ABSTRACT

Title of dissertation: NEW (ZERO-KNOWLEDGE) ARGUMENTS AND THEIR APPLICATIONS TO VERIFIABLE COMPUTATION

Yupeng Zhang
Doctor of Philosophy, 2018

Dissertation directed by: Professor Charalampos Papamanthou
Department of Electrical and Computer Engineering

Professor Jonathan Katz
Department of Computer Science

We study the problem of *argument* systems, where a computationally weak verifier outsources the execution of a computation to a powerful but untrusted prover, while being able to validate that the result was computed correctly through a proof generated by the prover. In addition, the *zero-knowledge* property guarantees that proof leaks no information about the potential secret input from the prover. Existing efficient zero-knowledge arguments with sublinear verification time require an expensive preprocessing phase that depends on a particular computation, and incur big overhead on the prover time and prover memory consumption.

This thesis proposes new constructions for zero-knowledge arguments that overcome the above problems. The new constructions require only a one time preprocessing and can be used to validate any computations later. They also reduce the overhead on the prover time and memory by orders of magnitude. We apply

our new constructions to build a verifiable database system and verifiable RAM programs, leading to significant improvements over prior work.

NEW (ZERO-KNOWLEDGE) ARGUMENTS AND THEIR APPLICATIONS TO VERIFIABLE COMPUTATION

A dissertation presented
by

Yupeng Zhang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Charalampos Papamanthou, Chair/Advisor
Professor Jonathan Katz, Co-Advisor
Professor Dana Dachman-Soled
Professor Gang Qu
Professor Lawrence Washington

Acknowledgments

I would like to begin by thanking my advisors Charalampos (Babis) Papamanthou and Jonathan Katz. Babis recruited me in my first year of PhD and led me to the exciting area of applied cryptography. He always made himself available to answer my questions and discuss the research projects. We usually had several meetings a week, through which I learnt quickly both the theory and the practice of cryptography. He always believed in my ability and encouraged me to come up with better solutions for many research problems, some of which led to papers included in this thesis. He provided great help for me on research, paper writing, presentation and gave me good advice for my future research career. I could not have had such an enjoyable life during my PhD without his guidance.

Jonathan's course on cryptography was the first time I got to know the area. I developed strong interest and fundamental skills during the course and I always believe it is the most important course for my PhD research. During our weekly meetings, he always understood the problems deep and fast, challenged my ideas and made helpful suggestions based on his broad knowledge and rich experience in the research areas. These were essential for me to develop thorough and rigorous solutions in many work. It is my great fortune to have him as my co-advisor. I am also very grateful that my advisors provided full financial support for my PhD and travels to conferences and workshops, so that I can focus on my research, and had opportunities to interact with leading researchers in the field.

I also want to thank the mentors of my summer internships, Nikos Trian-

dopoulos, Payman Mohassel, Ranjit Kumaresan and Melissa Chase. I spent several wonderful summers with them working on various topics that were different from my research at the graduate school. These great experiences helped my learn knowledge and develop interest on other areas such as secure computation and machine learning, and I also enjoyed the opportunities to live and work at different locations, RSA at Boston, Visa Research at the Bay area and MSR at Seattle.

I am also thankful to my collaborators and lab-mates. Dimitris and Daniel provided great help developing the ideas and writing several papers included in this thesis. I want to thank Xiao, Ahmed and Kartik for having interesting discussions on research and life, and senior students Alex, Aishwarya, Chang and Andrew for their advice and help during the early years of my PhD.

Finally, I would like to thank my parents. I would not have started my PhD without their important advice and they have always been showing love, understanding and support during my PhD. I also want to thank my girlfriend, Xi, for her accompany during my graduate school. We have visited many great places all around the world and she made my life outside research wonderful during my PhD.

Table of Contents

Acknowledgements ii

1 Introduction 1
1.1 Related Work . . . 3

2 Preliminaries 7
2.1 Bilinear Pairings and Assumptions . . . 7
2.2 Circuit and Polynomial Notations . . . 9
2.3 Interactive Proofs . . . 11
2.3.1 The Sum-Check Protocol . . . 11
2.3.2 The CMT Protocol . . . 13
2.4 Security Definitions . . . 17

3 Constructions 20
3.1 Verifiable Polynomial Delegation . . . 20
3.2 Improving the Expressiveness of the CMT Protocol . . . 30
3.3 The Construction of Our Argument System . . . 32

4 Applications: Verifiable Databases 41
4.1 SQL Queries . . . 43
4.2 Related Work on Verifiable Databases . . . 45
4.3 Definitions of Verifiable Databases . . . 46
4.4 Our Construction of Verifiable Databases . . . 49
4.5 Optimizations for SQL Queries . . . 61
4.5.1 Optimizing Equality Testing . . . 62
4.5.2 Supporting Inputs/Outputs at Arbitrary Circuit Layers . . . 65
4.5.3 Verifying Set Intersections . . . 67
4.5.4 Supporting Expressive Updates . . . 71
4.5.5 Efficient Value Insertions . . . 72
4.6 Experimental Results . . . 73
4.6.1 Experimental Setup . . . 73
4.6.2 Performance Comparison: Selection Queries . . . 76

4.6.3 Performance Comparison: Update Queries . . . 82
4.6.4 Scalability of Our Construction . . . 84
4.6.5 Microbenchmarks . . . 85

5 Applications: Verifiable RAM Programs 87
5.1 Preliminaries on RAM programs . . . 90
5.1.1 A Canonical RAM Architecture . . . 90
5.1.2 Previous Reductions from RAM to Circuit Satisfiability . . . 92
5.2 Our New RAM to Circuit Reduction . . . 96
5.2.1 Ensuring Correct Instruction Execution . . . 97
5.2.2 Verifying Instruction Fetches . . . 98
5.2.3 Ensuring Memory Accesses . . . 100
5.2.4 Checking Consistency . . . 101
5.3 Experimental Results . . . 103
5.3.1 Comparison with vnTinyRAM and Buffet . . . 104
5.3.2 Comparison to Other RAM-based VC systems . . . 113
5.3.3 Just-in-Time Architecture . . . 115
5.3.4 Microbenchmarks . . . 116

6 Zero Knowledge 119
6.1 Building Blocks . . . 120
6.2 Zero-Knowledge Polynomial Commitment . . . 123
6.3 Zero-Knowledge CMT Protocol . . . 136
6.3.1 A Sum-Check Protocol over Homomorphic Commitments . . . 136
6.3.2 A CMT Protocol over Homomorphic Commitments . . . 142
6.4 Zero-Knowledge with Function Independent Preprocessing . . . 152

7 Conclusions and Future Work 160
7.1 Conclusions . . . 160
7.2 Future Directions . . . 161

Bibliography 163

Chapter 1: Introduction

With the advent of cloud computing, there has been significant interest in techniques for ensuring correctness of computations performed by an untrusted server on behalf of a client. Protocols for *verifiable computation (VC)* allow a computationally weak verifier to outsource the execution of a computation to a powerful but untrusted prover (e.g., a cloud provider) while being assured that the result was computed correctly. Somewhat more formally, a verifier $\mathcal{V}$ and prover $\mathcal{P}$ agree on a function f and an input x. The prover then sends a result y to the verifier, together with a proof that $y = f(x)$, and the verifier can validate that the result is indeed correctly computed. In addition, it is particularly interesting when the time to validate the result on the verifier side is less than the time to compute $f(x)$ on its own.

VC protocols can be constructed from succinct *argument systems*, where the prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ the validity of a statement, and the proof size and verification time are smaller than the statement itself. See Section 2.4 for formal definitions of argument systems. There is a long line of work constructing VC protocols from argument systems for arbitrary computations, the most prominent of which rely on succinct non-interactive arguments of knowledge (SNARKs) [20,46].

This has resulted in several implemented systems; see Section 1.1 for an overview. However, SNARKs rely on a *preprocessing* phase, where a trusted party (possibly the verifier) generates a set of public parameters corresponding to a circuit for a specific function f. This preprocessing phase is orders of magnitude slower than evaluating f itself, and can only be used later to prove and verify the results of the same function f on different inputs. In addition, these SNARK-based protocols introduce a big overhead on the running time and the memory consumption of the prover to generate the proof.

In this thesis, we propose a new construction of a argument system for arbitrary computations. At a high level, our construction builds on top of prior work in *interactive proofs (IP)*, and combines them with a *verifiable polynomial delegation (VPD)* scheme to extend the supported class of computations from P to NP, allowing computations to take auxiliary inputs from the prover without sending them back to the verifier. Compared to existing SNARK-based protocols, the preprocessing phase of our construction is independent of the function and can be used to validate any computation later. Our construction also reduces the overhead on the prover time and memory consumption by orders of magnitude. We apply our new argument system to build VC protocols for verifiable databases and verifiable RAM programs, and show that they lead to significant improvements upon prior work. We also present a variant of our new argument that is zero-knowledge.

1.1 Related Work

Verifiable computation was formalized in [45, 72], but research on constructing interactive protocols for verifying general-purpose computations began much earlier with the works of Kilian [57] and Micali [66]. While those works have good asymptotic performance, and follow-up works further optimized those approaches (e.g., [10, 13, 54]), subsequent implementations revealed that the concrete costs of those approaches are prohibitively high for the prover [75].

SNARKs. The next big breakthrough in general-purpose verifiable computation and argument systems came with the work of Gennaro et al. [46] (building upon earlier work by Groth [50] and Lipmaa [63]), which introduced quadratic arithmetic programs (QAPs) and showed that they can be used to capture the correct evaluation of an arithmetic program. QAPs have since been the de-facto tool for constructing efficient succinct arguments of knowledge (SNARKs) [20, 23] that can be used to verify arbitrary NP computations. This has led to a long line of research providing both highly-optimized systems [16, 35, 37, 41, 61, 71, 74, 76, 81, 93] and significant protocol refinements [17, 39, 52, 64]. We refer to [85] for a detailed survey.

The major disadvantage of SNARK-based approaches is the extremely high prover time they currently impose. The prover time is $O(m \log^2 m)$, with $O(m)$ cryptographic operations (modulo exponentiations in a bilinear group) where m is the number of gates in a arithmetic circuit for a particular function. In our new argument system, the prover time is $O(m \log m)$ ($O(m)$ for highly regular circuits)

with only $O(n)$ cryptographic operations, where n is the number of inputs and usually much less than m.

In addition, the fastest existing implementations of SNARKs assume a circuit-specific preprocessing step, something that is not practical (and may be impossible) in a scenario where multiple queries that cannot be predicted in advance will be made on a given data. In contrast, our new construction requires only a one-time circuit-independent preprocessing, and can later be used to validate arbitrary computations.

Finally, we remark that the systems mentioned above are all "natively" designed to support verification only when the input is known to the verifier, which is enough for argument systems as defined in Section 2.4. However, in verifiable computation, the verifier also wants to outsource the storage of the input to the prover. Support for outsourced data can be handled by having the verifier compute a succinct hash of its data, and then verifying the hash computation along with verification of the result. However, this adds additional overhead as the hash computation needs to either be computed as part of the arithmetic circuit [29], or checked by an external mechanism [12, 41]. Alternatively, one could hard-code the data into the circuit being evaluated, but then the circuit-specific preprocessing needs to be executed after each data update. Instead, our new argument system naturally supports outsourcing data through a commitment and we can also support dynamic data efficiently.

While some other works also aim at verifying arbitrary computations over remotely stored data [27, 31, 33, 55], these approaches are only of theoretical interest at this point.

Interactive Proofs. Interactive proofs were introduced by Goldwasser et al. [49], and have been studied extensively in complexity theory. More recently, prover-efficient interactive proofs for log-depth circuits were introduced in [48]. Subsequent works have optimized and implemented this protocol, demonstrating the potential of interactive proofs for practical verifiable computation [36, 78, 81].

Structure of the Thesis:

- Chapter 2 introduces background on bilinear groups, circuits, polynomials and the details of the sum-check protocol and the interactive proof protocol of Cormode, Mitzenmacher, and Thaler [36] (that we call the *CMT protocol*) . In addition, it formally defines an argument system for verifiable computation and its correctness, soundness and zero-knowledge.

- Chapter 3 presents the detailed construction of our new argument system for verifiable computation. We first show our verifiable polynomial delegation scheme, and an improvement of the CMT protocol that supports a larger class of computation efficiently. Then we present our new argument system combining the VPD and the CMT protocol, followed by the security proofs and complexity analysis.

- In Chapter 4, we present a verifiable database system, vSQL, using our new argument protocol. We introduce background on SQL queries, definitions, and our construction of a verifiable database system. We also show several optimizations of our argument system for common SQL queries, followed by experimental results, showing that vSQL improves the prover time by two orders of magnitude and can be used to validate arbitrary SQL queries [88].

- Chapter 5 describes our construction of verifiable RAM programs using our new argument system as a backend. By utilizing the function-independent preprocessing feature of our argument system, we propose a tighter RAM-to-circuit reduction, and show that the prover time and the memory usage are improved by up to two orders of magnitude upon prior work [90].

- In Chapter 6, we present a variant of our argument system that is zero-knowledge. We show the construction of a zero-knowledge version of the VPD protocol, and show how to run the interactive proof protocol on commitments without leaking intermediate values. We give formal proofs for soundness and zero-knowledge of the construction [89].

- Chapter 7 concludes the thesis, gives a more detailed survey on related work in the literature of verifiable computation, zero-knowledge proofs, verifiable databases and verifiable RAM programs, and describes future research directions.

Chapter 2: Preliminaries

In this chapter, we present background material on bilinear maps, circuits, polynomials and interactive proofs.

We use λ to denote the security parameter. For a multivariate polynomial f, the degree of each monomial in f is the sum of the powers of its variables; the total degree of f is the maximum degree of any of its monomials.

2.1 Bilinear Pairings and Assumptions

We denote by $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ generation of bilinear-map parameters, where $\mathbb{G}$, $\mathbb{G}_T$ are groups of prime order p, with g a generator of $\mathbb{G}$, and where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficient map, i.e., for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$ it holds that $e(P^a, Q^b) = e(P, Q)^{ab}$. For simplicity we assume symmetric pairings in this thesis, but our scheme can be adapted to use asymmetric pairings as well.

Let PPT stand for "probabilistic polynomial-time". We rely on the following cryptographic assumptions.

Assumption 1 ([24] q-Strong Bilinear Diffie-Hellman (q-SBDH)). *For any* PPT

adversary Adv*, the following probability is negligible:*

$$\Pr\left[\begin{array}{c} (p,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow \mathsf{BilGen}(1^\lambda); \\ s\xleftarrow{R}\mathbb{Z}_p^*; \\ \sigma=((p,\mathbb{G},\mathbb{G}_T,e,g),g^s,\ldots,g^{s^q}); \\ (a,h)\leftarrow \mathsf{Adv}(1^\lambda,\sigma); \end{array} : h=e(g,g)^{\frac{1}{s+a}}\right].$$

We use $\mathcal{W}_{\ell,d}$ to denote the collection of all multisets of $\{1,\ldots,\ell\}$ where the cardinality of each element is at most d. The next assumption states the following. Assume a polynomial time algorithm that receives as input two ordered sequences of elements of $\mathbb{G}$ such that each element contains in the exponent a multivariate monomial with at most ℓ variables and of total degree at most $\ell \cdot d$, for some d, and for every ordered pair of elements (across the two sequences) it holds that the elements differ in the exponent by a fixed multiplicative factor α. Then, if the party outputs a new pair of elements that differ in the exponent by α, then it must hold that the first of these two elements was computed as a linear combination of the elements of the first sequence (and likewise for the second and the same linear combination). This fact is captured by the existence of a polynomial-time extractor ε that, upon the same input outputs this linear combination.

This knowledge-type assumption is a direct generalization of Groth's q-PKE assumption [51] for the case of multivariate polynomials. In fact, q-PKE is by definition the same as $(1,q)$-PKE, using our notation. Note that $\mathcal{W}_{\ell,d}$ has size $O(\binom{\ell+\ell d-1}{\ell d})$[1]. In our construction, we will be using this assumption for the case where $\binom{\ell+\ell d-1}{\ell d} = \mathsf{poly}(\lambda)$. The results of [22,28] show the impossibility of knowledge

[1]This is the number of multisets of cardinality ℓd, with elements taken from a set of ℓ elements.

assumptions with respect to arbitrary auxiliary inputs. In the following definition we use the notion of a benign auxiliary input (or, alternatively, a benign state generator), similar to [37, 41, 53], to refer to auxiliary inputs that make extraction possible, avoiding these negative results. Concretely, our proofs hold assuming the auxiliary input of the extractor comes from a benign distribution.

Assumption 2 ((d,ℓ)-Power Knowledge of Exponent (PKE))**.** *For any* PPT *adversary* Adv *there is a polynomial-time algorithm* $\mathcal{E}$ *(running on the same random tape) such that for all benign auxiliary inputs* $z \in \{0,1\}^{poly(\lambda)}$ *the following probability is negligible:*

$$\Pr\left[\begin{array}{c} (p,\mathbb{G},\mathbb{G}_T,e,g) \leftarrow \mathsf{BilGen}(1^\lambda); \tau_1,\ldots,\tau_\ell,\alpha \xleftarrow{R} \mathbb{Z}_p^*; \\ \sigma = (p,\mathbb{G},\mathbb{G}_T,e,\{g^{\prod_{i\in W}\tau_i},\ g^{\alpha\cdot\prod_{i\in W}\tau_i}\}_{W\in\mathcal{W}_{\ell,d}},g^\alpha); \\ \mathbb{G}\times\mathbb{G}\ni(h,\tilde{h}) \leftarrow \mathsf{Adv}(1^\lambda,\sigma,z); \\ (a_0,\ldots,a_{|\mathcal{W}_{\ell,d}|}) \leftarrow \mathcal{E}(1^\lambda,\sigma,z); \end{array} : \begin{array}{c} e(h,g^\alpha) = e(\tilde{h},g) \\ \wedge \\ \prod_{W\in\mathcal{W}_{\ell,d}} g^{a_W\prod_{i\in W}\tau_i} \neq h \end{array}\right].$$

2.2 Circuit and Polynomial Notations

An *arithmetic circuit* C is a directed acyclic graph whose vertices are called *gates* and whose edges are called *wires*. Every in-degree 0 gate in C is labeled by a variable from a set of variables $X = \{x_1,\cdots,x_n\}$ and is referred to as an input gate. All other gates in C have in-degree 2, are labeled by elements from $\{+,\times\}$, and referred to as addition and multiplication gates, respectively. Every gate of out-degree 0 is called an output gate. In the following, we focus only on *layered* circuits and we assume that the output gates are ordered. We say that a circuit is

layered if it can be divided into disjoint sets $L_1, \cdots, L_k$ such that every gate of g belongs to some set L_i and all the wires of C connect gates in two consecutive layers (i.e., between L_{j+1} and L_j for some j). We write $C : \mathbb{F}^n \to \mathbb{F}^m$ to indicate that C is an arithmetic circuit with n inputs and m outputs evaluated (as defined in a natural way) over a field $\mathbb{F}$. We denote by $|C|$ the number of gates in the circuit C, by $\mathsf{width}_i(C)$ the number of gates in the i-the layer of C and by $\mathsf{width}(C)$ the maximum width of C, i.e., $\mathsf{width}(C) = \max_i\{\mathsf{width}_i(C)\}$.

Polynomial decomposition. We use the following lemma for polynomial decomposition.

Lemma 1 ([69]). *Let $f : \mathbb{F}^\ell \to \mathbb{F}$ be a polynomial. For all $t \in \mathbb{F}^\ell$ there exist efficiently computable polynomials $q_1, \ldots, q_\ell$ such that: $f(x) - f(t) = \sum_{i=1}^{\ell}(x_i - t_i)q_i(x)$ where t_i is the ith element of t.*

Multilinear extensions. For any function $V : \{0,1\}^\ell \to \mathbb{F}$ we define the multilinear extension, $\widetilde{V} : \mathbb{F}^\ell \to \mathbb{F}$, of V as follows:

$$\widetilde{V}(x_1, \cdots, x_\ell) = \sum_{b \in \{0,1\}^\ell} \prod_{i=1}^{\ell} \mathcal{X}_{b_i}(x_i) V(b) \tag{2.1}$$

where b_i is the i-th bit of b, $\mathcal{X}_1(x_i) = x_i$ and $\mathcal{X}_0(x_i) = 1 - x_i$. Note that $\widetilde{V}$ is the unique polynomial that has degree at most 1 in each of its variables that satisfies $\widetilde{V}(x) = V(x)$ for all $x \in \{0,1\}^\ell$.

Multilinear extensions of arrays. An array $A = (a_0, \cdots, a_{n-1})$ where $a_i \in \mathbb{F}$ can be viewed as a function $A : \{0,1\}^{\log n} \to \mathbb{F}$ such that $A(i) = a_i$ for all $0 \leq i \leq n-1$. In the sequel, we abuse terminology by defining (in the natural way) the multilinear

extension $\tilde{A}$ of an array A.

2.3 Interactive Proofs

An interactive proof [49] is a protocol that allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of the validity of some statement. We phrase this in terms of $\mathcal{P}$ trying to convince $\mathcal{V}$ that $f(x) = 1$, where f is fixed and x is the common input. Of course, an interactive proof in this sense is only interesting if the running time of $\mathcal{V}$ is less than the time to compute f. Let $\langle \mathcal{P}, \mathcal{V} \rangle(x)$ denote the output of $\mathcal{V}$ after the interactions with $\mathcal{P}$ on input x.

Definition 1. *Let f be a boolean function. A pair of interactive algorithms $(\mathcal{P}, \mathcal{V})$ is an* interactive proof *for f with soundness ϵ if the following holds.*

- ***Completeness:*** *For every x such that $f(x) = 1$ it holds that*

$$\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = 1] = 1.$$

- ***ϵ-Soundness:*** *For any x with $f(x) \neq 1$ and any $\mathcal{P}^*$ it holds that*

$$\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(x) = 1] \leq \epsilon.$$

Note that the above can be easily extended to prove that $g(x) = y$ (where x, y are common input) by considering the function f defined as $f(x, y) = 1$ iff $g(x) = y$.

