneth Manders, and Gary Miller. On taking roots in finite fields. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 175–178. IEEE Computer Society, 1977. 18

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001. 1

[BBH$^+$19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 522–551. Springer, Heidelberg, December 2019. 1

[BCH$^+$22] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D Rothblum. Ppad is as hard as iterated squaring and lwe. In *TCC 2022*, 2022. https://eprint.iacr.org/2022/1272. 8

[Ber70] Elwyn R Berlekamp. Factoring polynomials over large finite fields. *Mathematics of computation*, 24(111):713–735, 1970. 3, 6

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 738–767. Springer, Heidelberg, August 2020. 3, 7

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015. 8

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 1, 3

[BSS99] Ian Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic curves in cryptography*, volume 265. Cambridge university press, 1999. 16

[CCH$^+$19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 1, 2, 3, 7, 8

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 389–415. Springer, Heidelberg, January 2016. 7

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018. 7

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009. 2

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998. 1, 3, 11

[CHK$^+$19a] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019. 1, 2, 4, 8, 19

[CHK$^+$19b] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. https://eprint.iacr.org/2019/667. 8

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg. 7, 8

[CJJ22] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for p from lwe. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79. IEEE, 2022. 1, 7, 8

[CZ81] David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981. 3, 6

[DGP09] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. 2

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–154. Springer, 2020. 8

[FGK$^+$10] David Mandell Freeman, Oded Goldreich, Eike Kiltz, Alon Rosen, and Gil Segev. More constructions of lossy and correlation-secure trapdoor functions. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 279–295. Springer, Heidelberg, May 2010. 3, 10

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 1, 3

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003. 1

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008. 1, 2, 5, 8, 12, 13, 21, 23

[Gol18] Oded Goldreich. On doubly-efficient interactive proof systems. *Foundations and Trends® in Theoretical Computer Science*, 13(3):158–246, 2018. 6, 20, 21

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016. 8

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 1

[GZ21] Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 529–557. Springer, Heidelberg, November 2021. 8

[HAB02] William Hesse, Eric Allender, and David A Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. 15, 18

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Heidelberg, May / June 2022. 1, 7, 8

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th FOCS*, pages 850–858. IEEE Computer Society Press, October 2018. 7

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D Rothblum. Fiat–shamir via list-recoverable codes (or: parallel repetition of gmw is not zero-knowledge). In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 750–760, 2021. 3, 7

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 672–683. Springer, 2006. 4, 6, 7, 13, 14, 15

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 3–32. Springer, Heidelberg, October 2021. 1, 3, 4, 7, 8, 9, 11, 21

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. 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*, pages 708–721, 2021. 1, 2, 3, 5, 8, 10, 12, 13, 19, 20, 21

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021. 8

[KLVW22] Yael Tauman Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. 2022. https://eprint.iacr.org/2022/1320. 1, 8

[KPY18] Yael Kalai, Omer Paneth, and Lisa Yang. On publicly verifiable delegation from standard assumptions. Cryptology ePrint Archive, Report 2018/776, 2018. https://eprint.iacr.org/2018/776. 20, 21

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 1, 8

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 652–673. Springer, Heidelberg, August 2020. 1, 8

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017. 7

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021. 8

[Lag70] Joseph-Louis Lagrange. Reflexions sur la resolution algebrique des equations, nouveaux memoires de l'acade. *Royale des sciences et belles-letteres, avec l'histire pour la meme annee*, 1:134–215, 1770. 7, 17

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990. 2

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 632–651. Springer, Heidelberg, August 2020. 8

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 1

[Pap94] Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994. 2

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019. 3, 7, 8

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008. 3, 9, 10, 21

[Rab80] Michael O Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on computing*, 9(2):273–280, 1980. 3, 6

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016. 1

[Tov84] Craig A Tovey. A simplified np-complete satisfiability problem. *Discrete applied mathematics*, 8(1):85–89, 1984. 4

[WW22] Brent Waters and David J Wu. Batch arguments for np and more from standard bilinear group assumptions. In *Proceedings of CRYPTO 2022*, 2022. 1, 8