2.3.1 The Sum-Check Protocol

A fundamental interactive protocol that serves as an important building block for our work is the *sum-check protocol* [65]. Here, the common input of the prover

and verifier is an ℓ-variate polynomial $g(x_1, \ldots, x_\ell)$ over a field $\mathbb{F}$; the prover's goal is to convince the verifier that

$$H = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, b_2, \ldots, b_\ell)\,.$$

Note that direct computation of H by $\mathcal{V}$ requires at least 2^ℓ work. Using the sum-check protocol, the verifier's computation is exponentially smaller. The protocol proceeds in ℓ rounds, as follows. In the first round, the prover sends the univariate polynomial $g_1(x_1) \stackrel{\text{def}}{=} \sum_{b_2,\ldots,b_\ell \in \{0,1\}} g(x_1, b_2, \ldots, b_\ell)$; the verifier checks that the degree of g_1 is at most the degree of x_1 in g, and that $H = g_1(0) + g_1(1)$; it rejects if these do not hold. Next, $\mathcal{V}$ sends a uniform challenge $r_1 \in \mathbb{F}$. In the ith round $\mathcal{P}$ sends the polynomial $g_i(x_i) \stackrel{\text{def}}{=} \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} g(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell)$. The verifier checks the degree of g_i and verifies that $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$; if so, it sends a uniform $r_i \in \mathbb{F}$ to the prover. After the final round, $\mathcal{V}$ accepts only if $g(r_1, \ldots, r_\ell) = g_\ell(r_\ell)$. We have [65]:

Theorem 1. *For any ℓ-variate, total-degree-d polynomial g over $\mathbb{F}$, the sum-check protocol is an interactive proof for the function $f(H) = 1$ (where $f(H) = 1$ iff $\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell) = H$) with soundness $d \cdot \ell / |\mathbb{F}|$. Moreover, $\mathcal{V}$ performs $\text{poly}(\ell)$ arithmetic operations over $\mathbb{F}$ and one evaluation of g on a random point r.*

Remark 1. *When g is a multilinear polynomial (the degree of each variable is at most 1, and the total degree is ℓ), the running time of $\mathcal{P}$ in round i of the sum-check protocol is $\min\{O(m), O(2^{\ell-i})\}$, where m is the total number of distinct monomials in g [36, 78, 81].*

2.3.2 The CMT Protocol

Cormode et al. [36, 80], building on work of Goldwasser et al. [48], show an efficient interactive proof for a certain class of functions.

High-level overview. Let C be a depth-d layered arithmetic circuit over a finite field $\mathbb{F}$. The CMT protocol processes the circuit one layer at a time, starting from layer 0 (that contains the output wires) and ending at layer d (that contains the input wires). The prover $\mathcal{P}$ starts by proposing a value y for the output of the circuit on input x. Then, in the ith round, $\mathcal{P}$ reduces a claim (i.e., an algebraic statement) about the values of the wires in layer i to a claim about the values of the wires in layer $i+1$. The protocol terminates with a claim about the wire values at layer d (i.e., the input wires) that can be checked directly by the verifier $\mathcal{V}$ who knows the input x. If that check succeeds, then $\mathcal{V}$ accepts.

Notation. Before describing the protocol more formally we introduce some additional notation. Let S_i be the number of gates in the ith layer and set $s_i = \lceil \log S_i \rceil$ so s_i bits suffice to identify each gate at the ith layer. The evaluation of C on an input x assigns in a natural way a value in $\mathbb{F}$ to each gate in the circuit. Thus, for each layer i we can define a function $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ that takes as input a gate g and returns its value (and returns 0 if g does not correspond to a valid gate). Using this notation, V_d corresponds to the input of the circuit, i.e., x. Finally, we define for each layer i two boolean functions $\mathsf{add}_i, \mathsf{mult}_i$, which we refer to as *wiring predicates*, as follows: $\mathsf{add}_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$ takes as input three gates g_1, g_2, g_3, where g_1 is at layer $i-1$ and g_2, g_3 are at layer i, and returns 1 if and only if g_1 is an

addition gate whose input wires are the output wires of gates g_2 and g_3. (We define mult_i for multiplication gates analogously.) The value of a gate g at layer $i < d$ can thus be recursively computed as

$$V_i(g) = \sum_{u,v \in \{0,1\}^{s_{i+1}}} \Big(\mathsf{add}_{i+1}(g,u,v) \cdot (V_{i+1}(u) + V_{i+1}(v)) + \mathsf{mult}_{i+1}(g,u,v) \cdot (V_{i+1}(u) \cdot V_{i+1}(v)) \Big).$$

Protocol details. One idea is for $\mathcal{V}$ to verify that $y = C(x)$ by checking that $V_i(g)$ is computed correctly for each gate g in each layer i. Since $V_i(g)$ can be expressed as a summation, this could be done using the sum-check protocol from Section 2.3.1. However, the sum-check protocol operates on polynomials defined over $\mathbb{F}$ and therefore we need to replace terms with their multilinear extensions. That is:

$$\begin{aligned} \tilde{V}_i(z) &= \sum_{\substack{g \in \{0,1\}^{s_i} \\ u,v \in \{0,1\}^{s_{i+1}}}} f_{i,z}(g,u,v) \qquad (2.2) \\ &\stackrel{\text{def}}{=} \sum_{\substack{g \in \{0,1\}^{s_i} \\ u,v \in \{0,1\}^{s_{i+1}}}} \tilde{\beta}_i(z,g) \cdot \Big(\tilde{\mathsf{add}}_{i+1}(g,u,v) \cdot (\tilde{V}_{i+1}(u) + \tilde{V}_{i+1}(v)) + \tilde{\mathsf{mult}}_{i+1}(g,u,v) \cdot (\tilde{V}_{i+1}(u) \cdot \tilde{V}_{i+1}(v)) \Big), \end{aligned}$$

where $\tilde{\mathsf{add}}_i$ (resp., $\tilde{\mathsf{mult}}_i$) is the multilinear extension of add_i (resp., mult_i) and $\tilde{\beta}_i$ is the multilinear extension of the selector function that takes two s_i-bit inputs a, b and outputs 1 if $a = b$ and 0 otherwise.[2] However, this approach would incur a cost to the verifier larger than the cost of evaluating C, as it requires one execution of the sum-check protocol per gate.

[2] Although using $\tilde{\beta}$ is not strictly necessary here [79], we use it in our construction to improve efficiency when C is composed of many parallel copies of different smaller circuits in Section 3.2.

Instead, by leveraging the recursive form of $\tilde{V}_i$, correctness of the circuit evaluation can be checked with a single execution of the sum-check protocol for each layer i, as follows. Assume for simplicity that the output of the circuit is a single value. The interaction begins at level 0, with the prover claiming that $y = \tilde{V}_0(0)$ (i.e., the circuit's output) for some value y. The two parties then execute the sum-check protocol for the polynomial $f_{0,0}$ in order to check this claim. Recall that, at the end of this execution, $\mathcal{V}$ is supposed to evaluate $f_{0,0}$ at a random point $\rho \in \mathbb{F}^{s_0+2s_1}$ (the randomness generated by the sum-check verifier). Since $f_{0,0}$ depends on $\tilde{V}_1(u)$ and $\tilde{V}_1(v)$, in this case $\mathcal{V}$ has to evaluate $\tilde{V}_1$ on the random points $q_1, q_2 \in \mathbb{F}^{s_1}$ where q_2 consists of the last s_1 entries of ρ, and q_1 the previous s_1 entries. If the verifier had access to all the correct gate values at layer 1, it could compute these evaluations himself. Since he does not, however, it must rely on the prover to provide it with these evaluations, say v_1, v_2. This effectively reduces the validity of the original claim that $y = \tilde{V}_0(0)$ to the validity of the two claims that $\tilde{V}_1(q_1) = v_1$ and $\tilde{V}_1(q_2) = v_2$. The two parties can now execute the sum-check protocol for these two claims. By repeatedly applying this idea, the final claim by the prover will be stated with respect to $\tilde{V}_d$ (i.e., the multilinear extension of the circuit's input), which can be checked locally by the verifier who has the input x.

Unfortunately, this approach still potentially requires 2^d executions of the sum-check protocol, since the number of claims being verified doubles with each level.

Condensing to a single evaluation per layer. Efficiency can be improved by reducing the proof that $v_1 = \tilde{V}_1(q_1)$ and $v_2 = \tilde{V}_1(q_2)$ to a *single* sum-check execution,

as follows. Let $\gamma : \mathbb{F} \to \mathbb{F}^{s_1}$ be the unique line with $\gamma(0) = q_1$ and $\gamma(1) = q_2$. The prover sends a degree-s_1 polynomial h that is supposed to be $\tilde{V}_1(\gamma(x))$, i.e., the restriction of $\tilde{V}_1$ to the line γ. The verifier checks that $h(0) = v_1$ and $h(1) = v_2$, and then picks a new random point $r'_1 \in \mathbb{F}$ and initiates a *single* invocation of the sum-check protocol to verify that $\tilde{V}_1(\gamma(r'_1)) = h(r'_1)$. Proceeding in this way, it is possible to obtain a protocol that uses only $O(d)$ executions of the sum-check protocol.

We assumed so far that there is a single output value y. Larger outputs can be handled efficiently [81] by adapting the above approach so that the initial claim by the prover is stated directly about the multilinear extension of the claimed output.

The CMT protocol is formally described in Construction 1.

Construction 1 (CMT protocol). *Let $\mathbb{F}$ be a prime-order field, and let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a depth-d layered arithmetic circuit. $\mathcal{P}$ and $\mathcal{V}$ hold x, y, and $\mathcal{P}$ wants to convince $\mathcal{V}$ that $y = C(x)$. To do so:*

1. *Let $V_0 : \{0,1\}^{\lceil \log k \rceil} \to \mathbb{F}$ be such that $V_0(j)$ equals the jth element of y. Verifier $\mathcal{V}$ chooses uniform $r_0 \in \mathbb{F}^{\lceil \log k \rceil}$ and sends it to $\mathcal{P}$. Both parties set $a_0 = \tilde{V}_0(r_0)$.*

2. *For $i = 1, \ldots, d$:*

   (a) *$\mathcal{P}$ and $\mathcal{V}$ run the sum-check protocol for value a_{i-1} and polynomial $f_{i-1,r_{i-1}}$ as per Equation (2.2). In the last step of that protocol, $\mathcal{P}$ provides (v_1, v_2) for which it claims $v_1 = \tilde{V}_i(q_1)$ and $v_2 = \tilde{V}_i(q_2)$.*

   (b) *Let $\gamma : \mathbb{F} \to \mathbb{F}^{s_i}$ be the line with $\gamma(0) = q_1$ and $\gamma(1) = q_2$. Then $\mathcal{P}$ sends the degree-s_i polynomial $h(x) = \tilde{V}_i(\gamma(x))$. Next, $\mathcal{V}$ verifies that $h(0) = v_1$ and*

$h(1) = v_2$, *and rejects if not. Then* $\mathcal{V}$ *chooses uniformly at random* $r'_i \in \mathbb{F}$, *sets* $r_i = \gamma(r'_i)$, $a_i = h(r'_i)$ *and sends them to* $\mathcal{P}$.

3. $\mathcal{V}$ *accepts iff* $a_d = \tilde{V}_d(r_d)$, *where* $\tilde{V}_d$ *is the multilinear extension of the polynomial representing the input* x.

Throughout the paper, when reporting asymptotic complexities we omit a factor that is polylogarithmic in the field/blinear group size, implicitly assuming all operations take constant time.

Theorem 2 ([36, 48, 78, 81])**.** *Let* $C : \mathbb{F}^n \to \mathbb{F}^k$ *be a depth-*d *layered arithmetic circuit. Construction 1 is an interactive proof for the function computed by* C *with soundness* $O(d \cdot \log S/|\mathbb{F}|)$, *where* S *is the maximal number of gates per circuit layer. It uses* $O(d \log S)$ *rounds of interaction, and the running time of* $\mathcal{P}$ *is* $O(|C| \log S)$. *If* $\tilde{\mathsf{add}}_i$ *and* $\tilde{\mathsf{mult}}_i$ *are computable in time* $O(\text{polylog}\, S)$ *for all layers* $i \leq d$, *then the running time of the verifier* $\mathcal{V}$ *is* $O(n + k + d \cdot \text{polylog}\, S)$.

Remark 2 ([78])**.** *If* C *can be expressed as a composition of (i) parallel copies of a layered circuit* C' *whose maximum number of gates at any layer is* S', *and (ii) a subsequent layered "aggregation" circuit* C'' *of size* $O(|C|/\log|C|)$, *the running time of* $\mathcal{P}$ *is* $O(|C| \log |S'|)$.

2.4 Security Definitions

In this section, we give the formal definitions of (zero-knowledge) argument systems. Let R be an NP relation. An argument system for R is a protocol between computationally bounded prover $\mathcal{P}$ and a verifier $\mathcal{V}$ at the end of which $\mathcal{V}$

is convinced in the validity of a statement made by $\mathcal{P}$ of the form "there exists w such that $(x; w) \in R$" for some input x. In the sequel we focus on *arguments of knowledge* which have the stronger property that if the prover manages to convince the verifier of the statement's validity, then the prover must know w. We use the definition of [46] which includes a parameter-generation phase executed by a trusted party, the preprocessor. Formally, consider Definition 2 below.

Definition 2. *Let R be an NP relation and let λ be a security parameter. A tuple of algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a zero knowledge* argument *for R if the following holds.*

- ***Completeness:*** *For every $(\mathsf{pk}, \mathsf{vk})$ outout by $\mathcal{G}(1^\lambda)$ and all $(x; w) \in R$ we have*

$$\langle \mathcal{P}(\mathsf{pk}, w), \mathcal{V}(\mathsf{vk}) \rangle(x) = 1.$$

- ***Knowledge soundness:*** *For any* PPT *prover $\mathcal{P}^*$ there exists a* PPT *extractor $\mathcal{E}$ which runs on the same randomness as $\mathcal{P}^*$ such that for any x it holds that*

$$\Pr[\langle \mathcal{P}^*(\mathsf{pk}, \mathsf{vk}), \mathcal{V}(\mathsf{vk}) \rangle(x) = 1 \wedge (x, w) \notin R : (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{G}(1^\lambda), w \leftarrow \mathcal{E}(\mathsf{pk}, x)] \leq \mathsf{neg}(\lambda).$$

- ***Zero knowledge:*** *There exists a* PPT *simulator $\mathcal{S}$ such that for any* PPT *adversary $\mathcal{A}$, and auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, the following probability is negligible:*

$$\Pr\left[(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{G}(1^\lambda); (x; w) \in R; \langle \mathcal{P}(\mathsf{pk}, w), \mathcal{A}(\mathsf{pk}, \mathsf{vk}) \rangle(x) = 1 : (x, w) \leftarrow \mathcal{A}(z, \mathsf{pk}, \mathsf{vk})\right] -$$

$$\Pr\left[(\mathsf{pk}, \mathsf{vk}, \mathsf{trap}) \leftarrow \mathcal{S}(1^\lambda); (x; w) \in R; \langle \mathcal{S}(\mathsf{trap}, \mathsf{pk}), \mathcal{A}(\mathsf{pk}, \mathsf{vk}) \rangle(x) = 1 : (x, w) \leftarrow \mathcal{A}(z, \mathsf{pk}, \mathsf{vk})\right]$$

the definition can be extended in a straight-forward manner for statistical and perfect zero-knowledge.

We call $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *a* succinct *argument system if the running time of* $\mathcal{V}$ *is* $poly(\lambda, |x|, \log |w|)$.

Chapter 3: Constructions

In this chapter, we present the basic construction of our new argument system for NP that is complete and sound. We defer the zero-knowledge version of our protocol to Chapter 6.

Recall that the CMT protocol, and interactive proof protocols as defined in Definition 1, only allow the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that $f(x) = 1$ for a fixed f and a common input x. It does not allow auxiliary input w to f, limiting the supported class of computation to P. To extend the interactive proof protocols to prove statements in NP, we propose a new scheme for *verifiable polynomial delegation*, which allows the prover $\mathcal{P}$ to commit a polynomial defined by w efficiently, and later open it to a random evaluation point, as required by the CMT protocol.

3.1 Verifiable Polynomial Delegation

In the last step of the CMT protocol, the verifier $\mathcal{V}_{cmt}$ evaluates a polynomial $\widetilde{V}_d$ on a random point r_d. Since the number of terms in $\widetilde{V}_d$ is equal to the number of input gates of C, this makes the verifier's work linear not only in the size of the input x but also the length of the witness w. We propose a verifiable polynomial delegation (VPD) scheme to address this problem. We give the formal definition of

a VPD scheme here.

Definition 3. *Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ a family of ℓ-variate polynomials over $\mathbb{F}$, and d a variable-degree parameter.* $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Evaluate}, \mathsf{Ver})$ *constitute an* extractable VPD scheme *for $\mathcal{F}$ if:*

- ***Perfect completeness.*** *For any polynomial $f \in \mathcal{F}$ it holds that*

$$\Pr\left[\begin{array}{lc} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d); & \mathsf{Ver}(\mathsf{com}, t, y, \pi, \mathsf{vp}) = 1 \\ \mathsf{com} \leftarrow \mathsf{Commit}(f, \mathsf{pp}); & : \quad \wedge \\ (y, \pi) \leftarrow \mathsf{Evaluate}(f, t, \mathsf{pp}); & y = f(t) \end{array}\right] = 1.$$

- ***Soundness.*** *For any* PPT *adversary* Adv *the following probability is negligible:*

$$\Pr\left[\begin{array}{lc} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d); & \mathsf{Ver}(\mathsf{com}, t^*, y^*, \pi^*, \mathsf{vp}) = 1 \\ (f^*, t^*, y^*, \pi^*) \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{pp}); & : \quad \wedge \\ \mathsf{com} \leftarrow \mathsf{Commit}(f^*, \mathsf{pp}); & y^* \neq f^*(t^*) \end{array}\right].$$

- ***Extractability.*** *For any* PPT *adversary* Adv *there exists a polynomial-time algorithm $\mathcal{E}$ with access to Adv's random tape such that for all benign auxiliary inputs $z \in \{0,1\}^{poly(\lambda)}$ the following probability is negligible:*

$$\Pr\left[\begin{array}{lc} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d); & \mathsf{CheckCom}(\mathsf{com}^*, \mathsf{vp}) = 1 \\ \mathsf{com}^* \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{pp}, z); & : \quad \wedge \\ f' \leftarrow \mathcal{E}(1^\lambda, \mathsf{pp}, z); & \mathsf{com}^* \neq \mathsf{Commit}(f', \mathsf{pp}) \end{array}\right].$$

  where $\mathsf{CheckCom}$ *checks if a commitment is well-formed.*

There are several works in the literature on verifiable polynomial delegation [19,42,56,69]. Our construction extends the scheme of Papamanthou et al. [69] (which itself extends prior work [56] to the multivariate case) to achieve a "knowledge" property, i.e., to ensure that if the server can successfully prove that y is the correct output relative to com for some input t, then the server in fact knows a polynomial f of the correct degree for which $f(t) = y$. Thus, our construction can be viewed as a special-purpose SNARK for polynomial evaluation.

As our starting point we use the selectively secure VPD scheme of Papamanthou et al. [69]. Unfortunately, selective security means that the parameters used for the VPD protocol are computed as a function of the specific point r_d on which the VPD will be executed. This is insufficient for our application since VPD's parameters will be generated once during the preprocessing phase which happens *before* the CMT protocol.

To overcome this limitation, we modify this scheme to require the prover to provide additional "extractability" terms as part of the evaluation proof. Our modified VPD scheme is given in Construction 2. We define the *variable degree* of a multivariate polynomial f be the maximum degree of f in any of its variables, and use $\mathcal{W}_{\ell,d}$ to denote the collection of all multisets of $\{1, \ldots, \ell\}$ for which the multiplicity of any element is at most d.

Construction 2 (Verifiable Polynomial Delegation). *Let $\mathbb{F}$ be a prime-order field, and ℓ, d variable and degree parameters such that $O(\binom{\ell+\ell d-1)}{\ell d})$ is poly(λ). Consider the following protocol for the family $\mathcal{F}$ of ℓ-variate polynomials of variable-*

degree d over $\mathbb{F}$.

1. $\mathsf{KeyGen}(1^\lambda, \ell, d)$*: Select uniform* $\alpha, s_1, \ldots, s_\ell \in \mathbb{F}$*, run* $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^\lambda)$ *and compute* $\mathbb{P} = \{g^{\prod_{i \in W} s_i}, g^{\alpha \cdot \prod_{i \in W} s_i}\}_{W \in \mathcal{W}_{\ell,d}}$*. The public parameters are* $\mathsf{pp} = ((p, \mathbb{G}, \mathbb{G}_T, e, g), \mathbb{P}, g^\alpha)$*, and the verifier parameters are* $\mathsf{vp} = ((p, \mathbb{G}, \mathbb{G}_T, e, g), g^{s_1}, \cdots, g^{s_\ell}, g^\alpha)$*. For every* $f \in \mathcal{F}$ *we denote by* $\mathsf{pp}_f \subseteq \mathsf{pp}$ *the minimal subset of the public parameters* pp *required to invoke* Commit *and* Evaluate *on* f*.*

2. $\mathsf{Commit}(f, \mathsf{pp}_f)$*: If* $f \notin \mathcal{F}$ *output null. Else, compute* $c_1 = g^{f(s_i, \ldots, s_\ell)}$ *and* $c_2 = g^{\alpha \cdot f(s_i, \ldots, s_\ell)}$*, and output the commitment* $\mathsf{com} = (c_1, c_2)$*.*

3. $\mathsf{CheckCom}(\mathsf{com}, \mathsf{vp})$*: Check whether* com *is well-formed, i.e., output* 1 *if* $e(c_1, g^\alpha) = e(c_2, g)$ *and* 0 *otherwise.*

4. $\mathsf{Evaluate}(f, t, \mathsf{pp}_f)$*: On input* $t = (t_1, \ldots, t_\ell)$*, compute* $y = f(t)$*. Next, using Lemma 1 compute the polynomials* $q_i(x_i, \ldots, x_\ell)$ *for* $i = 1, \ldots, \ell$*, such that* $f(x_1, \ldots, x_\ell) - f(t_1, \ldots, t_\ell) = \sum_{i=1}^{\ell} (x_i - t_i) \cdot q_i(x_i, \ldots, x_\ell)$*. Output* y *and the proof* $\pi := \{g^{q_i(s_1, \ldots, s_\ell)}, g^{\alpha q_i(s_1, \ldots, s_\ell)}\}_{i=1}^{\ell}$*.*

5. $\mathsf{Ver}(\mathsf{com}, y, t, \pi, \mathsf{vp})$*: Parse the proof* π *as* $(\pi_1, \pi_1' \ldots, \pi_\ell, \pi_\ell')$*. If* $e(c_1/g^y, g) = \prod_{i=1}^{\ell} e(g^{s_i - t_i}, \pi_i)$ *and* $e(c_1, g^\alpha) = e(c_2, g)$ *and* $e(\pi_i, g^\alpha) = e(\pi_i', g)$ *for* $1 \leq i \leq \ell$ *output* 1*. Otherwise, output* 0*.*

We have the following theorem:

Theorem 3. *Under Assumptions 1 and 2, Construction 2 is an extractable VPD scheme. For a variable-degree-d ℓ-variate polynomial $f \in \mathcal{F}$ containing m monomials, algorithm* KeyGen *runs in time* $O(\binom{\ell(d+1)-1}{\ell d})$*,* Commit *in time* $O(m)$*,* Evaluate

in time $O(\ell dm)$, Ver *in time* $O(\ell)$ *and* $\mathsf{CheckCom}$ *in time* $O(1)$. *If* $d = 1$, $\mathsf{Evaluate}$ *runs in time* $O(2^\ell)$. *The commitment produced by* Commit *consists of* $O(1)$ *group elements, and the proof produced by* $\mathsf{Evaluate}$ *consists of* $O(\ell)$ *elements of* $\mathbb{G}$.

Proof. The completeness requirement immediately follows from the construction of $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Evaluate}, \mathsf{Ver})$.

We now prove the extractability property. Let Adv be a PPT adversary that on input $(1^\lambda, \mathsf{pp})$, where $(\mathsf{pp}, \mathsf{vp})$ is the output of $\mathsf{KeyGen}(1^\lambda, \ell, d)$, outputs commitment com^* such that $\mathsf{CheckCom}(\mathsf{com}^*, \mathsf{vp})$ accepts. This implies that $e(c_1, g^\alpha) = e(c_2, g)$ where $\mathsf{com}^* \stackrel{\text{def}}{=} (c_1, c_2)$. By Assumption 2, there exists PPT extractor $\mathcal{E}'$ for Adv such that upon the same input as Adv, and with access to the same random tape, outputs $a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|} \in \mathbb{F}$ such that $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_W \prod_{i \in W} s_i} = c_1$, except with negligible probability. Note that, the coefficients $(a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|})$ can be encoded as a variable-degree-d, ℓ-variate polynomial that has a_i as its monomial coefficients. We now build extractor $\mathcal{E}$:

1. Upon input $(1^\lambda, \mathsf{pp})$, $\mathcal{E}$ runs $\mathcal{E}'$ on the same input.

2. $\mathcal{E}$ tries to parse the output of $\mathcal{E}'$ as $a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|} \in \mathbb{F}$ and aborts if this fails.

3. $\mathcal{E}$ outputs f', where $f' \in \mathcal{F}$ is the polynomial with coefficients $a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|}$.

Note that $\mathcal{E}$ is PPT as $\mathcal{E}'$ is PPT and it only performs polynomially many operations in $\mathbb{F}$. It remains to argue that f' is a valid pre-image of Commit except with negligible probability. Observe that, if $\mathcal{E}$ does not abort, it follows from the construction of Commit that $\mathsf{Commit}(f', \mathsf{pp}) = \mathsf{com}$, where com is the output commitment of Adv.

By assumption 2, the probability that the output $\mathcal{E}'$ is not a valid set of coefficients is negligible which concludes the proof.

Next, we prove the soundness property. Let Adv be a PPT adversary that wins the soundness game with non-negligible probability. For $i = 1, \ldots, \ell$ we define adversary Adv_i that receives the same input as Adv and executes the same code, but outputs only $(\pi_i, \pi_i') \in \pi^*$ (where π^* is the proof output by Adv). Moreover, since Adv is PPT, all these adversaries are also PPT. Thus, for $i = 1, \ldots, \ell$, from Assumption 2 there exists PPT $\mathcal{E}_i$ (running on the same random tape as Adv_i) which on input $(1^\lambda, \mathsf{pp})$ outputs $a_{0,i}, \ldots, a_{|\mathcal{W}_{\ell,d}|,i} \in \mathbb{F}$ such that the following holds: If $e(\pi_i, g^\alpha) = e(\pi_i', g)$ then $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_{W,i} \prod_{j \in W} s_j} \neq \pi_i$, except with negligible probability. Note that, the coefficients $(a_{0,i}, \ldots, a_{|\mathcal{W}_{\ell,d}|,i})$ for $i = 1, \ldots, \ell$ can always be encoded as a variable-degree-d, ℓ-variate polynomial which we denote by $q_i'(\mathbf{x})$ for undefined variable $\mathbf{x} = (x_1, \ldots, x_\ell)$.

We construct an adversary $\mathcal{B}$ that breaks Assumption 1. On input $(1^\lambda, p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, g^{s^2}, \ldots, g^{s^{\ell \cdot d}})$, $\mathcal{B}$ does the following:

Parameter generation. $\mathcal{B}$ implicitly sets $s_1 = s$ and for $i = 1, \ldots, \ell$ he chooses $r_i \in \mathbb{F}$ uniformly at random and sets (also implicitly) $s_i = s \cdot r_i$. Then he chooses uniformly at random a value $\alpha \in \mathbb{F}$. Next $\mathcal{B}$ needs to generate the terms in $\mathbb{P} = \{g^{\prod_{i \in W} s_i}, g^{\alpha \cdot \prod_{i \in W} s_i}\}_{W \in \mathcal{W}_{\ell,d}}$. Since the exponent of each term is a product of at most $\ell \cdot d$ factors where each factor is one of the values $s_i = s \cdot r_i$, it can be written as a polynomial in s with degree at most $\ell \cdot d$. Therefore, $\mathcal{B}$ can compute these terms from the values $g, g^s, g^{s^2}, \ldots, g^{s^{\ell \cdot d}}$ and α. Finally, $\mathcal{B}$ runs Adv on input $(1^\lambda, \mathsf{pp})$,

where $\mathsf{pp} = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^\alpha, \mathbb{P})$.

Query evaluation. Upon receiving (f^*, t^*, y^*, π^*) from Adv, $\mathcal{B}$ first runs $\mathsf{Commit}(f^*, \mathsf{pp})$ to receive $\mathsf{com} \stackrel{\text{def}}{=} (c_1, c_2)$ and then runs $\mathsf{Ver}(\mathsf{com}, t^*, y^*, \pi^*, \mathsf{vp})$ where $\mathsf{vp} = (1^\lambda, p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, g^{s^2}, \ldots, g^{s^{\ell \cdot d}}, g^\alpha)$. If Ver rejects, $\mathcal{B}$ aborts, else he runs extractors $\mathcal{E}_1, \ldots, \mathcal{E}_\ell$ (defined above) on the same input as Adv and receives polynomials $q'_1, \ldots, q'_\ell$. If for the output of any of the $\mathcal{E}_i$ it holds that $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_{W,i} \prod_{j \in W} s_j} \neq \pi_i$, $\mathcal{B}$ aborts. Otherwise, let $\delta = y^* - f^*(t^*)$ and let $Q(\mathbf{x})$ be the polynomial over $\mathbb{F}$ defined as $Q(\mathbf{x}) \stackrel{\text{def}}{=} f^*(\mathbf{x}) - f^*(t^*) - \sum_{i=1}^{\ell}(x_i - t_i) q'_i(\mathbf{x})$ where $t^* \stackrel{\text{def}}{=} (t_1, \ldots, t_\ell)$. $\mathcal{B}$ picks $\tau \in \mathbb{F}$ uniformly at random. If $g^\tau = g^{-s}$, he sets $\tau \leftarrow \tau + 1$. He then computes polynomial $Q'(x) \stackrel{\text{def}}{=} Q(\mathbf{x})/(\tau + x_1)$ and finally outputs $(\tau, e(g,g)^{\delta^{-1} \cdot Q'(s_1, \ldots, s_\ell)})$ as a challenge tuple for Assumption 1.

Since $s_1 = s, s_2 = r_2 \cdot s, \ldots, s_\ell = r_\ell \cdot s$, we have $Q'(s_1, \ldots, s_\ell) = Q''(s)$ where Q'' is an efficiently computable univariate polynomial of degree $\ell \cdot d$ hence $e(g,g)^{-\delta \cdot Q'(s_1, \ldots, s_\ell)}$ is computable from $(1^\lambda, p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, g^{s^2}, \ldots, g^{s^{\ell \cdot d}})$. $\mathcal{B}$ is clearly PPT since all of $\mathcal{E}_i$ are PPT and he performs polynomially many operations in $\mathbb{F}, \mathbb{G}, \mathbb{G}_T$. Next, we analyze the success probability of $\mathcal{B}$. Recall that, by assumption Adv succeeds in violating soundness with probability ϵ. We observe that, *conditioned on not aborting*, $\mathcal{B}$'s output is always a valid tuple for breaking Assumption 1. Let us argue why this is true. Since verification succeeds, it holds that

$e(c_1/g^{y^*}, g) = \prod_{i=1}^{\ell} e(g^{s_i - t_i}, \pi_i)$; since extraction succeeds, this can be replaced with

$$e(g,g)^{f^*(s_1,\ldots,s_\ell)-\delta-f^*(t^*)} = \prod_{i=1}^{\ell} e(g^{s_i-t_i}, g^{q_i'(s_1,\ldots,s_\ell)})$$

$$e(g,g)^{\delta} = e(g,g)^{f^*(s_1,\ldots,s_\ell)-f^*(t^*)} \prod_{i=1}^{\ell} e(g^{s_i-t_i}, g^{-q_i'(s_1,\ldots,s_\ell)})$$

$$e(g,g)^{\delta} = e(g,g)^{f^*(s_1,\ldots,s_\ell)-f^*(t^*)-\sum_{i=1}^{\ell}(s_i-t_i)q_i'(s_1,\ldots,s_\ell)}.$$

By the definition of Q' it follows that

$$e(g,g)^{\delta} = e(g,g)^{Q(s_1,\ldots,s_\ell)}$$

$$e(g,g)^{\frac{\delta}{\tau+s_1}} = e(g,g)^{\frac{Q(s_1,\ldots,s_\ell)}{\tau+s_1}} = e(g,g)^{Q'(s_1,\ldots,s_\ell)}$$

$$e(g,g)^{\frac{1}{\tau+s_1}} = e(g,g)^{\delta^{-1}\cdot Q'(s_1,\ldots,s_\ell)}.$$

Thus, the final piece in order to conclude the proof is to bound the probability that $\mathcal{B}$ aborts. Note that, conditioned on Adv winning, $\mathcal{B}$ will only abort if extraction fails which can only happen with negligible probability $\mathsf{neg}(\lambda)$. This holds since, if verification succeeds it must be that $e(\pi_i', g) = e(\pi_i, g^{\alpha})$ for $i = 1, \ldots, \ell$ and in this case, by Assumption 2, extraction for any of $\mathcal{E}_1, \ldots, \mathcal{E}_\ell$ fails with negligible probability. Since ℓ is polynomial in λ it follows that the probability any of them fails (which by a union bound is at most equal to the sum of each individual failure probability) is also negligible.

Finally, let us argue that the polynomial division $Q(\mathbf{x})/(\tau + x_1)$ is always possible. Recall, that for polynomials defined over finite fields division is always possible assuming that the dividend's degree is at least as large as that of the divisor's. Moreover, the degree of the quotient is at most that of the dividend's and that of the remainder is strictly smaller than that of the divisor. Let us assume for

contradiction that $Q(\mathbf{x})$ is a constant polynomial. Since, $e(g,g)^{\delta} = e(g,g)^{Q(s_1,\ldots,s_{\ell+1})}$ and $e(g,g)$ is a generator or $\mathbb{G}_T$, it must be that $Q(\mathbf{x}) \stackrel{\text{def}}{=} \delta$ therefore we can write

$$\begin{aligned} -\delta &= \sum_{i=1}^{\ell}(x_i - t_i)q_i'(\mathbf{x}) - f^*(\mathbf{x}) + f^*(t^*) \\ f^*(\mathbf{x}) - \delta - f^*(t^*) &= \sum_{i=1}^{\ell}(x_i - t_i)q_i'(\mathbf{x}) \\ f^*(\mathbf{x}) - y^* &= \sum_{i=1}^{\ell}(x_i - t_i)q_i'(\mathbf{x}) \end{aligned}$$

From the above relation it follows that t^* is a root of the polynomial $f' \stackrel{\text{def}}{=} f^*(\mathbf{x}) - y^*$, i.e., $f'(t^*) = 0$ which implies that $f^*(t_1,\ldots,t_\ell) = y^*$. Thus, in this case, y^* is the correct evaluation of f^* on t^*, i.e., $\delta = 0$ and Adv did not cheat. In all other cases, the polynomial division is possible.

From the above analysis it follows that the probability that $\mathcal{B}$ succeeds is at least $(1 - \mathsf{neg}(\lambda))\epsilon$. By assumption, ϵ is the non-negligible probability that Adv wins the soundness game, therefore $\mathcal{B}$'s success probability is also non-negligible. This contradicts Assumption 1 and our proof is complete.

Asymptotic analysis. The claims for the general polynomial case follow directly from the analysis of [69]. For $d = 1$, i.e., for multi-linear polynomials, we prove the tighter bound for the runtime of $\mathsf{Evaluate}$ below.

Recall that during $\mathsf{Evaluate}$ the prover computes polynomials $q_i(x_i,\ldots,x_\ell)$ for $i = 1,\ldots,\ell$, such that $f(x_1,\ldots,x_\ell) = \sum_{i=1}^{\ell}(x_i - t_i)\cdot q_i(x_i,\ldots,x_\ell) + f(t_1,\ldots,t_\ell)$ and proof $\pi = \{g^{q_i(s_i,\ldots,s_\ell)}, g^{\alpha q_i(s_i,\ldots,s_\ell)}\}_{i=1}^{\ell}$. We start by computing $q_1(x_1,\ldots,x_\ell)$. Since the degree of every variable is at most 1, the multi-linear polynomial f can be written as $f(x_1,\ldots,x_\ell) = g(x_2,\ldots,x_\ell) + x_1 \cdot h(x_2,\ldots,x_\ell)$, where $g(x_2,\ldots,x_\ell)$ and

$h(x_2, \ldots, x_\ell)$ are multi-linear polynomials of variables $x_2, \ldots, x_\ell$. In this way, f can be decomposed as

$$f(x_1, \ldots, x_\ell) = g(x_2, \ldots, x_\ell) + x_1 \cdot h(x_2, \ldots, x_\ell)$$

$$= (g(x_2, \ldots, x_\ell) + t_1 \cdot h(x_2, \ldots, x_\ell)) + (x_1 - t_1)h(x_2, \ldots, x_\ell)$$

$$= R_1(x_2, \ldots, x_\ell) + (x_1 - t_1)h(x_2, \ldots, x_\ell)\,.$$

We set $q_1(x_1, \ldots, x_\ell) = h(x_2, \ldots, x_\ell)$ (which means q_1 contains no monomial with x_1), and proceed to decompose the multi-linear polynomial $R_1(x_2, \ldots, x_\ell)$ with $\ell - 1$ variables in the same way as f to compute $q_2(x_2, \ldots, x_\ell)$. Regarding the complexity of this, note that both $g(x_2, \ldots, x_\ell)$ and $h(x_2, \ldots, x_\ell)$ contain at most $2^{\ell-1}$ monomials. Therefore, it takes $2^{\ell-1}$ additions and multiplications to compute $q_1(x_1, \ldots, x_\ell)$ and $R_1(x_2, \ldots, x_\ell)$, and $2^{\ell-1}$ exponentiations to generate $g^{q_1(s_1,\ldots,s_\ell)}$ and $g^{\alpha q_1(s_1,\ldots,s_\ell)}$ in the proof, respectively. The exact same reasoning applies for all of $q_3, \ldots, q_\ell$. At the last step after computing $q_\ell(x_\ell)$, the remaining constant term is equal to the answer $f(t_1, \ldots, t_\ell)$. In general, in the ith step, we are decomposing $R_{i-1}(x_i, \ldots, x_\ell)$ with $\ell - i + 1$ variables in the same way above to compute $q_i(x_i, \ldots, x_\ell)$ and $R_i(x_{i+1}, \ldots, x_\ell)$, and the complexity is $O(2^{\ell-i})$. Thus, the total complexity of computing $q_1, \ldots, q_\ell$ is $O(2^{\ell-1}) + O(2^{\ell-2}) + \ldots = O(2^\ell)$. The polynomial evaluation in order to get the answer takes the same time. Each pair π_i, π_i' is computed with two exponentiations, thus the overall running time is $O(2^\ell)$. $\square$

3.2 Improving the Expressiveness of the CMT Protocol

As presented in Theorem 2 and Remark 2, the prover complexity of the CMT protocol is in particular efficient if the circuit can be represented as many parallel copies of the same small sub-circuit. In this section, we show how to modify the CMT protocol to achieve the same prover efficiency for circuits that consist of multiple (different) sub-circuits.

Let C be a depth-d, size-n, layered arithmetic circuit consisting of B independent ("parallel") sub-circuits $C_1, \cdots, C_B$, each of depth at most d' and size at most n', where the outputs of $C_1, \cdots, C_n$ are fed into an aggregation circuit D of depth-d'' and size n''. In this section, we show how to modify the CMT protocol so as to prove statements about the output of C in time which is linear in the size of C. Our modified protocol proceeds as follows. We start by following the standard CMT protocol for the d'' layers of sub-circuit D. Next, for the remaining $d - d'' = d'$ layers, we modify things in a similar way to [78] and [79]. Let S_i now denote the maximum number of gates in layer i across $C_1, \cdots, C_B$, and let $s_i = \lceil \log S_i \rceil$. We let V_i again be a function mapping a gate at level i to its value, but we now specify a gate g by a pair g_1, g_2, where $g_2 \in [B]$ indicates the sub-circuit in which g lies and $g_1 \in [S_i]$ is the index of g (at level i) within that sub-circuit. The prover and verifier then run a CMT-like protocol, but using the equation $V_i(g_1, g_2) = \sum_{u_1, v_1 \in \{0,1\}^{s_{i+1}}} (\mathsf{add}_{i+1}(g_1, u_1, v_1, g_2) \cdot (V_{i+1}(u_1, g_2) + V_{i+1}(v_1, g_2)) + \mathsf{mult}_{i+1}(g_1, u_1, v_1, g_2) \cdot (V_{i+1}(u_1, g_2) \cdot V_{i+1}(v_1, g_2)))$.

The equation above still recursively defines V_i in terms of V_{i+1}, but takes

advantage of the fact that there is no interconnection between the different sub-circuits. This has the effect of reducing the number of variables in add_{i+1} and mult_{i+1} from $2s_{i+1} + s_i + 3\lceil \log B \rceil$ to $2s_{i+1} + s_i + \lceil \log B \rceil$. Next, we define the multilinear extension of $V_i(g_1, g_2)$.

$$\begin{aligned}\tilde{V}_i(z_1, z_2) &= \sum_{u_1, v_1 \in \{0,1\}^{s_{i+1}}, g_2 \in \{0,1\}^{\log \lceil B \rceil}} f_{i,z_1,z_2}(u_1, v_1, g_2) \\ &\stackrel{\text{def}}{=} \sum_{u_1, v_1 \in \{0,1\}^{s_{i+1}}, g_2 \in \{0,1\}^{\lceil \log B \rceil}} \tilde{\beta}_i(z_2, g_2) \cdot \Big(\tilde{\mathsf{add}}_{i+1}(z_1, u_1, v_1, g_2) \cdot (\tilde{V}_{i+1}(u_1, g_2) \\ &+ \tilde{V}_{i+1}(v_1, g_2)) + \tilde{\mathsf{mult}}_{i+1}(z_1, u_1, v_1, g_2) \cdot (\tilde{V}_{i+1}(u_1, g_2) \cdot \tilde{V}_{i+1}(v_1, g_2)) \Big).\end{aligned} \tag{3.1}$$

The only difference between equation 3.1 and the equation used for data-parallel circuits with same sub-circuits in [78, 79] is that $\tilde{\mathsf{add}}_{i+1}$ and $\tilde{\mathsf{mult}}_{i+1}$ take an extra variable g_2, which denotes that the gates and wiring patterns can be different in each sub-circuit. We further observe that running the same algorithm for the sumcheck protocol as in [78, 79] on equation 3.1 results in the same complexity on the prover, which is $O(BS_i \log S_{i+1})$. To see this, for the first $2s_{i+1}$ rounds, there are at most BS_i monomials per round, as there are at most BS_i gates in the i-th layer of the circuit and the number of non-zero monomials in $\tilde{\mathsf{add}}_{i+1}$ and $\tilde{\mathsf{mult}}_{i+1}$ is bounded by the number of gates. By Remark 1, this takes $O(BS_i)$ arithmetic operations per round, so the complexity for these rounds is $O(BS_i \log S_{i+1})$. For the remaining rounds, by Remark 1, $\mathcal{P}$'s running time is $O(2^{\lceil \log B \rceil - j})$ in round $2s_{i+1} + j$ $(j = 1, \ldots, \lceil \log B \rceil)$ and the complexity is $O(B)$. Thus, the complexity is dominated by the first part, i.e., $O(BS_i \log S_{i+1})$.

In this way, we extend the class of the circuit efficiently supported by the CMT protocol in [78, 79] without any overhead on the prover time.[1] We present the following result.

Theorem 4. *Let $C : \mathbb{F}^n \to \mathbb{F}$ be a depth-d layered arithmetic circuit consisting of B parallel sub-circuits $C_1, \ldots, C_B$ connected to an "aggregation" circuit D such that $|D| = O(|C|/\log|C|)$, and let $S = \max_j\{\mathsf{width}(C_j)\}$. Executing the CMT protocol from Construction 1 using Equation 3.1 and the above described modifications to the sum-check protocol, yields an interactive proof for C with soundness $O(d \cdot \mathsf{width}(C)/|\mathbb{F}|)$. Moreover, $\mathcal{P}$'s running time is $O(|C| \log S)$ and the protocol uses $O(d \log(\mathsf{width}(C)))$ rounds of interaction. If $\tilde{\mathsf{add}}_i$ and $\tilde{\mathsf{mult}}_i$ are computable in time $O(\mathrm{polylog}(\mathsf{width}(C)))$ for all the layers of C, then the running time of the verifier $\mathcal{V}$ is $O(n + d \cdot \mathrm{polylog}(\mathsf{width}(C)))$.*

3.3 The Construction of Our Argument System

Finally, we present our new argument system with circuit-independent preprocessing. Our construction combines the modified CMT protocol from Section 3.2 with the VPD scheme presented in Section 3.1. We refer to the prover and verifier of the CMT protocol as $(\mathcal{P}_{cmt}, \mathcal{V}_{cmt})$, respectively, and to the algorithms of the VPD scheme as $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Evaluate}, \mathsf{Ver})$. We construct an argument system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for the satisfiability of arithmetic circuits over finite fields, where the preprocessing done by $\mathcal{G}$ depends on a bound on the size of the circuit, the size of its

[1] The complexity of the CMT protocol for circuits composed of identical sub-circuits has recently been improved to $O(BS_i + S_i \log S_i)$ in [82].

input, and the field over which it is defined, but not the circuit itself.

Let $\mathcal{V}_{cmt}^{1+2}$ be the restriction of the CMT verifier from Construction 1 which performs Steps 1 and 2 of $\mathcal{V}_{cmt}$ and outputs (r_d, a_d) without performing Step 3. Construction 3 is a formal description of our argument system.

Construction 3. *Let $\mathbb{F}$ be a prime-order field with $|\mathbb{F}|$ exponential in λ, and let n, t be input size and circuit size parameters. For simplicity of exposition we assume that n is a power of 2. Consider the algorithms $\mathcal{G}, \mathcal{P}, \mathcal{V}$ described below.*

Preprocessing phase. *$\mathcal{G}(1^\lambda, n, t)$ runs $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, 1)$. The proving key pk is set to be pp and the verification key vk is set to be vp.*

Evaluation phase. *Let $C : \mathbb{F}^{n_x+n_w} \to \mathbb{F}$ be a depth-d layered arithmetic circuit over $\mathbb{F}$ with at most t gates such that $n_x + n_w \leq n$. Moreover, let $x \in \mathbb{F}^{n_x}$ and $w \in \mathbb{F}^{n_w}$ be such that $C(x; w) = 1$. Assume that $n_w/n_x = 2^m - 1$ for some $m \in \mathbb{N}$. Consider the following protocol between $\mathcal{P}$ and $\mathcal{V}$.*

1. *$\mathcal{P}$ first commits to the multilinear extension $\widetilde{V}_d$ of the input layer of $C(x; w)$. That is, $\mathcal{P}$ runs $c \leftarrow \mathsf{Commit}(\widetilde{V}_d, \mathsf{pp})$ and sends c to $\mathcal{V}$. Upon receving c, $\mathcal{V}$ runs $\mathsf{CheckCom}(c, \mathsf{vp})$. If the output is reject, $\mathcal{V}$ rejects.*

2. *$\mathcal{V}$ computes the multilinear extension $\tilde{x}$ of the input x, generates a random point $r \in (\mathbb{F}^{\log(n_x)} \times 0^{\log(n_w)})$ and sends r to $\mathcal{P}$. Upon receiving r, $\mathcal{P}$ executes $(a, \pi) \leftarrow \mathsf{Evaluate}(\tilde{V}_d, r, \mathsf{pp})$ and sends (a, π) to $\mathcal{V}$. Upon receiving (a, π), $\mathcal{V}$ executes $\mathsf{Ver}(c, a, r, \pi, \mathsf{vp})$. In case Ver outputs 0 or $a \neq \tilde{x}(r)$, $\mathcal{V}$ outputs 0.*

3. *$\mathcal{V}$ runs $\mathcal{V}_{cmt}^{1+2}$ and $\mathcal{P}$ runs $\mathcal{P}_{cmt}$ to verify $C(x; w) = 1$. If $\mathcal{V}_{cmt}^{1+2}$ rejects at any point,*

$\mathcal{V}$ *outputs* 0. *Otherwise, let* r_d, a_d *be the final values returned by* $\mathcal{V}_{cmt}^{1+2}$. *At this point,* $\mathcal{V}$ *must verify that* $\tilde{V}_d(r_d) = a_d$.

4. $\mathcal{V}$ *sends* r_d *to* $\mathcal{P}$. *Upon receiving* r_d, $\mathcal{P}$ *executes* $\mathsf{Evaluate}(\tilde{V}_d, r_d, \mathsf{pp})$ *and obtains* (a'_d, π') *which he sends to* $\mathcal{V}$.

5. $\mathcal{V}$ *upon receiving* (a'_d, π') *executes* $\mathsf{Ver}(c, a'_d, r_d, \pi', \mathsf{vp})$. *In case* Ver *outputs* 0 *or* $a'_d \neq a_d$, $\mathcal{V}$ *outputs* 0. *Otherwise,* $\mathcal{V}$ *outputs* 1.

We have the following theorem:

Theorem 5. *If Construction 2 is an extractable VPD scheme, then Construction 3 is an argument system for arithmetic circuits, as defined by Definition 2. The running time of* $\mathcal{P}$ *is* $O(|C| \log |C|)$. *When used for a depth-*d*, layered circuit* C *consisting of* B *parallel sub-circuits* $C_1, \ldots, C_B$ *(the sub-circuits can be the same or different) whose outputs feed into a circuit* D *with* $|D| \leq |C|/\log |C|$, *the running time of* $\mathcal{P}$ *is* $O(|C| \cdot \log \max_j \{\mathsf{width}(C_j)\})$ *and the protocol has* $O(d \log(\mathsf{width}(C)))$ *rounds. If* C *has input length* n *and is log-space uniform then the running time of* $\mathcal{V}$ *is* $O(n + d \cdot \mathrm{poylog}(|C|))$. *Finally, if* d *is* $\mathrm{polylog}\,(|C|)$, *the above construction is a* succinct *argument.*

Proof. The completeness requirement immediately follows from the construction of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$.

We now argue about the knowledge soundness property. Let Adv be an adversary which outputs a circuit C with $|C| \leq t$ and $n_x + n_w \leq n$, where n_x (n_w) is the size of the input (auxiliary input) of C) and an input $x \in \mathbb{F}^{n_x}$ and is able to

make $\mathcal{V}$ accept for (C, x). We will construct a corresponding extractor $\mathcal{E}$ which is able to produce a witness $w \in \mathbb{F}^{n_w}$ such that $C(x; w) = 1$, except with negligible probability.

We begin by observing that the prover parameters pk of the argument, that are given as input to Adv, are equivalent to pp output by $\mathsf{KeyGen}(1^\lambda, n, 1)$. Therefore, since Adv convinces $\mathcal{V}$ which runs $\mathsf{CheckCom}$ as a sub-routine, it holds that in Step 1 Adv outputs a $c \stackrel{\text{def}}{=} (c_1, c_2)$ for the input layer of the circuit C such that $\mathsf{CheckCom}(c, \mathsf{vp})$ accepts.

Next, let Adv' be a simplified version of Adv that runs the same code of but halts right after Step 1, outputting only commitment c. Clearly, whenever $\mathsf{CheckCom}(c, \mathsf{vp})$ accepts when interacting with Adv, it will also accept upon receiving the output of Adv' since they produce the same output. Thus, from the extractability property of our VPD scheme, it follows that there exists extractor $\mathcal{E}'$ that upon the same input as Adv' outputs f' such that $c = \mathsf{Commit}(f', \mathsf{pp})$ except with negligible probability. Note that Adv' is PPT as Adv' is PPT, therefore, by extractability, $\mathcal{E}'$ is also PPT.

Now we are ready to define our main extractor $\mathcal{E}$ for Adv. Upon input $(1^\lambda, \mathsf{pk})$, $\mathcal{E}$ operates as follows:

1. Run $\mathcal{E}'(1^\lambda, \mathsf{pk})$ and receive polynomial f'. If f' is not a n-variate polynomial of variable-degree 1, abort.

2. Output $w = (f'(n_x), \ldots, f'(n_w - 1))$.

$\mathcal{E}$ is PPT as $\mathcal{E}'$ is PPT, and he performs only polynomially many operations in $\mathbb{F}$.

It remains to show that in case Adv convinces $\mathcal{V}$, then it holds that $C(x;w) = 1$ where w is the output of $\mathcal{E}$, except with negligible probability. Assume for contradiction that Adv convinces $\mathcal{V}$ and $C(x;w) \neq 1$ with some non-negligible probability ϵ, where w is the output of $\mathcal{E}$. We will build an adversary $\mathcal{B}$ that uses $\mathsf{Adv}, \mathcal{E}$ in order to break the soundness of the CMT protocol or of our VPD construction, as follows:

1. $\mathcal{B}$ receives as input $(\mathsf{pp}, \mathsf{vp})$ generated from $\mathsf{KeyGen}(1^\lambda, n, 1)$ from a challenger for the VPD soundness game. He then runs $\mathsf{Adv}(1^\lambda, \mathsf{pk} = \mathsf{pp})$.

2. Let C, x be the circuit and input chosen by Adv and let d be the depth of C. Moreover, let c be the commitment output by Adv (claimed to be a commitment to the input layer of C).

3. $\mathcal{B}$ runs $\mathsf{CheckCom}(c, \mathsf{vp})$ and if it outputs reject he aborts. Else, he runs $\mathcal{E}(1^\lambda, \mathsf{pk})$ and receives witness w. If $w \notin \mathbb{F}^{n_w}$, $\mathcal{B}$ aborts. Else, he sets polynomial $f : \mathbb{F}^n \to \mathbb{F}$ to be the multilinear extension of the array $x||w$ (i.e. the multilinear extension of the entire input to C).

4. $\mathcal{B}$ chooses $r \in \mathbb{F}^{\log n_x} \times 0^{\log n_w}$ uniformly at random and forwards it to Adv. Upon receiving a, π, he runs $\mathsf{Ver}(c, a, r, \pi, \mathsf{vp})$. If it outputs reject he aborts. Else, if $a \neq f(r)$, $\mathcal{B}$ outputs (f, r, a, π) as a challenge for VPD and terminates.

5. $\mathcal{B}$ initializes the interaction with $\mathcal{V}_{cmt}$ for circuit C and input x, w. For all layers of C (from 0 to d) $\mathcal{B}$ simply forwards the messages of $\mathcal{V}_{cmt}$ to Adv and vice versa.

6. Let r_d be the random point established by $\mathcal{V}_{cmt}$ for the d-th layer of C. $\mathcal{B}$ forwards

r_d to Adv and receives a'_d, π'. He then runs $\mathsf{Ver}(c, a'_d, r_d, \pi', \mathsf{vp})$. If it outputs reject he aborts. Else, if $a'_d \neq f(r_d)$, $\mathcal{B}$ outputs (f, r_d, a'_d, π') as a challenge for VPD and terminates.

7. $\mathcal{B}$ forwards a'_d to $\mathcal{V}_{cmt}$ and terminates.

$\mathcal{B}$ is PPT since $\mathsf{Adv}, \mathcal{E}$ are PPT and he performs polynomially many operations in $\mathbb{F}, \mathbb{G}, \mathbb{G}_T$. Let us now argue about $\mathcal{B}$'s success probability.

We define the following events:

- B_{wins} is the event that $\mathcal{B}$ succeeds in breaking the VPD or the CMT soundness.
- B_{aborts} is the event that $\mathcal{B}$ aborts during his interaction with Adv.
- A is the event that Adv succeeds in convincing $\mathcal{P}$ to accept and $C(w, x) \neq 1$, where w is the output of $\mathcal{E}$.
- R is the event that during the interaction of $\mathcal{B}$ with Adv, $a \neq f(r)$ or $a'_d \neq f(r_d)$.
- $E = A \cap \mathcal{B}^c_{aborts}$.

For an event Y, let Y^c denote its complement. In general, by the law of total probability we can write

$$\Pr[B_{wins}] \geq \Pr[B_{wins}|E \cap R] \Pr[E \cap R] + \Pr[B_{wins}|E \cap R^c] \Pr[E \cap R^c].$$

Next, we turn our attention to evaluating the two summands.

For $\Pr[B_{wins}|E \cap R]$ we argue as follows. Conditioning on $E \cap R$ implies that: (i) Adv successfully convinces $\mathcal{V}$, (ii) $C(w, x) \neq 1$, (iii) $\mathcal{B}$ does not abort, and (iv)

$a \neq f(r)$ or $a'_d \neq f(r_d)$. Then, the view provided to Adv by $\mathcal{B}$ is a perfect emulation of the interaction with $\mathcal{V}$ and the view provided by $\mathcal{B}$ to $\mathcal{V}_{cmt}$ is a perfect emulation of the execution of CMT for $C(x, w)$, up to the point where $\mathcal{B}$ finds out that $a \neq f(r)$ (during Step (4) above), or that $a'_d \neq f(r_d)$ (during Step (7) above). Without loss of generality, we focus on the case where this occurs for $a \neq f(r)$ and the exact same reasoning follows for the other case. Simply observe that (f, r, a, π) is indeed a valid challenge tuple for the VPD soundness game, since $\mathsf{Ver}(c, a, r, \pi, \mathsf{vp})$ accepts and $\mathsf{Commit}(f, \mathsf{pp}) = c$, as $\mathcal{B}$ did not abort. It follows that, under these conditions, $\mathcal{B}$ always succeeds in winning therefore $\Pr[B_{wins}|E \cap R] = 1$.

Regarding $\Pr[B_{wins}|E \cap R^c]$ we argue as follows. Conditioning on $E \cap R^c$ implies that: (i) Adv successfully convinces $\mathcal{V}$, (ii) $C(w, x) \neq 1$, (iii) $\mathcal{B}$ does not abort, and (iv) $a = f(r)$ and $a'_d = f(r_d)$. Then, the view provided to Adv by $\mathcal{B}$ is a perfect emulation of the interaction with $\mathcal{V}$. Moreover, the view provided to $\mathcal{V}_{cmt}$ is a perfect emulation of the CMT protocol. Therefore, in this case too, $\mathcal{B}$ always succeeds as he falsely convinces $\mathcal{V}_{cmt}$ to accept for $C(x, w)$. Hence, we have that $\Pr[B_{wins}|E \cap R^c] = 1$ and replacing both probabilities in the above inequality we get

$$\begin{aligned}\Pr[B_{wins}] &\geq 1 \cdot \Pr[E \cap R] + 1 \cdot \Pr[E \cap R^c] \\ &= \Pr[E] = \Pr[A \cap B^c_{aborts}]\end{aligned}$$

where we used the fact that R, R^c are complementary events. Finally, by the definition of conditional probability, we get

$$\Pr[B_{wins}] \geq \Pr[A \cap B^c_{aborts}] = \Pr[B^c_{aborts}|A] \cdot \Pr[A].$$

Conditioned on Adv winning, we have that all of $\mathsf{CheckCom}(c, \mathsf{pp})$, $\mathsf{Ver}(c, a, r, \pi, \mathsf{vp})$, $\mathsf{Ver}(c, a'_d, r_d, \pi', \mathsf{vp})$ output accept since they are sub-routines of $\mathcal{V}$. It follows, that $\mathcal{B}$ may only abort if the extraction fails during Step (3) above. By the extractability property of our VPD and the construction of $\mathcal{E}$ this can happen only with negligible probability. Thus, we can write

$$\Pr[B_{wins}] \geq \Pr[B^c_{aborts}|A] \cdot \Pr[A] \geq (1 - \mathsf{neg}(\lambda)) \cdot \epsilon$$

which is non-negligible since, by our assumption, ϵ is non-negligible in λ.

Under our original assumption about the soundness of our VPD and that of CMT this should only happen with negligible probability. Thus, this contradicts our claim that Adv convinces $\mathcal{V}$ and $C(x, w) \neq 1$ with non-negligible probability (where w is the output of extractor $\mathcal{E}$), which concludes the proof of knowledge soundness of our argument system.

Asymptotic analysis. The input size of the circuit is $|x| + |w|$, which is bounded by the maximum width of the circuit $\mathsf{width}(C)$. Applying Construction 2 with $d = 1$ and $\ell = \log(\mathsf{width}(C))$, by Theorem 3, in Step 1, 2, 4 and 5 the running time of $\mathcal{P}$ is $O(\mathsf{width}(C))$ and the running time of $\mathcal{V}$ is $O(\log(\mathsf{width}(C)) + |x|)$. Combined with Step 3, by Theorem 2 for Construction 3, the running time of $\mathcal{P}$ is $O(|C| \cdot \log(\mathsf{width}(C)))$, the running time of $\mathcal{V}$ is $O(|x| + d \cdot \mathrm{polylog}(|C|))$, and $\mathcal{P}$ and $\mathcal{V}$ interact for $O(d \log(\mathsf{width}(C)))$ rounds.

In the analysis above, applying Remark 2 and Theorem 4 for Construction 1 to Step 3, the running time of $\mathcal{P}$ becomes $O(|C| \cdot \log S)$ if C consists of parallel sub-circuit.

□

Comparison to SNARKs. As presented above, the preprocessing of our construction only depends on an upper bound of the size of the input, but not on any particular circuit. In this way our construction does not have function-dependent preprocessing, as required by SNARKs.

In the evaluation phase, the CMT protocol only requires modular additions and multiplications. The only place having modular exponentiations in our scheme is in the VPD protocol, and the number of modular exponentiations is linear to the size of the input, while it is linear to the size of the circuit in SNARKs. As the modular exponentiation is the bottleneck of the schemes, this leads to significant improvements on the prover time. In addition, the fast multi-scalar exponentiation and fast Fourier transform (FFT) in SNARKs consumes high memory, while the CMT protocol and VPD protocol in our construction are memory friendly.

Our construction does introduce overhead on the proof size and the verification time. The proof size increases from $O(1)$ to $O(d \log(\mathsf{width}(C)))$ and the verification time increases from $O(|x|)$ to $O(|x|+d \cdot \mathrm{polylog}(|C|))$. However, they are still succinct (polylogarithmic on the witness size) and reasonable concretely in practice, as we will show in Section 4.6 and 5.3.

Chapter 4: Applications: Verifiable Databases

All major cloud providers offer Database-as-a-Service solutions that allow companies and individuals (*clients*) to alleviate storage costs and achieve resource elasticity by delegating storage and maintenance of their data to a cloud server. A client can then query and/or update its data using, e.g., standard SQL queries.

Outsourcing data in this way, however, introduces new security challenges: in particular, the client may need to ensure the integrity of the results returned by the server. Providing such a guarantee is important if the client does not trust the server, or even if the client is concerned about the possibility of server errors or external compromise.

Prior works on *verifiable databases* address exactly this problem, but have significant drawbacks. Function-specific schemes (e.g., authenticated data structures) target specific classes of computations and can be much more efficient than generic solutions; however, they suffer from limited expressiveness, and in particular they cannot handle a wide range of SQL queries. Generic solutions (e.g., SNARKs) can be used to verify arbitrary computations, but impose an unacceptable overhead at the server, and requires an expensive setup phase for every possible SQL query.

Our new argument system in Chapter 3 provides a solution for verifiable

databases. It is an argument system for any computations in NP, which can be used to validate arbitrary SQL queries in principle. Meanwhile, it does not require a separate setup phase for every SQL query. In this chapter we use our new argument system to construct vSQL, a system for verifiable databases and SQL queries. vSQL allows a client who owns a relational database to outsource it to an untrusted server while storing only a small *digest* locally. Later, the client can issue arbitrary SQL queries to the server, who returns the query's result. (In the case of an update query, the result is an updated digest.) The client can then verify the validity of the result using our argument system with the server; if the result returned by the server is incorrect, the client will reject with overwhelming probability. In addition, vSQL benefits from several performance optimizations that improve both the server's and client's concrete efficiency (see Section 4.5).

vSQL overcomes the drawbacks of existing works. It is highly expressive, supporting any computation expressed as an arithmetic circuit (which in particular means arbitrary SQL queries, including updates) efficiently. We empirically demonstrate vSQL's concrete performance and expressiveness using the TPC-H [9] benchmark, and find that the server-side computation (which is usually the limiting factor in verifiable-computation schemes) is 5–120× better for vSQL than it is in highly optimized SNARK-based constructions [6] (that further require query-dependent preprocessing), and comparable to or better than a state-of-the-art database-delegation scheme [91] that only supports a limited subset of SQL.

In this chapter, we first give some additional background on SQL queries in Section 4.1, and introduce additional related work on verifiable databases in

Section 4.2. We present the definition and our construction of verifiable databases in Section 4.3 and 4.4. We then describe additional optimizations of our argument system tailored for SQL queries in Section 4.5, followed by experimental results in Section 4.6.

4.1 SQL Queries

Structured Query Language (SQL) is a very popular programming language designed for querying and managing relational database systems. It operates on databases that consist of collections of two-dimensional matrices called *tables*. In the following, we briefly present the general structure of such queries and provide concrete examples for common types.

In SQL, a simple query begins with the keyword `SELECT` followed by a function $A(col_1, \ldots)$ and then the keyword `FROM` followed by a number of tables, where A is

$\mathbf{T_1}$:

row_id	employee_id	name	age	salary
1	2019	John	28	45,000
2	1905	Kate	31	55,000
3	1908	Lisa	44	70,000
4	2117	Leo	23	39,000
5	2003	Alice	29	34,000

$\mathbf{T_2}$:

row_id	employee_id	department
1	1905	Sales
2	1906	Sales
3	1908	HR
4	2003	R&D
5	2022	HR
6	2117	R&D

defined over (a subset of) the columns of the specified tables. This sequence of clauses and expressions dictates the output of the query. Following these, there is a `WHERE` clause followed by a sequence of predicates connected by logical operators (e.g, `AND`, `OR`, `NOT`) that restrict the rows used when computing the output. The above is best illustrated by a series of examples. Consider a database consisting of tables $\mathbf{T_1}$ and $\mathbf{T_2}$:

The first example we provide is a SQL range query which is used to select rows for which particular values fall within a set of specified ranges. The conditions may be defined over multiple columns, in which case we refer to it as a *multi-dimensional range query*. For example, the query "`SELECT` * `FROM` $\mathbf{T_1}$ `WHERE` age < 35 `AND` salary $> 40,000$" is a two-dimensional range query that returns the following table.

row_id	employee_id	name	age	salary
1	2019	John	28	45,000
2	1905	Kate	31	55,000

A `FROM` clause can be followed by `JOIN` sub-clauses that are used to combine multiple tables based on common values in specific columns. An example of such a `JOIN` query is "`SELECT` $\mathbf{T_1}$.name, $\mathbf{T_2}$.department `FROM` $\mathbf{T_1}$ `JOIN` $\mathbf{T_2}$ `ON` $\mathbf{T_1}$.employee_id = $\mathbf{T_2}$.employee_id," which returns:

The result of any SQL query is itself a table to which another SQL query can be applied. In other words, a SQL query may be composed of several sub-queries. SQL also provides queries for adding, updating, and deleting data from a SQL database. Data-manipulation queries start with an `INSERT`, `DELETE`, or `UPDATE`

name	department
Kate	Sales
Lisa	HR
Alice	R&D

clause followed by a table identifier, a series of values, and (optionally) a sequence of `WHERE` clauses. For example, the query "`DELETE FROM` $\mathbf{T_2}$ `WHERE` department = Sales" deletes the first two rows from $\mathbf{T_2}$. Finally, there are queries that manipulate the database structure, e.g, by adding new columns or creating a new table.

Note that a common theme of the examples presented above is that they process each row of some table independently, performing a specific operation (e.g., comparing values from given columns with a specified range) on each row. This structure can be leveraged to improve efficiency of our argument system, as noted in Remark 2.

4.2 Related Work on Verifiable Databases

Most existing work on verifiable databases comes from *authenticated data structures* [77]. It typically focuses on handling only a specific class of computations on the outsourced database, e.g., range queries [62, 67], joins [43, 87, 94], pattern matching [40, 68] and set operations [32, 61, 70, 92]. The most relevant point of comparison to our work is IntegriDB [91], which supports a subset of SQL. In Section 4.6, we show that vSQL is significantly more expressive than IntegriDB while

enjoying comparable efficiency.

4.3 Definitions of Verifiable Databases

In this section, we present our security definition for a verifiable database system, viewed as a two-party protocol run between a *client* that owns a database D which it wishes to outsource to a remote *server*. In a setup phase, the client computes a short digest of D, which it stores locally, and uploads D to the server. Subsequently, he issues queries about the data or requests to update the data, which are processed by the server. Each query evaluation is executed by an interactive protocol between the two parties, at the end of which the client either accepts the returned output or rejects it. Informally, the required security property is that no computationally bounded adversarial server can convince the client into accepting a false result. This is defined formally in Definition 4. To simplify notation, we do not distinguish between verification parameters (that are stored by the client and should be succinct) and proof-computation parameters (stored by the server).

Definition 4. *A verifiable database system for database class $\mathcal{D}$ and query class $\mathcal{Q} = \mathcal{U} \cup \mathcal{S}$ (where $\mathcal{U}$ denotes update queries and $\mathcal{S}$ denotes selection queries), is a tuple of algorithms defined as follows:*

1. Setup *takes as input 1^λ, a database $D \in \mathcal{D}$ and outputs a digest δ and public parameters* pp.

2. $\mathsf{Evaluate}$ *is an interactive protocol run between two probabilistic polynomial-time algorithms* C *and* S *on common input a digest δ, a query $Q \in \mathcal{Q}$, and public*

parameters pp*. Moreover,* S *holds database* D*. If* $Q \in \mathcal{S}$*, then at the end of the protocol* C *either outputs a result* y *(and accepts) or rejects. If* $Q \in \mathcal{U}$*, then at the end of the protocol* C *outputs a new digest* δ' *(and accepts), or rejects.*

Denote by $Q(D)$ *the evaluation of query* Q *on database* D*. We require that* Setup *and* Evaluate *have the following properties.*

- ***Perfect completeness.*** *For any* λ*, any* $D_0 \in \mathcal{D}$*, any* $t \geq 0$*, and any queries* $Q_1, \ldots, Q_t \in \mathcal{Q}$ *and* $Q^* \in \mathcal{S}$*, we require that* $y = Q^*(D_t)$ *in the following experiment:*

  - Setup *is invoked on the input* $(1^\lambda, D_0)$ *and outputs* (δ_0, pp)*.*
  - *For* $1 \leq i \leq t$*, do:* S *and* C *run* Evaluate *on inputs* $(Q_i, \delta_{i-1}, D_{i-1}, \mathsf{pp})$ *and* $(Q_i, \delta_{i-1}, \mathsf{pp})$*, respectively. If* $Q_i \in \mathcal{U}$*, let* δ_i *denote the output of* C *and set* $D_i = Q_i(D_{i-1})$*; otherwise, set* $\delta_i = \delta_{i-1}$ *and* $D_i = D_{i-1}$*.*
  - S *and* C *run* Evaluate *on inputs* $(Q^*, \delta_t, D_t, \mathsf{pp})$ *and* $(Q^*, \delta_t, \mathsf{pp})$*, respectively. Let* y *denote the output of* C*.*

- ***Soundness.*** *For any* t *and polynomial-time attacker* S^**, the probability that* S^* *succeeds in the following experiment is negligible:*

  1. $\mathsf{S}^*(1^\lambda)$ *outputs* $D_0 \in \mathcal{D}$ *.*
  2. $\mathsf{Setup}(1^\lambda, D_0)$ *outputs* (δ_0, pp)*.*
  3. *For* $1 \leq i \leq t$*, do:* S^* *outputs* Q_i*. Then* S^* *and* C *run* Evaluate *on inputs* $(Q_i, \delta_{i-1}, D_{i-1}, \mathsf{pp})$ *and* $(Q_i, \delta_{i-1}, \mathsf{pp})$*, respectively. If* C *rejects, the experi-*

ment ends. If C *accepts and* $Q_i \in \mathcal{U}$*, let* δ_i *denote the output of* C *and set* $D_i = Q_i(D_{i-1})$*; otherwise, set* $\delta_i = \delta_{i-1}$ *and* $D_i = D_{i-1}$*.*

4. S^* *outputs* $Q^* \in \mathcal{S}$*. Then* S^* *and* C *run* $\mathsf{Evaluate}$ *on inputs* $(Q^*, \delta_t, D_t, \mathsf{pp})$ *and* $(Q^*, \delta_t, \mathsf{pp})$*, respectively. Let* y *denote the output of* C*. We say that* S^* *succeeds if* C *accepts with output* y*, but* $y \neq Q^*(D_t)$*.*

Supporting database size increases. For some constructions (including ours), the size of the public parameters pp may depend on the database size. If the database size increases (as a result of updates), it may be necessary to extend pp; there are various ways this can be done. For instance, the database owner can choose an upper bound for the database size, and generate a long-enough pp during the setup phase. Alternatively, the owner may maintain some (succinct) trapdoor information that allows it to extend pp as needed.

Efficiency considerations. One important aspect of a verifiable database system is efficiency; a trivial approach is to transmit D for each query and have the client evaluate it himself. Therefore, a basic efficiency requirement is that the communication between client and server for query evaluation should be sublinear in the database size $|D|$. Also important is the client's computational cost for, which should ideally be smaller than evaluating the query (so the client can benefit not only from delegation of its storage but also from delegation of its computation). A final efficiency metric is the computational overhead of the server, which should ideally be asymptotically the same as the cost of evaluating the query.

4.4 Our Construction of Verifiable Databases

In this section, we present our construction of a verifiable database system. Recall that we refer to the prover and verifier of the CMT protocol as $(\mathcal{P}^{cmt}, \mathcal{V}^{cmt})$, and we refer to the algorithms of our polynomial-delegation protocol as (KeyGen, Commit, Evaluate, Ver).

The only modification on our argument system presented in Construction 3 is that the client also wants to outsource the storage of her input, the database D, to the server. To do so, initially, the client views its database D as an array of $|D|$ elements (where $|D|$ is equal to number of rows times number of columns) and computes the multilinear extension $\tilde{D}$. Note that the number of variables in $\tilde{D}$ is logarithmic in the total size of D. Next, the client generates a commitment com to $\tilde{D}$ using our polynomial-delegation protocol, stores com locally, and uploads D to the untrusted server. We stress that this phase does not depend on any specific queries the client may choose to issue later.

Construction 4. *Let λ be a security parameter, let D be a database and let $\mathbb{F}$ be a prime-order field with $|\mathbb{F}|$ exponential in λ.*

Setup phase. *On input 1^λ and a database $D \in \mathcal{D}$, the client picks a parameter $N \geq |D|$ such that $N \in O(|D|)$, which denotes an upper bound on the size of databases (in terms of values in the database) that can be supported, and sets $n = \lceil \log N \rceil$. Let $\tilde{D}$ denote the multilinear extension of D. The client runs $\mathsf{KeyGen}(1^\lambda, n, 1)$ to compute public parameters pp, and $\mathsf{Commit}(\tilde{D}, \mathsf{pp})$ to compute commitment com on $\tilde{D}$. It*

then sends $(D, \mathsf{pp}, \mathsf{com})$ *to the server and stores* $(\mathsf{pp}, \mathsf{com})$.

Evaluation phase. *Let* $(x_0, \ldots, x_{N-1})$ *be the current version of the database* D *stored by the server and let* com *be the commitment stored by both client and server. Given a query* $Q \in \mathcal{Q}$*, let* C *be a depth-*d *circuit over* $\mathbb{F}$ *that evaluates* Q *on input* D *and (possibly empty) auxiliary input* $B \in \mathbb{F}^{|B|}$*. Assume w.l.o.g. that* $|B| = (2^m - 1) \cdot N$ *for some integer* m*. Partition the input of* C *into* 2^m *arrays* $(B_1, \ldots, B_{2^m})$ *each of size* N *with* B_1 *corresponding to* D *and the rest corresponding to the auxiliary input. Finally, let* $\tilde{B}_1, \ldots, \tilde{B}_{2^m}$ *denote the corresponding multilinear extensions of* $B_1, \ldots, B_{2^m}$ *where* $\tilde{B}_1 = \tilde{D}$*.*

- *If* Q *is a selection query, the two parties then interact as follows:*

  1. S *computes the necessary auxiliary input* $B_2, \ldots, B_{2^m}$*, and runs* $\mathsf{Commit}(\tilde{B}_i, \mathsf{pp})$ *for* $2 \leq i \leq 2^m$ *to obtain values* $\mathsf{com}_2, \ldots, \mathsf{com}_{2^m}$*, which it sends to* C*.*

  2. C *runs* $\mathcal{V}^{cmt,1+2}$ *and* S *runs* $\mathcal{P}^{cmt}$ *to evaluate* $C(B_1, \ldots, B_{2^m})$*. If* $\mathcal{V}^{cmt,1+2}$ *rejects at any point,* C *outputs* $\mathsf{0}$*. Otherwise, let* r_d, a_d *be the final values returned by* $\mathcal{V}^{cmt,1+2}$*. Let* $\tilde{V}_d$ *be the multilinear extension of the input layer of* C*. At this point,* C *must verify that* $\tilde{V}_d(r_d) = a_d$*, which is done as follows.*

  3. C *sends to* S *values* $\rho^{(1)}, \ldots, \rho^{(2^m)} \in \mathbb{F}^{n-1}$ *chosen uniformly at random.*

  4. S *parses* r_d *as* $r_d := (\kappa_1, \ldots, \kappa_{m+n})$ *and defines* $r'_d := (\kappa_{m+1}, \ldots, \kappa_{m+n})$*.* S *then sends to* C *the evaluations* $(v_1, \ldots, v_{2^m})$ *of polynomials* $\tilde{B}_1(r'_d), \ldots, \tilde{B}_{2^m}(r'_d)$ *along with corresponding proofs* π_i *computed by* $\mathsf{Evaluate}(\tilde{B}_i, r'_d, \rho^{(i)}, \mathsf{pp})$*, for all* $1 \leq i \leq 2^m$*.*

5. C *runs* $\mathsf{Ver}(\mathsf{com}_i, r'_d, v_i, \pi_i, \rho^{(i)}, \mathsf{pp})$ *for* $1 \leq i \leq 2^m$. *If any execution outputs* 0, C *outputs* 0. *Otherwise,* C *defines* $r''_d := (\kappa_1, \ldots, \kappa_m)$ *and computes* $\tilde{V}_d(r_d)$ *by combining values* $v_1, \ldots, v_{2^m}$ *as per Equation 2.1. If* $\tilde{V}_d(r_d) \neq a_d$, C *outputs* 0, *otherwise* 1.

6. *The output of* S *is set to* $C(B_1, \ldots, B_{2^m})$.

- *If* Q *is an update query, the two parties then interact as follows:*

  1. S *computes the necessary auxiliary input* $B_2, \ldots, B_{2^m}$, *and runs* $\mathsf{Commit}(\tilde{B}_i, \mathsf{pp})$ *for* $2 \leq i \leq 2^m$ *computing values* $\mathsf{com}_2, \ldots, \mathsf{com}_{2^m}$. *Moreover, it computes the multilinear extension* $\tilde{V}_{out}$ *of the output of* $C(B_1, \ldots, B_{2^m})$ *and runs* $\mathsf{Commit}(\tilde{V}_{out}, \mathsf{pp})$ *to compute output commitment* com_{out}. *Finally, it sends* $\mathsf{com}_{out}, \mathsf{com}_2, \ldots, \mathsf{com}_{2^m}$ *to* C.

  2. C *chooses* $r_0 \in \mathbb{F}^n$, *(the output of* C *is the entire new database which by assumption is at most* N *therefore its multilinear extension operates on* $n = \log N$ *elements), and sends it to the server along with a uniform value* $\rho_{out} \in \mathbb{F}^{n-1}$.

  3. S *responds with* $a_0 = \tilde{V}_{out}(r_0)$ *and corresponding proof* π_{out} *computed with* $\mathsf{Evaluate}(\tilde{V}_{out}, r_0, \rho_{out}, \mathsf{pp})$.

  4. C *runs* $\mathsf{Ver}(\mathsf{com}_{out}, r_0, a_0, \pi_{out}, \rho_{out}, \mathsf{pp})$ *and rejects if it outputs* 0.

     Otherwise, C *runs* $\mathcal{V}^{cmt,2}$ *while* S *runs* $\mathcal{P}^{cmt,2}$ *on common input* r_0, a_0. *If* $\mathcal{V}^{cmt,2}$ *rejects at any point,* C *outputs* 0. *Otherwise, let* r_d, a_d *be the final values returned by* $\mathcal{V}^{cmt,2}$. *Let* $\tilde{V}_d$ *be the multilinear extension of the input*

layer of C. At this point, C must verify that $\tilde{V}_d(r_d) = a_d$. This is achieved by having C and S perform steps 3–5 from above.

5. *The output of S is set to $C(B_1, \ldots, B_{2^m})$ and com_{out}. If C accepts, it sets $\mathsf{com} \leftarrow \mathsf{com}_{out}$.*

We have the following theorem:

Theorem 6. *If Construction 2 is an extractable, verifiable polynomial-delegation protocol, then Construction 4 is a verifiable database system for SQL queries.*

If Construction 4 is executed on a database D with $|D|$ values, to evaluate a query expressed as a non-deterministic, depth-d arithmetic circuit C with at most S gates per layer, that consists of parallel copies of a circuit C' with at most S' gates per layer, followed by a post-processing circuit C'' of size $O(|C|/\log|C|)$, and with auxiliary input B, then

1. *The running time of Setup is $O(|D|)$.*

2. $\mathsf{Evaluate}$ *requires $O(d \log S)$ rounds of interaction.*

3. *The running time of C is $O(k + d \cdot \mathrm{polylog}(S) + \lceil |B|/|D| \rceil \log(|D|))$, where k is the size of the result for selection queries and k is $O(\log|D'|)$ for updates (D' is the output size).*

4. *The running time of S is $O(|C| \cdot \log S' + (|B| + |D|) \cdot \mathrm{polylog}(|B| + |D|))$.*

Proof. The correctness requirement immediately follows from the construction of $(\mathsf{Setup}, \mathsf{C}, \mathsf{S})$. We now proceed to analyze the complexities of Setup, $\mathsf{Evaluate}$, C and

S. First note that the multilinear extension for D, as computed by Setup, contains N terms. Thus, using an optimization from [69, Appendix D], $\mathsf{KeyGen}, \mathsf{Commit}$ take time $O(N) = O(|D|)$ which is the total running time for Setup. The number of rounds of interaction during $\mathsf{Evaluate}$ follow directly by the execution of the CMT protocol. The overall running time of C for a selection query results from executing the first two steps of $\mathcal{V}$ from Construction 1 which has running time of $O(k + d \cdot \text{polylog}(S))$ from Theorem 2 (as the last step of $\mathcal{V}$ is omitted). This is followed by a series of $O(\lceil |B|/|D| \rceil)$ executions of Ver from Construction 2, each for a polynomial of $\log |D|$ variables and variable-degree 1. This yields an overall running time of $O(\lceil |B|/|D| \rceil \cdot \log |D|)$ For updates, the only difference is that the first step of $\mathcal{V}$ is replaced by an execution of Ver from Construction 2 for a polynomial of $\log |D|$ variables and variable-degree 1, which is subsumed by $O(\text{polylog}(S))$. The running time of S (again using the optimization from [69]) is the sum of the running time of $\mathcal{P}$ in Construction 1 ($O(|C| \cdot \log S'$) as per Remark 2) and a number of $\mathsf{Evaluate}$ from Construction 2 for $\ell = \log |D|$ and $d = 1$. Using an FFT for polynomial division (as in [69]), the cost for $\mathsf{Evaluate}$ can be upper bounded by $O((|B| + |D|) \cdot polylog(|B| + |D|))$.

Finally, we prove the soundness property. Assume that there exists an adversary Adv which is able to break the soundness of the verifiable database system. We will use Adv in order to construct an adversary Adv' which is capable of either breaking the soundness property of our verifiable polynomial-delegation protocol or the soundness of the CMT interactive protocol.

Indeed, let Adv be an adversary which is capable of breaking the security of

Construction 3. We build Adv' as follows.

1. On input security parameter 1^λ Adv' runs $\mathsf{Adv}(1^\lambda)$ to receive database D_0 and picks large enough bound parameter N.

2. Adv' plays the security game of the verifiable polynomial-delegation scheme (Definition 3). He sends variable parameter $n = \lceil \log N \rceil$ and degree parameter $d = 1$ and receives $\mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, n, 1)$ and sends pp to Adv.

3. Adv' computes the multilinear extension $\tilde{D}_0$ of D_0 (parsed as an array of N elements) and computes $\mathsf{com}^0 \leftarrow \mathsf{Commit}(\tilde{D}_0, \mathsf{pp})$. He then sends $(\mathsf{com}^0, \mathsf{pp})$ to Adv. The latter responds with upper bound t.

4. Adv' chooses i' uniformly at random from $[t]$. Then, for all $1 \leq i < i'$, Adv' performs the following.

   (a) Adv' receives query Q_i and $2^m - 1$ commitments $\mathsf{com}^i_2, \ldots, \mathsf{com}^i_{2^m}$ related to auxiliary inputs B^i.

      i. Q_i **is a selection query** $(Q_i \in \mathcal{S})$**.** In this case Adv sends Adv' the query's output y_i.

      ii. Q_i **is an update query** $(Q_i \in \mathcal{U})$**.** In this case Adv sends Adv' a commitment com^i_{out} corresponding to the query's result (instead of the query's output).

   (b) Adv' emulates Adv's interaction with C during the $\mathsf{Evaluate}$ protocol. If the output of C is 0, Adv' aborts. Otherwise, if $Q_i \in \mathcal{S}$ Adv' sets $D_i \leftarrow D_{i-1}$ and $\mathsf{com}^i \leftarrow \mathsf{com}^{i-1}$. If $Q_i \in \mathcal{U}$ Adv' sets $D_i \leftarrow Q_i(D_{i-1})$ and $\mathsf{com}^i \leftarrow \mathsf{com}^i_{out}$.

5. **If** $Q_i \in \mathcal{S}$ **and** $y_i = Q_i(D_{i-1})$ Adv' aborts.

6. **If** $Q_i \in \mathcal{U}$ **and** $\mathsf{Commit}(Q_i(D_{i-1})) = \mathsf{com}^i_{out}$ Adv' aborts.

7. **If** $Q_i \in \mathcal{S}$ **and** $y_i \neq Q_i(D_{i-1})$ For all $2 \leq j \leq 2^m$ Adv' runs the extractor $\mathcal{E}_j$ (see below for a description) that outputs the pre-image of com_j As a result Adv' receives an output o_j. Adv' attempts to parse o_j as a variable-degree-1 n-variate polynomial $\tilde{B}_j$. If this step fails Adv' aborts. Otherwise, Adv' proceeds as follows.

   (a) Since $(\tilde{B}_2, \ldots, \tilde{B}_{2^m})$ are variable-degree-1 n-variate polynomials, $\tilde{B}_j$ is a valid multilinear extension of an array of N elements B_j, for all $2 \leq j \leq 2^m$ (as per Section 2.2).

   (b) Adv' initializes the soundness game with $\mathcal{V}^{cmt}$ (as per Definition 1) on $C_i, (D_{i-1}, B_2, \ldots, B_{2^m}, y_i)$, where C_i is the depth$-d$ circuit representation of Q_i.

   (c) Adv' emulates Adv's interaction with C during the $\mathsf{Evaluate}$ protocol. At any point where Adv requires a random point from C, Adv' requests a random point from $\mathcal{V}^{cmt}$ and forwards it to Adv.

   (d) At the end of the execution of $\mathsf{Evaluate}$ for the round corresponding to the input layer of C_i, let a_d be the claimed evaluation of the multilinear extension of the inputs on point r_d, as provided by Adv.

   (e) At this point, Adv requires from C randomized challenges $\rho^{(1)}, \ldots, \rho^{(2^m)}$ to produce the evaluations of the 2^m multilinear extensions of the input chunks of C_i on challenge point r'_d. Adv' requests $\rho^{(1)}$ from the verifiable polynomial

delegation challenger, generates $\rho^{(2)}, \ldots, r^{(2^m)}$ uniformly at random himself and sends them all to Adv. Adv responds with evaluations $v_1, \cdots, v_{2^m}$ and proofs $\pi_1, \ldots, \pi_{2^m}$.

(f) **It holds that** $v_1 \neq \tilde{D}_{i-1}(r'_d)$**.** In this case Adv' outputs $(\mathsf{com}^{i-1}, r'_d, v_1, \pi_1, \rho^{(1)}, \mathsf{pp})$ as the challenge input to Ver and halts.

(g) **It holds that** $v_1 = \tilde{D}_{i-1}(r'_d)$**.** Adv' checks that $\tilde{B}_j(r'_d) = v_j$, for all $2 \leq j \leq 2^m$. If any check fails, he aborts. Otherwise, he outputs a_d as the final evaluation to $\mathcal{V}^{cmt}$, where a_d is the claimed evaluation received by Adv' for the evaluation of the multilinear extension of the input of C_i on point r_d and halts.

8. **If** $Q_i \in \mathcal{U}$ **and** $\mathsf{Commit}(Q_i(D_{i-1})) \neq \mathsf{com}^i_{out}$ For all $2 \leq j \leq 2^m$ Adv' runs the extractor $\mathcal{E}_j$ that outputs a pre-image of com_j (see below for a description), receiving output o_j. Moreover, Adv' runs the extractor $\mathcal{E}_{out}$ that outputs a pre-image of com_{out} receiving output o'. Adv' attempts to parse each of o_j as a variable-degree-1 n-variate polynomial $\tilde{B}_j$ and likewise for o' and a variable-degree-1 n-variate polynomial $\tilde{Y}$. If this step fails Adv' aborts. Otherwise, Adv' proceeds as follows.

(a) Since $(\tilde{B}_2, \ldots, \tilde{B}_{2^m}, Y)$ are variable-degree-1 n-variate polynomials, $\tilde{B}_j$ is a valid multilinear extension of an array of N elements B_j, for all $2 \leq j \leq 2^m$ (as per Section 2.2). Likewise, $\tilde{Y}$ is a valid multilinear extension of an array Y of N elements.

(b) Adv' initializes the soundness game with $\mathcal{V}^{cmt}$ (as per Definition 1) on

$C_i, (D_{i-1}, B_2, \ldots, B_{2^m}, Y)$, where C_i is the depth$-d$ circuit representation of Q_i.

(c) Adv' emulates Adv's interaction with C during the $\mathsf{Evaluate}$ protocol. At any point where Adv requires a random point from C, Adv' requests a random point from $\mathcal{V}^{cmt}$ and forwards it to Adv.

(d) During the execution of $\mathsf{Evaluate}$ for the round corresponding to the output layer of C_i, Adv requires from C randomized challenge $\rho^{(out)}$ to produce the evaluation of the multilinear extension of the output of C_i on point r_0. Adv' generates $\rho^{(out)}$ uniformly at random and sends it to Adv. Adv responds with evaluation v_{out} and proof π_{out}.

(e) Adv' checks that $\tilde{Y}^i(q_0) = v^i_{out}$. If the check fails, he aborts. Otherwise he continues emulating Adv's interaction with C during the $\mathsf{Evaluate}$ protocol.

(f) Adv' proceeds to run steps 7d-7g above.

Since Adv and the algorithms of the verifiable database scheme run in time polynomial in λ, it follows that Adv' also runs in time polynomial in λ. Let us now argue about the success probability of Adv'.

We begin by defining the extractors $\mathcal{E}_j$ necessary for retrieving the commitment pre-images. Let μ be the number of commitments output by Adv during round i' ($\mu = 2^m - 1$ or 2^m depending on the type of Q_i), and $T_{i'}$ be the transcript that consists of all the inputs received by Adv during the $i' - 1$ first rounds.

Let Adv'' denote the adversary that $1^\lambda, \mathsf{pp}$ and $T_{i'}$ as auxiliary input z_1, and interacts internally with Adv as follows.

- Use z_1 to recreate the state of Adv right until the beginning or round i'.
- Upon receiving $\mathsf{com}_1, \ldots, \mathsf{com}_\mu$ in the beginning of round i', output them.
- Receive point r'_d, randomized challenges $\rho^{(1)}, \ldots, \rho^{(\mu)}$, and auxiliary input z_2 that consists of all the messages exchanged during the CMT execution in round i'.
- Use z_2 to recreate the state of Adv until the end of the CMT execution in round i'.
- Upon receiving evaluations $v_1, \ldots, v_\mu$ and proofs $\pi_1, \ldots, \pi_\mu$ from Adv, output them.

Finally, let Adv''_j for $j = 1, \ldots, \mu$ denote the adversary that receives the same input as Adv'' and runs the exact same code, but only outputs com_j initially and v_j, π_j finally.

By the knowledge soundness property of our verifiable polynomial delegation protocol, for each Adv''_j there exists $\mathcal{E}_j$ that on input $(1^\lambda, \mathsf{pp}, z_1)$ outputs variable -degree 1, n-variate polynomial $\tilde{B}_j$ such that, if Ver accepts, then $\mathsf{Commit}(\tilde{B}_j, \mathsf{pp}) = \mathsf{com}_j$ and $\tilde{B}_j(r'_d) = v_j$ with all but negligible probability. Here, we need to make the assumption that the auxiliary inputs z_1, z_2 comes from a benign distribution.[1]

[1] We stress that, in our construction, z_1, z_2 consist of values that are chosen uniformly at random from their respective domains (random evaluation points for the CMT execution and randomized challenges for the verifiable polynomial delegation), therefore assuming that this distribution is benign seems like a mild assumption.

We denote by E_{Adv} the event that Adv wins the soundness game of the verifiable database system (as per Definition 4) and by $E_{\mathsf{Adv}'}$ the event that Adv' wins the soundness game of the verifiable polynomial-delegation protocol (as per Definition 3) or the soundness game of the interactive protocol (as per Definition 1). By definition, E_{Adv} takes place if and only if there exists $1 \leq i \leq t$ such that $Q_i \in \mathcal{S} \wedge b_i = 1 \wedge y_i \neq Q_i(D_{i-1})$. We denote by E_1 the event that there exists $1 \leq i'' \leq t$ such that $Q_{i''} \in \mathcal{U} \wedge b_{i''} = 1 \wedge \mathsf{Commit}(Q_{i''}(D_{i''-1}) \neq \mathsf{com}_{out}^{i''}$, i.e., the event that Adv produced an update query and an incorrect commitment for round i''. Note that $\Pr[E_{\mathsf{Adv}}] = \Pr[E_{\mathsf{Adv}} \wedge E_1] + \Pr[E_{\mathsf{Adv}} \wedge E_1^c)$ where E^c denotes the complement of event E.

Regarding the probability that Adv' wins we can write

$$\begin{aligned}
\Pr[E_{\mathsf{Adv}'}] \geq{}& \Pr[E_{\mathsf{Adv}'} | E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''] \Pr[E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''] \\
&+ \Pr[E_{\mathsf{Adv}'} | E_{\mathsf{Adv}} \wedge E_1^c \wedge i' = i] \Pr[E_{\mathsf{Adv}} \wedge E_1^c \wedge i' = i] \\
={}& \frac{1}{t}\Big(\Pr[E_{\mathsf{Adv}'} | E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''] \Pr[E_{\mathsf{Adv}} \wedge E_1]) \\
&+ \Pr[E_{\mathsf{Adv}'} | E_{\mathsf{Adv}} \wedge E_1^c \wedge i' = i] \Pr[E_{\mathsf{Adv}} \wedge E_1^c] \Big)
\end{aligned}$$

where the last step follows from the fact that i' is chosen uniformly at random.

We now lower-bound $\Pr[E_{\mathsf{Adv}'} | E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i'']$ as follows. Conditioned on $E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''$, Adv' provides a perfect emulation of C to Adv during the first $i'' - 1$ rounds. Moreover, assuming Adv' does not abort in round i'', he provides a perfect emulation for both Adv' and $\mathcal{V}^{cmt}$ for round i'' up to, and including, step 7d. If $v_1 \neq \tilde{D}_{i''-1}(r'_d)$ then the tuple $(\mathsf{com}^{i-1}, r'_d, v_1, \pi_1, \rho(1)_i, \mathsf{pp})$ is indeed a valid challenge for the soundness game of the verifiable polynomial protocol, therefore Adv'

wins. Otherwise, $v_1 = \tilde{D}_{i''-1}(r'_d)$, and it follows that a_d is the correct evaluation of the multilinear extension of the input layer of C_i, therefore Adv' wins. From the above it follows that $\Pr[E_{\mathsf{Adv}'}|E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''] = 1 - \Pr[\mathsf{Adv}'$ aborts in round $i''|E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i'']$. Note that conditioned on $E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''$, Adv' will only abort if any of the checks in steps 8, 8e, 7g fail, i.e., the extraction of the pre-image polynomials for output and auxiliary input fails. By the knowledge soundness property of the verifiable delegation protocol (and since, conditioned on E_{Adv}, Ver which is a subroutine of C outputs $\mathsf{1}$ for all of them), each of them can fail with probability $\mathsf{neg}(\lambda)$. Since there are only $O(poly(\lambda))$ many of them, by a simple union bound we get that $\Pr[E_{\mathsf{Adv}'}|E_{\mathsf{Adv}} \wedge E_1 \wedge i' = i''] \geq 1 - \mathsf{neg}(\lambda)$

We can lower-bound $\Pr[E_{\mathsf{Adv}'}|E_{\mathsf{Adv}} \wedge E_1^c \wedge i' = i]$ in a similar manner. Conditioned on $E_{\mathsf{Adv}} \wedge E_1^c \wedge i' = i$, Adv' provides a perfect emulation for Adv for all rounds before the ith, and for both Adv and $\mathcal{V}^{cmt}$ for round i up to, and including, step 7d. With the same argument as above, under these conditions Adv' always wins the game unless he aborts at round i. This last probability is again negligible, i.e., $\mathsf{neg}'(\lambda)$ (for some neg' not necessarily equal to neg defined above) due to the knowledge soundness property of the verifiable delegation protocol. By substituting in the above equation (and assuming without loss of generality that $\mathsf{neg} \geq \mathsf{neg}'$) we get

$$\begin{aligned}\Pr[E_{\mathsf{Adv}'}] &\geq \frac{1}{t}\Big((1 - \mathsf{neg}(\lambda))(\Pr[E_{\mathsf{Adv}} \wedge E_1] + \Pr[E_{\mathsf{Adv}} \wedge E_1^c])\Big) \\ &= \frac{1}{t}\Big((1 - \mathsf{neg}(\lambda))\Pr[E_{\mathsf{Adv}}]\Big) = \frac{\Pr[E_{\mathsf{Adv}}]}{t} - \mathsf{neg}(\lambda).\end{aligned}$$

Assuming Adv wins with non-negligible probability in λ, we get that Adv' also wins

with non-negligible probability in λ (since t is polynomial in λ) which concludes our proof.

□

4.5 Optimizations for SQL Queries

In this section, we introduce additional optimizations to improve the efficiency of our argument system for SQL queries. In particular, we leverage the ability of our scheme to efficiently handle auxiliary inputs in order to: (i) achieve faster equality testing (which is useful for selection queries), (ii) allow for input/output gates at arbitrary layers of the circuit with minimal overhead, and (iii) verify the results of set intersections using a smaller number of gates (which is useful for join queries). Finally, we discuss how to support expressive SQL updates, and how simple updates (that consist of assigning values to unused table cells) can be verified using one round of interaction.

Most of the optimizations discussed below exploit various techniques for constructing efficient representations of computations commonly when answering SQL queries. These techniques include modifying the queries' circuit representations in order to utilize auxiliary inputs, encoding some of the query computations directly as polynomials, and utilizing interaction in order to reduce the circuit size. Since these modifications are applied directly to the underlying circuit being computed, security when using these optimizations follows readily from security of our protocol.

4.5.1 Optimizing Equality Testing

A very common subroutine used in both selection and join queries is testing whether two values are equal, which can be reduced to testing whether their difference is 0. Here we show how we can efficiently perform such zero tests using auxiliary input provided by the prover.

Optimized zero testing. Ideally, we would like a small arithmetic circuit that takes as input a field element x and outputs $x' = 0$ if $x = 0$ and $x' = 1$ otherwise. It is well known [36] that, by relying on Fermat's little theorem, this can be done by computing $x' = x^{p-1}$ (where p is the field size). This approach is relatively expensive, however, since it requires a circuit of size and depth $O(\log p)$. Instead, we will construct a non-deterministic circuit for this task that has two outputs x', z and satisfies the following: $x = 0$ iff there is an auxiliary input y such that $x' = 0$ and $z = 0$; also, $x \neq 0$ iff there is an auxiliary input y such that $x' = 1$ and $z = 0$. Thus, the rest of the computation can use x', and the client will additionally verify that $z = 0$.

We can achieve the above by computing $x' = xy$ and $z = x \cdot (1 - xy)$. Note that setting $y = x^{-1}$ if $x \neq 0$ (and setting y arbitrarily otherwise) yields correct values for x' and z. Moreover, if $x = 0$ then $x' = z = 0$ for any choice of y, and if $x \neq 0$ then the only way to force $z = 0$ is to set $x' = 1$. We note that the same high-level idea has appeared before (e.g., [71,76]) in the context of SNARKs that are defined based on constraint systems. In our case, the CMT protocol only supports the evaluation of arithmetic circuits (and not constraint systems), and so we need

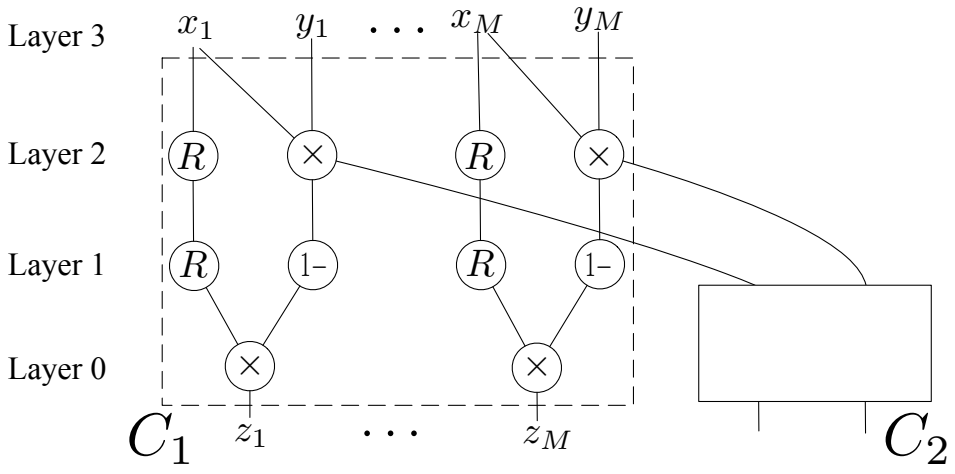

Figure 4.1: Zero testing. If $z_i = 0$ then the input to C_2 is a 0/1 value indicating whether x_i is zero.

a slightly different technique.

Enforcing zero values. A trivial implementation of the above would require the server to send all the x', y values to the client, resulting in the client performing work linear in the number of zero tests. Since zero testing may be done at least once per database row, this will lead to large overheads.

Instead (cf. Figure 4.1), we split the computation into two parts: (i) a circuit C_1 that computes $z = x(1 - xy)$, and (ii) a circuit C_2 that evaluates the SQL query using the result of the zero test (i.e., $x' = xy$). Without loss of generality, we assume the result of the zero test is used at the input layer of C_2, as shown in Figure 4.1. The client and the server will run two separate interactive proof protocols for C_1 and C_2. First, the protocol for C_2 is executed up to one layer before its input layer (i.e., the client and server pause before proceeding to its input layer). After that, the protocol for C_1 is initiated. Note that the honest prover does not need to send

any of the outputs of C_1 to the verifier since the verifier knows all of them are supposed to be 0. Moreover, in order to initiate the execution of this protocol, the verifier needs to compute the multilinear extension of the outputs of C_1 evaluated at a random point. Since the multilinear extension of the 0-vector is the 0-polynomial, this step is free. Once the interactive protocol for C_1 finishes layer 1, the verifier uses *the same randomness* for the next layer of both circuits (layer 2 of C_1 and the input layer of C_2, which have the same values).[2] This reduces the claims in both executions to a single evaluation of the multilinear extension of the joint input for that layer. Finally, layer 3 (the input layer) of C_1 is verified normally. In this way, the prover's overhead for zero testing is only linear in the size of C_1, which only has 3 layers. The verifier's overhead is only polylogarithmic in the size of C_1.

In our experiments (where $\lceil \log p \rceil = 254$), the above zero testing and enforcement method yield an 80× speedup for both prover and verifier compared to the deterministic approach using Fermat's little theorem.

Handling conjunctions and disjunctions. In multi-dimensional SQL selection queries, `AND` or `OR` operators are applied on the results of multiple selection clauses over different columns, and thus the number of zero tests required potentially grows with the number of columns. But note that `OR` clauses can be trivially reduced to a single zero test; e.g., testing $x_1 = 0 \vee x_2 = 0$ reduces to testing $x_1 x_2 = 0$. We further observe that `AND` clauses can also be reduced to a single zero test if the input values are known to be in a bounded range. For example, if it is known

[2]Using the same randomness for both C_1 and C_2 does not affect the soundness of the CMT protocol here.

that $-\sqrt{p/2} < x_1, x_2 < \sqrt{p/2}$ then we may reduce evaluating the conjunction $x_1 = 0 \wedge x_2 = 0$ to evaluating whether $x_1^2 + x_2^2 = 0$. In particular, if all values in question are 32 bits long and p is a 254-bit value, then we can test conjunctions involving up to 2^{189} values using just a single zero test. Alternatively, we can handle conjunctions using packing: e.g., if x_1, x_2 are 32-bit values (and $|p| > 64$) then testing whether $x_1 = 0 \wedge x_2 = 0$ is equivalent to testing whether $2^{32}x_1 + x_2 = 0$. These approaches ensure the number of required auxiliary inputs (as well as the size of the zero-test circuit) for a multi-dimensional selection query depends linearly on the number of rows in the table and is almost independent of the number of columns involved in the query.

4.5.2 Supporting Inputs/Outputs at Arbitrary Circuit Layers

So far, we have assumed that the circuit being computed takes all its inputs at the same layer, and produces all its outputs at the same layer. This is without loss of generality since one can always define a "relay" gate that simply passes its input to the next layer. In practice however, such relay gates will contribute some cost to the execution of the interactive-proof protocol [81]. For many natural SQL queries, this might even result in a highly inefficient circuit where most gates are relay gates. For example, consider an SQL query of the form `SELECT * FROM` $\mathbf{T}$ `WHERE` $\mathrm{col}_i = x$. A circuit for evaluating this query takes the entire table as input, but only values from the ith column are involved in the selection process. All the other values, from all other columns, are simply relayed between the various circuit layers.

Avoiding relaying the inputs. We now describe a technique that avoids relay gates by leveraging the property of the multilinear extension described in Section 2.2. Concretely, consider a circuit C such that some internal layer k operates on $2M$ values with $m = \lceil \log M \rceil$. Assume the second half (denoted by B) of the $2M$ values are "fresh inputs" (these may either be from the database itself, or auxiliary input from the prover), while the first half (denoted by A) come from layer $k+1$. Before running the CMT protocol for C, the verifier holds the commitment (either obtained from the preprocessing or received from the server) to the multilinear extension, $\tilde{V}_k^B$, of the fresh inputs to the kth layer. Next, during the execution of the CMT protocol, the client receives the evaluation of the multilinear extension of the values at layer k, i.e., $\tilde{V}_k(r_1, \ldots, r_{m+1})$, at some random point $(r_1, \ldots, r_{m+1})$ as before. As only the first M wires (corresponding to A) are connected to layer $k+1$, the client needs to obtain the evaluation of the multilinear extension (denoted by $\tilde{V}_k^A$) of the first M values, at a random point and use it to continue the CMT protocol for layer $k+1$.

This is done as follows. By Equation 2.1 in Section 2.2, we have $\tilde{V}_k(r_1, \ldots, r_{m+1}) = (1-r_1)\tilde{V}_k^A(r_2, \ldots, r_{m+1}) + r_1 \cdot \tilde{V}_k^B(r_2, \ldots, r_{m+1})$. Since B are all input gates, the client can request the evaluation of $\tilde{V}_k^B$ at point $(r_2, \ldots, r_{m+1})$ along with a corresponding proof (using the verifiable polynomial-delegation protocol). Next, the client computes $\tilde{V}_k^A(r_2, \ldots, r_{m+1}) = \left(\tilde{V}_k(r_1, \ldots, r_{m+1}) - r_1 \cdot \tilde{V}_k^B(r_2, \ldots, r_{m+1})\right)/(1-r_1)$, obtaining an evaluation of $\tilde{V}_k^A$ at the random point $(r_2, \ldots, r_{m+1})$. The client then uses it to continue the execution of the CMT protocol for layer $k+1$ as usual.

We note that similar optimizations can be performed in order to avoid relying output gates as well.

Generalizations. For both inputs and outputs, using Equation 2.1 allows us to avoid relaying a number of input and the output gates. We notice that the number of input (resp. output) gates does not have to be half of the total number of gates in the layer, but can be any fraction $1/m'$ such that m' is a power of 2. Moreover, while we described the solution assuming that the "fresh" inputs at some layer are all in the second half of the inputs to that layer, this is not required. With small modifications we can accommodate more complicated wiring patterns, e.g., the case where odd wires are routed from the previous layer and even wires are fresh inputs to the circuit.

4.5.3 Verifying Set Intersections

A join operation requires computing the intersection of two large sets of column values (assuming for now there are no duplicates). The naive way to compute the intersection of two N-element sets, where each element is represents using z bits, requires a circuit that performs N^2 equality tests on z-bit inputs. We describe here several ways this can be improved.

A sorting-based $O(zN \log^2 N)$ solution. An asymptotic improvement can be obtained by first sorting the $2N$ elements, and then comparing consecutive elements in the sorted result. Sorting can be done using $O(N \log^2 N)$ comparator gadgets of width z, resulting in a circuit of size $O(zN \log^2 N)$ overall. The concrete overhead of this approach is high, as each comparator must be implemented by decomposing the inputs to their bit-level representations.

A routing-based $O(zN + N \log N)$ solution. Prior literature on SNARKs [16] improves the above by relying on auxiliary input from the prover to replace sorting networks with switching networks that can induce arbitrary permutations on N elements. Using this approach, the server will simply specify the permutation that sorts the elements; the client can verify that the elements are sorted in linear time. Switching networks can be built using $O(N \log N)$ gadgets that swap their inputs if an auxiliary bit is set to 1. The total complexity of this approach is $O(zN+N \log N)$.

An $O(zN)$ interactive solution. In our setting, where we have interaction, we can do better. We simply have the server provide the sorted list $x'_1, \ldots, x'_{2N}$ corresponding to the original items $x_1, \ldots, x_{2N}$. The client can verify that the new list is sorted in $O(N)$ time, so all that remains is for the client to verify that it is a permuted version of the original list. This can be done by having the server commit to the new values (as part of the auxiliary input he computes) using our verifiable polynomial-delegation scheme. The client then chooses and sends to the server a uniform value r, and both parties then run an interactive proof protocol to verify that $\prod_{i=1}^{N}(x_i - r) - \prod_{i=1}^{N}(x'_i - r) = 0$. Overall, this approach requires $O(zN)$ auxiliary inputs and gates.

Sorting 0 values. The concrete cost can be further reduced as follows. In case many of the elements are 0, after the sorting step they will be pushed to the front of the auxiliary-input array (assuming, for simplicity, that all values are non-negative). Instead of providing one auxiliary input per element, it suffices for the prover to tell the verifier the number of non-zero elements, and only provide aux-

iliary inputs for those. For example, assume only the last $1/m$ of elements are non-zero (where m is a power of 2), using Equation 2.1 in Section 2.2, the evaluation of the multilinear extension for all elements at point $r = (r_1, \ldots, r_{\log(mn)})$ is $\tilde{V}(r) = r_1 \ldots r_{\log m} \tilde{V}_m(r_{\log m+1}, \ldots, r_{\log mn})$, where $\tilde{V}_m$ is the multilinear extension of the non-zero elements. Thus, the size of the auxiliary input and the number of necessary comparisons only depend on the number of non-zero elements (as opposed to the total number of elements).

In the context of SQL queries, the scenario above is very common. Consider a query where a join clause is applied on the result of two range queries. It is often the case that only a small portion of rows in the table fall within the bounds imposed by the latter. Therefore, after evaluating the range selection, the values in these rows will be propagated through the circuit, while the values in all other rows will effectively be set to 0. The join query (and therefore the sorting) will then be applied on this result which has the property that many of its elements are 0. Thus the above optimization can significantly lower the join evaluation cost in this case.

Sorting multiple columns. Another challenge arises when the output of a join query includes more than just the reference column, e.g., `SELECT * FROM` $\mathbf{T}_1$,$\mathbf{T}_2$, `WHERE` $\mathbf{T}_1.col_i = \mathbf{T}_2.col_j$. In this case, in order to compute the set intersection using the above interactive method, the verifier must make sure that the prover permuted all of the columns of $\mathbf{T}_1$ (resp., $\mathbf{T}_2$) with the same permutation used for col_i (resp., col_j).

We achieve this using the following *packing* technique. Assume for simplicity

that each database row has two columns with values x_i, y_i respectively, and that the elements are arranged as tuples $(x_1, y_1), \cdots, (x_N, y_N)$. Suppose the elements x_i, y_i have length at most z bits, with $z < \lfloor \log p \rfloor / 2$. To sort both columns based on the x_i values, we ask the server to provide auxiliary inputs $(a_1, \ldots, a_{2N}) = (x_{\pi(1)}, y_{\pi(1)}, \cdots, x_{\pi(N)}, y_{\pi(N)})$, such that the $\{x_{\pi(i)}\}$ are sorted and the $\{y_{\pi(i)}\}$ are permuted by the same permutation. The client then chooses and sends to the server two random values r_1, r_2, and both parties run the interactive proof protocol described above for the following three checks:

1. $\prod_{i=1}^{N}(x_i - r_1)(y_i - r_1) - \prod_{i=1}^{2N}(a_i - r_1) = 0$;

2. $\prod_{i=1}^{N}(b_i - r_2) - \prod_{i=1}^{N}(b'_i - r_2) = 0$, where $b_i = x_i + y_i 2^z$ and $b'_i = a_{2i-1} + a_{2i} 2^z$;

3. $(a_1, a_3, \ldots, a_{2N-1})$ are sorted.

The first check guarantees that a_is are a permutation of x_i, y_is, which also implies that a_is have length at most z bits. Now as x_i, y_i, a_is all have length at most z bits, the second check guarantees that $\exists \pi : a_{2i-1} = x_{\pi(i)}$ and $a_{2i} = y_{\pi(i)}$ (note that we cannot omit the first check as there exist a_is with more than z bits that can pass the second check). This, together with the last check, guarantees $x_{\pi(i)}$s are sorted and $y_{\pi(i)}$s are permuted by the same permutation.

The technique generalizes naturally to sort multiple columns based on a reference column. As long as the packing result does not overflow in $\mathbb{F}_p$, we can pack all the columns. Otherwise, we can duplicate the reference column, perform a separate packing of subsets of columns, and sort them separately. In particular, assuming $z = 32$ and p is 254 bits long, we can pack up to 7 columns in a single field element.

Handling duplicate values. Finally, if there are duplicate values in the reference columns, the result of a join query can no longer be described as a set intersection. In this case, a pairwise comparison of the elements of the two columns, viewed as multisets, provides the correct result but the cost is quadratic in the number of database rows. Instead we can do the following. First, we extract the unique values from each multiset (using a linear-size circuit as described in [78]). Then we compute the intersection of the resulting sets with our previous technique for the case of no duplicates. Following this, we apply again the same technique to intersect this intersection with each of the original multisets. This returns two multisets such that: (i) each of them contains exactly those elements that appear in both original multisets, and (ii) every element appears in each multiset exactly the same amount of times as it appeared in the the corresponding original multiset. Finally, the join result can be computed with a pair-wise comparison of the elements of these two multisets. Note that the cost for this final step is asymptotically optimal as it is exactly the same as simply parsing the join's output.

4.5.4 Supporting Expressive Updates

A common problem of existing dynamic authenticated data structures (e.g., [67, 91]) is that they support limited types of updates: element insertions and deletions. Thus, they cannot handle general updates that can be expressed as SQL queries themselves, e.g., the query `UPDATE` **Employees**; `SET` **Salary** = 45000; `WHERE` **Age** = 33.

The main reason such update queries are hard to handle is that the client must eventually compute the corresponding updated database commitment. Without access to the database, it must again rely on the untrusted server to provide this new commitment. SNARK-based constructions can support expressive updates by including the commitment computation in the circuit. However, this would considerably increase the prover's overhead.

Our approach avoids this cost by separating the computation of the update from its verification. First, the server computes the updated database normally, and commits to the multilinear extension of the result using our verifiable polynomial delegation scheme. The client and server then verify that the update was performed correctly by running the CMT protocol on the circuit that performs the update. In order to initiate the CMT protocol, the client needs to compute the multilinear extension of the updated database (which is here the circuit's output) and evaluate it on a random point. This would naively require transmitting the entire updated database back to the client. Instead, we rely on the server to compute the evaluation for the client, and verify this value using our verifiable polynomial-delegation scheme. Once this is done, the remainder of the CMT evaluation proceeds normally.

4.5.5 Efficient Value Insertions

As explained above, our construction can handle any update query by having the server evaluate the update-query circuit and then commit to the output as the new digest. For simple updates such as adding/subtracting a constant from an

element, we have a much simpler mechanism. By utilizing the closed form of the multilinear extension, in order to add a constant v to the bth entry in the database, the multilinear extension of the database is increased by $\mathcal{X}_b(x_1, \cdots, x_n)v$ (as defined in Equation 2.1). Therefore, the client only needs to multiply the commitment of the database by $g^{\mathcal{X}_b(x_1,\cdots,x_n)v} = p_b^v$, where p_b is the bth element of the public key $\mathbb{P}$. In practice, as the size of $\mathbb{P}$ is linear in the size of the database, the client can outsource its storage to the server and obtain an authenticated value of p_b using a Merkle hash tree or digital signatures. Thus, simple updates of this form can be handled with one round of interaction, and the running time for both parties is logarithmic in the database size using a Merkle tree, or constant using a digital signature scheme. The update above also captures inserting a new element/row to the database, which is adding their values to previously unused cells.

4.6 Experimental Results

4.6.1 Experimental Setup

We implemented our constructions (including the circuit generator and CMT protocol) in C++, and compiled it with g++ 4.8.4. We use the NTL library [7] for number-theoretic operations, and SHA-256 from the OpenSS libraryL [8] to instantiate a random oracle. For the bilinear pairing we use the ate-paring library [1] on a 254-bit elliptic curve. The EMP toolkit [86] was used for the network I/O between the server and the client.

Since the running time of our verifiable polynomial-delegation protocol is over-

whelmingly dominated by the modular exponentiations, in what follows we estimate the running time of this component of our system by simply performing the same number of exponentiation operations (using the same setup as above).

Hardware and network. Our experiments were executed on two Amazon EC2 c4.8xlarge machines running Linux Ubuntu 14.04, with 60GB of RAM and Intel Xeon E5-2666v3 CPUs with 36 virtual cores running at 2.9 GHz. For the WAN experiments, we used machines hosted in two different regions, one in the US East and the other in the US West. The average network delay was measured at 72ms and the bandwidth was 9MB/s. For each data point, we collected 10 experimental results and report their average.

Database setup. We evaluate performance using the TPC-H benchmark [9], which contains 8 synthetic tables and 22 SQL queries and is widely used by the database community for performance evaluation. We represented decimal numbers, dates, and categorical strings in the tables as elements in the field used by our constructions. In our experimental evaluations, we do not consider substring or wildcard queries, and the corresponding columns were discarded. The TPC-H database contains tables of various sizes. The two largest tables used in our experiments contained 6 million rows and 13 columns and 0.8 million rows and 4 columns, respectively.

TPC-H queries. We tested five TPC-H queries: query #2, #5, #6, #15, and #19. As a representative example, query #5 is shown in Figure 4.2. It gives an example of multi-way join queries on different columns of different tables. sub-query in line 6 is a selection query on table region, and the query in lines 7–8 is a range query on

```
1. SELECT n_name, SUM(l_extendedprice*(1-l_discount))
2. AS revenue
3. FROM customer, orders, lineitem, supplier, nation, region
4. WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
5.     AND l_suppkey = s_suppkey   AND c_nationkey = n_nationkey
6.     AND n_regionkey = r_regionkey AND r_name = 'MIDDLE EAST'
7.     AND o_orderdate >= date '1997-01-01'
8.     AND o_orderdate < date '1997-01-01'+interval '1' year
9. GROUP BY n_name
10. ORDER BY (revenue) DESC;
```

Figure 4.2: Query #5 of the TPC-H benchmark.

table order. Lines 4–6 consist of join queries among tables customer, order, lineitem, supplier, nation, region. In line 1, the result is projected to three columns, two of which are aggregated. Finally, in lines 9–10, the aggregated values are summed for each unique value of n_name, and sorted based on n_name in descending order.

Query #2 is a nested query. The inner query consists of a 4-way join followed by a MIN query, resulting in a single value. The outer query consists of selection queries, where the result of the inner query is used as a constraint, followed by a 4-way join and projections. Query #6 is a simple 3-dimensional range query followed by an aggregation. Query #19 consists of range and selection queries on two tables, followed by a single join query and an aggregation. Query #15 creates a new table that is the result of a one-dimensional range query and a SUM query. All the other

queries in TPC-H are variants of these five queries with different dimensions and constraints.

Query representation and field sizes. For every TPC-H query we implemented a circuit generator that takes as input the database size and outputs an arithmetic circuit for evaluating the specified query on a database of that size, using the optimizations described in Section 4.5 (when possible). We implemented both the CMT protocol (Construction 1) and the verifiable polynomial-delegation protocol (Construction 2) using a prime-order field with a 254-bit prime.

4.6.2 Performance Comparison: Selection Queries

We compare the performance of our construction with prior work, including IntegriDB [91], a special purpose system optimized for a class of SQL queries, and libsnark [6], the state-of-the-art general-purpose SNARK implementation. Below, we report the results on queries #2, #5, #6, and #19.

For IntegriDB, we downloaded the implementation from [4] and executed it on our machine. For libsnark, we estimated the performance as follows. For each query, we first produced its circuit representation the jSNARK compiler [5], hardcodeding the TPC-H dataset in the circuit. This resulted in a circuit which takes as inputs the values used in selection and range queries. We then constructed a SNARK using libsnark for this circuit, and report its performance. We note that this approach of hardcoding the database and the query into the circuit yields a preprocessing phase whose results are only useful for that specific query and database. In particular, the

results of the preprocessing phase *cannot* be reused for other queries or databases, or even an updated version of the database. Although clearly unrealistic, this approach gives a lower bound on the server time when using a SNARK-based approach.[3] Even with this more efficient approach, we were not able to generate SNARKs for circuits containing more than 2^{20} multiplication gates (see Table 4.7 for the circuit sizes of the queries we used in our evaluation). Therefore, for experiments requiring larger circuits, we estimated the cost assuming the prover time grows linearly in the circuit size (this is, again, an underestimate since the prover time actually grows quasilinearly in the circuit size).

Setup phase. The setup phases in both IntegriDB and vSQL are query independent and thus need to be executed only once, after which any supported queries can be handled. We run the setup phases of both IntegriDB and vSQL on all eight TPC-H tables. The setup for vSQL took about 1,185 seconds. For IntegriDB, the setup phase could not be completed on the entire TPC-H database due to excessive memory consumption. Our estimate for the setup phase of IntegriDB was about 350,000 seconds. Our construction is about 295× faster than IntegriDB because the complexity of our setup phase is linear in the number of columns compared to quadratic in IntegriDB.

For libsnark, the setup time depends on the query. The fastest setup time, for query #6, is estimated to take 36,000 seconds, which is an order-of-magnitude

[3]It is possible to use SNARKs that support arbitrary queries by constructing a SNARK for a universal circuit [18] and supporting delegation of storage [29, 41]. However, these approaches introduce additional overhead.

	IntegriDB		**SNARKs**		**vSQL** (ours)			
Query	Server	Client	Server*	Client*	Server	Client	Total (WAN)	Total (NI)
#19	6,376s	232ms	196,000s	6ms	4,923s	148ms	4,989s	4,923s
#6	1,818s	74ms	19,000s	6ms	3,878s	112ms	3,896s	3,878s
#5	✗	✗	615,000s	110ms	5,172s	305ms	5,379s	5,172s
#2	✗	✗	58,000s	40ms	2,421s	427ms	2,633s	2,421s

Figure 4.3: Comparison of server and client times for evaluating queries using IntegriDB, a SNARK-based approach and our construction. (See text for details.) The numbers in columns marked by * are estimated. ✗ denotes an unsupported query.

.

slower than vSQL. Running setup for all four queries is estimated to require about $1.7 \cdot 10^7$ seconds (roughly 197 days).

Evaluation phase. The results of the evaluation phase are summarized in Table 4.3. The numbers reported in the "Server" column reflect the computation time required for the server to evaluate the SQL query and produce a valid proof; those in the "Client" column reflect the time for the client to verify the proof. For vSQL, in the "Total (WAN)" column we also report the total end-to-end time which includes the overhead due to communication between the client and server over a WAN network. For comparison, the total time for IntegriDB and SNARKs (which are non-interactive) is essentially the same as the server time, since the client time is negligible. Note, however, that vSQL can be made non-interactive in the random oracle model, virtually eliminating the cost of interaction at the negligible expense of a small number of SHA-256 computations. We report the performance of

this non-interactive mode, including the SHA-256 computation time, under "Total (NI)."

Evaluation phase: vSQL vs. IntegriDB. As shown in Table 4.3, IntegriDB can only support queries #19 and #6, compared with vSQL which can support all TPC-H queries. While being more expressive, the running times of vSQL's client and server are on the same order of magnitude as those of IntegriDB; in fact, for query #19, vSQL's server (resp., client) outperforms that of IntegriDB by about 23% (resp., 36%). We observe also that the cost of interaction for vSQL (even over a WAN) is small, mainly because prover time is by far the dominating cost.

Evaluation phase: vSQL vs. SNARKs. Compared to libsnark, the server time of our constructions is significantly faster (ranging from approximately 5× for query #6 to approximately 120× for query #5). At a high level, the better performance of vSQL is a consequence of two features. First, our construction is mostly information-theoretic and the number of (relatively slow) cryptographic operations it requires is linear in the *input and output length*, whereas SNARK-based approaches require a number of cryptographic operations linear in the *circuit size*. In addition, as described in Section 4.5, our construction leverages interaction and auxiliary input to reduce the size of a query's circuit representation. Verification when using a SNARK-based approach is faster than in vSQL since it requires only a number of cryptographic operations linear in the output length. In practice, however, the difference is at most 0.5sec which we consider negligible for most applications. We stress that all numbers reported for libsnark are underestimations since they

assume the database and queries are fixed in advance. We expect our construction's improvement to be even more significant compared to more general SNARK-based systems that support arbitrary queries and dynamic outsourced databases (see discussion below).

Communication. For libsnark-based systems, the additional communication required for the proof is always constant (e.g., 288 bytes). For IntegriDB and vSQL the communication required in all experiments was under half a megabyte. We consider this to be negligible in practice for modern networks and thus omit additional details.

Comparison with other SNARK-based systems. SNARKs can be used for verifiable computation in various ways other than the one we used for our comparison.

Exploiting Structured Computations via Bootstrapping. Geppetto [37] takes a complex computation and splits it into smaller building blocks, each represented as a "small" circuit. Each such circuit can then be pre-processed with a SNARK separately. In the context of SQL queries, the natural way to split the computation is by having one small circuit that operates on a single row, and then applying that circuit iteratively to each row in the database. An additional SNARK is then needed to aggregate and verify the outputs of all the smaller SNARKs into a single succinct proof, in a "bootstrapping" step. In practice, Geppetto has the potential to significantly reduce the preprocessing time and memory consumption since the same small circuit is used throughout the query evaluation. However, the total prover

time to execute the smaller SNARK on every row is similar to that of the single large SNARK we used as our benchmark, as the total number of exponentiations is linear in the total number of multiplication gates and breaking a large circuit into multiple smaller ones does not reduce that. Our results already show that the prover time in that case is up to 2 orders of magnitude slower than for vSQL. Additionally, the bootstrapping phase requires approximately 30,000–100,000 gates per instance of the smaller circuit [37, Section 7.3.1]. Applying this to a table with 6 million rows (as in the TPC-H dataset) will thus introduce an additional overhead of 1.4–4.8×10^7s for the prover, based on our estimations with libsnark (which is itself an underestimate as the bootstrapping phase operates over a larger and less efficient elliptic curve).

Memory Delegation via Hash Functions. Pantry [29] can be used to outsource memory by implementing a Merkle hash tree on top of Pinocchio [71]. The consistency of each memory read/write access must be proven by checking the corresponding Merkle path as part of the SNARK that evaluates the query. This approach has the benefit of allowing the verifiable evaluation of RAM programs on outsourced memory (as opposed to expressing the computation directly as a circuit). For SQL queries, assuming the existence of pre-built database indices (as is typically the case with modern database management systems), there are programs that can evaluate certain queries in time sublinear in the database size. (E.g., assuming the existence of a search tree that stores the ordered element values at its connected leaves, a simple 1-D range query can be evaluated in time logarithmic in the database size and

linear in the result.) Thus, for specific queries for which such indices can be built, Pantry can in theory outperform vSQL. Regarding the specific queries we evaluated here, we note that the number of memory accesses would still be very large even with the help of pre-built indices. The simplest TPC-H query we tested is query #6, which is a 3-dimensional range query followed by a summation. Assuming a search tree is built for each dimension and each 1-D range query has a 1% selectivity (which is well below the selectivity in our experiments), getting the result of each dimension requires 60,000 memory accesses. In practice, the concrete cost of proving the correctness of each memory access would be approximately 2.5s, using a SNARK-friendly algebraic hash function [60] for 10^6 4-byte memory blocks. Therefore, just verifying the memory accesses for query #6 would take around 450,000s (5 days) in this case.

Finally, in contrast to vSQL, both approaches require a query-specific setup phase that can only be avoided if one uses a universal circuit [18] or proof-bootstrapping [17], but these techniques incur considerable additional overheads.

4.6.3 Performance Comparison: Update Queries

We further test the performance of vSQL on a `CREATE` query (query #15 in TPC-H). As shown in Table 4.4, the communication required is only 86.4KB, even though the newly created table is itself more than 1MB in size. IntegriDB cannot directly support such expressive updates. The only solution for IntegriDB would be for the client to download the entire new table, verify its correctness, and preprocess

Server	Client	Total (WAN)	Total (NI)	Comm.
2,034.3s	66ms	2,089.4s	2,034.4s	86.4KB

Figure 4.4: Performance of our construction on TPC-H query #15, creating a new table from table lineitem.

it from scratch.

Next we look at updates that can be supported by IntegriDB, in particular inserting a new row. In this case, vSQL outperforms IntegriDB since the total time for inserting a row into the lineitem table in TPC-H is only 5.2ms using vSQL vs. 1.46s using IntegriDB. This is because the vSQL client only needs to verify the corresponding elements of the public parameters using a Merkle-tree proof and then perform one exponentiation per column. For IntegriDB, the required number of exponentiations is quadratic in the number of columns and logarithmic in the number of rows.

In order for a SNARK-based system to support updates, it must offer a way to check the validity of a new database digest returned by the prover. The standard way of doing this is by incorporating the digest computation in the circuit that is evaluated, which introduces a huge cost in practice. More recent approaches can achieve this via an external mechanism that is not part of the circuit of the SNARK (e.g., [12, 41]). Note however, that at the very least the update circuit must be evaluated and the SNARK proof must be computed by the prover. According to our performance comparison in the previous section, this already takes more time than vSQL.

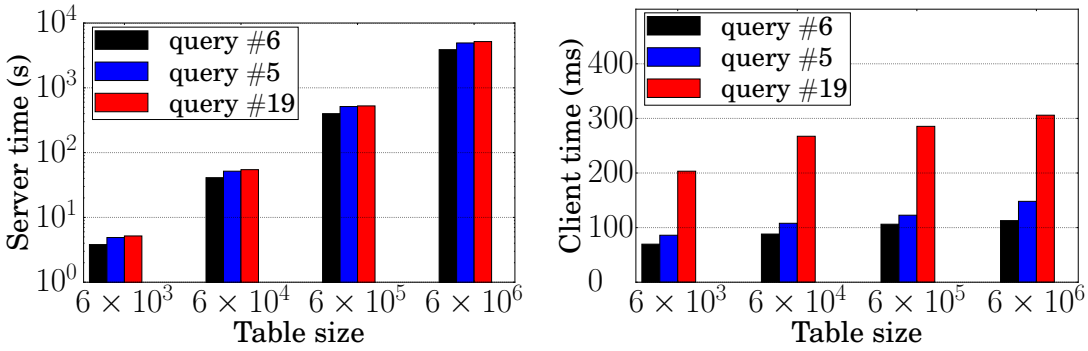

Figure 4.5: Performance of vSQL for three TPC-H queries as a function of the number of rows in the largest table involved in the query.

4.6.4 Scalability of Our Construction

In this section, we evaluate the performance of our constructions as a function of the database size. To that end, we run our construction on the largest three of the previous queries and scale the number of rows in the largest participating table from 6,000 to 6,000,000.

Server time. As shown in Figure 4.5, the server performance for query evaluation scales almost linearly with the size of the largest table, matching the theoretical analysis of Theorem 5.

Client time. Figure 4.5 shows that the client's verification time grows logarithmically with the number of rows in the largest table participating in a query (note the logarithmic scale of the horizontal axis). This again matches Theorem 5.

Query	# of Inputs	Time (sec)	Time/Input (ms)
#15	12,002,430	1,389	0.1157
#2	17,840,340	2,065	0.1158
#6	24,004,860	2,741	0.1142
#5	31,397,075	3,612	0.1151
#19	32,406,075	3,726	0.1150

Figure 4.6: Prover time for our polynomial-delegation scheme. The number of inputs includes both the database and the prover's auxiliary inputs.

4.6.5 Microbenchmarks

In addition to evaluating vSQL's end-to-end performance, we also report on the performance of vSQL's main components—namely, our implementations of the CMT protocol and the polynomial-delegation scheme.

Performance of the polynomial-delegation scheme. Table 4.6 shows the prover time the polynomial-delegation scheme (Construction 2). As can be seen, the prover spends about 0.11ms per input, which is the same order of magnitude as what is achieved with SNARK-based schemes. Preprocessing for our polynomial-delegation scheme (which is the only part of our construction that requires preprocessing) took only 1,185 seconds for 95,525,880 inputs (12.4μs per gate).

Performance of the CMT protocol. Table 4.7 shows the performance of our implementation of the CMT protocol. As can be seen, the average time required

Query	# of Gates	Time (sec)	Time/Gate (μs)
#2	198,646,335	356	1.79
#15	367,495,719	646	1.76
#6	704,643,060	1,137	1.61
#19	801,374,196	1,198	1.49
#5	945,828,996	1,560	1.65

Figure 4.7: Prover time for our implementation of the CMT protocol.

per gate is about 1.7μs.

Chapter 5: Applications: Verifiable RAM Programs

While circuits can model arbitrary programs, most real-world computations are expressed in terms of random-access memory (RAM) machines. This is true both in terms of most programmers' mental model of computation, as well as in terms of the execution of assembly code on general-purpose computers. However, since most constructions of VC protocols, including our new protocol in Section 3, work on computations expressed as arithmetic circuits, verification of a RAM program P is usually done by verifying the correct evaluation of an arithmetic circuit C_P that validates the execution of the program P. As mentioned in related work in Section 1.1, most VC protocols (e.g., SNARKs) require the circuit to be fixed ahead of time, during a trusted preprocessing phase. Due to this, previous works for verifying RAM programs can be roughly divided into two main categories.

1. **Program-specific preprocessing.** If the program P to be verified is known ahead of time, it is possible to tailor the circuit C_P so as to verify P as efficiently as possible. While this tailoring is beneficial to the protocol's overall performance, it comes at the expense of usability since C_P cannot be used to verify another program P'. Examples of this approach are Pantry [29] and Buffet [83].

2. **Universal preprocessing.** In case the RAM program to be verified is not known ahead of time, it is possible to construct a universal circuit C_{RAM} which is capable of verifying any RAM program that runs for at most T steps. Examples of this approach include [16, 18].

Both these approaches have significant drawbacks. In the first case, the verifier cannot change the RAM program P being verified without re-running the (expensive) preprocessing phase. This is a major drawback as the preprocessing cost can only be amortized by running the same program on different inputs.

In the second case, although the preprocessing cost can be amortized over the evaluation of different programs on different inputs, the universal preprocessing used in this approach imposes large concrete overheads during the proving phase.

This results from the fact that C_{RAM} must be able to emulate all possible operations at every CPU step in order to handle arbitrary RAM programs. In contrast, the program-specific approach benefits from the fact that P is known when C_P is chosen, and so the set of possible instructions at each step is potentially much smaller.

Two notable exceptions to the above are the works of [13, 17], which do not need a preprocessing phase tied to a specific circuit. However, the concrete cost of these systems remains significantly higher than that of the preprocessing-based solutions mentioned above. (See Section 5.3.2.)

In this chapter, we present a new verifiable RAM construction, vRAM, that achieves the best of both categories. It has similar (or even better) performance

than what is achievable with program-specific preprocessing, but without knowing the program during the preprocessing phase. This is achieved by using our new argument system on the circuit validating the execution of a RAM program as a backend, which has faster prover time and does not require the circuit to be fixed during the preprocessing phase.

Experimental evaluations show that vRAM improvse the prover's running time by 9–30× as compared to the state-of-the-art-implementation in the universal setting [18]. On the other hand, compared to systems using program-specific preprocessing [83], vRAM achieves very similar prover performance; in fact, in some cases our prover is faster despite the fact that systems with program-specific preprocessing can deploy program-specific optimizations during the preprocessing phase.

We also show that vRAM is much better in terms of memory consumption, which is currently the main bottleneck for running large instances of verifiable computation. vRAM achieves an improvement of 55–110× in terms of memory consumption compared to [18], which allows us to prove computations involving more than 2 million CPU cycles with 256GB memory (65× more than [18]). The improvements achieved by vRAM come at the cost of increased verifier's running time and proof size, however these still remain well within the capabilities of modern machines.

In this chapter, we first give preliminaries on RAM programs and RAM to circuit reductions in Section 5.1, and present our new reduction in Section 5.2. We then show experimental results in Section 5.3.

5.1 Preliminaries on RAM programs

5.1.1 A Canonical RAM Architecture

In this section we establish notation for a random-access machine supporting some instruction-set architecture.

Hardware. We focus on RAM machine computations, where the machine is parametrized by the number of registers K and the register width (word size) W. The CPU state consists of a W-bit program counter (pc) and K general-purpose, W-bit registers $r_1, \ldots, r_K$. Each instruction operates over two operands (registers) and stores its result in a third register, to which we shall refer as the destination register. The machine's memory is a randomly accessible array of 2^W bytes. We also assume two read-only unidirectional tapes containing W-bit words. The first tape is used for the program input x, and the second tape may potentially be used for auxiliary input aux.

Program execution. A program is a sequence of instructions, where each instruction has two operands (which are either register numbers or constants) and stores its result in a third register called the destination register. A random-access machine starts executing a program with all registers, its memory, and the program counter initialized to 0. At each step, the instruction pointed by the pc is executed. By default, every instruction increments the pc by one (i.e., pointing to the next instruction), but an instruction (e.g., jump) can also modify the pc directly to facilitate arbitrary control flow. The machine's inputs are the above-mentioned tapes,

S and S^{**}	t, pc, $r_1, \ldots, r_K$, O, flag, auxiliary[1]
I and I^{**}	line number, opcode, i, j (source registers), k (target register)
A	a, t, O, b (denoting memory load or store)

Table 5.1: Values in a state and an instruction.

accessible via special read instructions, as well as the initial contents of its memory. The machine outputs either accept or reject. We say program P accepts input (x, aux) if the machine running program P with the specified input terminates with output accept.

Machine state and instruction encoding. We define the notion of machine *state* as the values of the machine's registers $pc, r_1, \cdots, r_K$ at any point during the program execution. Let $S_1, \cdots, S_T$ be a list of the machines states during the execution of some program P. We augment each state S_i to also include i in it, referring to i as S_i's *step number* as well as to include an additional field O_i, referring to it as the instruction's *output field*. An *instruction* I contains information about what operation the machine should execute (e.g., addition, multiplication, etc.), the two source registers r_i, r_j as well as the target register r_k. For a specific program (which is a sequence of instructions) $P = P_1, \cdots, P_\ell$, we augment every instruction P_i to include its location i (line number) within P. The detailed values in a state and an instruction used in our implementation is shown in Table 5.1.

[1]Auxiliary includes data from the prover for efficient implementation purposes, i.e., bit-decomposition of the values for computation modulo 2^{32} and bits denoting whether an instruction is jump, memory store or load.

We take as our set of available instructions from those used by TinyRAM [16, 18]. This is an ideal starting point for our implementation as the universal circuit for the TinyRAM CPU can be described by a relatively small arithmetic circuit.

Execution traces. The trace $tr = (S_1, I_1, S_2, I_2, \ldots, I_{T-1}, S_T)$ of a program P on inputs x, aux is a sequence of CPU states and instructions, where S_1 is the initial state and each S_i is produced by executing instruction I_{i-1} on S_{i-1}. A trace tr is *valid* for a program P on input x if there is an aux such that $P(x, \mathsf{aux})$ has trace tr. Similarly, a trace tr of a program P on input x is accepting if there exists aux such that tr is valid and we say that P accepts input (x, aux).

A universal NP relation for RAM programs. The following NP relation $RAM_{\ell,n,T}$ captures accepting RAM programs:

Definition 5. *For $\ell, n, T \in \mathbb{N}$, relation $RAM_{\ell,n,T}$ consists of tuples $(P, x; \mathsf{aux})$ such that: (i) P is a program with $\leq \ell$ instructions, (ii) x is an input of $\leq n$ words, and (iii) $P(x, \mathsf{aux})$ accepts in $\leq T$ steps.*

5.1.2 Previous Reductions from RAM to Circuit Satisfiability

Before describing our improvements, in this section we present previous approaches for constructing a circuit that can verify the execution of RAM programs. More specifically, given a time bound T, [18] constructs a circuit C such that for any RAM program P, $\exists w : C(P, x; w) = 1$ if and only if $\exists \mathsf{aux}$ such that $P(x; \mathsf{aux})$ accepts. Throughout this paper, unless otherwise noted, we do not distinguish between the program and the input data, and we let ℓ be a bound on both the

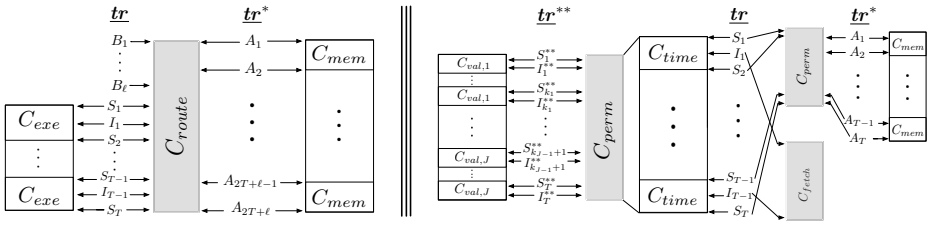

Figure 5.1: Circuits for the reductions from RAM programs to circuits from Section 5.1.2 (left) and Section 5.2 (right). Circuits C_{fetch} and C_{perm} receive additional input from the verifier as described in Sections 5.2.2 and 5.2.4, respectively.

program length and the input size.

The circuit C takes as input a program P and a witness w that contains a trace $tr = (S_1, I_1, S_2, I_2 \cdots, I_{T-1}, S_T)$ and aux. C then outputs 1 only if S_1 is the initial state, S_T is an accepting state, and the following hold at every step i in tr:

1. **Correct instruction execution.** State S_{i+1} is obtained from S_i after executing instruction I_i.

2. **Correct instruction fetches.** I_i is the instruction in P pointed to by the program counter (pc) in S_i. If $i = 1$ we require that $pc = 0$.

3. **Correct memory accesses.** If I_i is a load instruction accessing address a then the value loaded is v, where v is the last value written to address a by some previous instruction (and $v = 0$ if I_i is the first load from a.)

In order to verify the above three conditions, the circuit C is constructed from three sub-circuits C_{exe}, C_{mem}, and C_{route} (cf. Figure 5.1(left)), which we explain below.

Ensuring correct instruction execution. To ensure (1), every triple S_i, I_i, S_{i+1} is given as input to a circuit C_{exe} which performs the following two checks. (a) Check that the value O_i in S_i is correctly computed by executing I_i.[2] In case I_i is a memory load instruction, C_{exe} optimistically assumes that the loaded value O_i is correct (this will be tested separately when checking memory accesses). (b) Check that O_i is equal to r_j of S_{i+1} (or pc_{i+1} in case of jump), all other registers of S_{i+1} are the same as S_i, S_i's step number is indeed i and pc_i is equal to the line number of I_i in P (as encoded in I_i).

Ensuring correct instruction fetches. To ensure (2), C must check that the instruction I_i is fetched from the location in the program P pointed by pc_i in state S_i (i.e., that I_i is the pc_i-th instruction in P). In [18], this is achieved by storing P in memory and then loading instructions before they are executed. Formally, a booting sequence $B_1, \ldots, B_\ell$ is prepended to the trace tr, with B_i storing the i-th instruction of P in memory at address i. This results in a new trace $tr = (B_1, \cdots, B_\ell, S_1, I_1, S_2, I_2 \cdots, I_{T-1}, S_T)$ of length $2T + \ell$. Each $I_i \in tr$ is then viewed as two operations: One is a load operation fetching an instruction from the memory address pointed by its line number, and the other is I_i itself. In this way, the correctness of instruction fetches is reduced to checking the consistency of the memory stores and loads performed by Bs and Is, which we describe next.

Ensuring correct memory accesses. To ensure (3), Ben-Sasson et al. [18] include in w an additional trace $tr^* = (A_1, \cdots, A_{2T+\ell})$, which is a permuted version of tr

[2] If I_i is a memory instruction, O_i is the loaded or stored value; if I_i is a jump instruction, O_i is the jump destination.

where: (a) all the states in which a memory access is performed are sorted by the memory address a being accessed (with ties broken by their step number in tr), and (2) non-memory instructions are pushed to the end of tr. Notice that B_i and I_i are also sorted, using the addresses i and the line number respectively. For two adjacent entries $A_i, A_{i+1} \in tr^*$ with outputs O_i, O_{i+1}, step numbers t_i, t_{i+1} and accessing addresses a_i, a_{i+1}, respectively, the circuit C_{mem} checks the following:[3]

- If $a_i = a_{i+1}$ then $t_i < t_{i+1}$. If A_{i+1} is a load instruction, the loaded value O_{i+1} is the same as the value O_i stored or loaded by A_i.
- If $a_i \neq a_{i+1}$ then $a_{i+1} > a_i$, and if A_{i+1} is a load instruction then $O_{i+1} = 0$.

Checking consistency between tr and tr^*. Finally, C must ensure that tr^* is a copy of tr that contains exactly the same states and instructions, just sorted by their accessed addresses. Note that the fact that tr^* is sorted correctly has already been checked by C_{mem}. Hence, it remains to ensure that a state appears in tr^* if and only if it appears in tr. This can be done by checking that there exists a permutation π such that $\pi(tr^*) = tr$. To that end, C contains a sub-circuit C_{route} which implements a $O(T \log T)$ switching network that routes every entry in tr to its matching entry in tr^*. The control bits used for the switching network (which specifies the permutation π) are provided by the prover and included in w.

Overall complexity. For a program of size ℓ running for T steps, the above reduction yields a circuit C of size $T \cdot |C_{exe}| + (2T + \ell) \cdot |C_{mem}| + |C_{route}|$. Since

[3] In case A_i corresponds to B_j or I_j, the value O_i loaded is the encoding of the instruction, i.e. the concatenation of the machine operation code and the source and destination registers.

C_{exe}, C_{mem} are fixed for a given architecture (i.e., they are independent of T, ℓ), and C_{route} can be implemented using $O((T+\ell)\cdot\log(T+\ell))$ gates, we have $|C| = O((T+\ell)\cdot\log(T+\ell))$.

5.2 Our New RAM to Circuit Reduction

In this section, we present a "tigher" reduction than the one of [18] presented in Section 5.1.2, resulting in a more efficient argument for RAM programs. We rely on our new argument system as a backend, which is both interactive and has a circuit-independent preprocessing phase. More specifically, having a circuit-independent preprocessing phase allows us to produce a concretely smaller circuit where at each step the prover only proves the correct execution of the instruction that is actually executed by the RAM program on its specific inputs, as opposed to proving the correctness of a circuit evaluating all possible instructions. Next, the interactivity property allows us to replace the routing network used in [18] for checking trace consistency with an efficient interactive protocol for randomized polynomial identity testing. This reduces the prover's complexity from $O((T+\ell)\log(T+\ell))$ to $O(T+\ell)$ as well as improves the prover's concrete efficiency.

Our final circuit construction is shown in Figure 5.1(right). As in Section 5.1.2, we must check correctness of (1) instruction execution, (2) instruction fetches, and (3) memory accesses. Next we describe our implementation of these checks.

5.2.1 Ensuring Correct Instruction Execution

Let $tr = (S_1, I_1, S_2, I_2 \cdots, I_{T-1}, S_T)$. Recall that in the reduction described in Section 5.1.2, the correct execution of tr's instructions is checked via a universal C_{exe} which performs two sets of tests on every triple $S_i, I_i, S_{i+1} \in tr$. The first test (a) checks the correctness of O_i (i.e., that performing I_i on S_i results in O_i) while the second test (b) checks that the values from S_i are consistently propagated to S_{i+1} (including correct pc_i update and ordering of steps). Notice that while the second test is relatively simple and identical for all triples, the majority of C_{exe}'s gates are actually required for performing the first test. This is since this part of C_{exe} is often implemented by a composition of smaller circuits each of which can check the execution of a specific instruction, together with a multiplexer that specifies which instruction should be checked at this step. In order to optimize the size of C_{exe}, while maintaining the succinct representation of the result circuit C, we split C_{exe} into two sub-circuits which perform these two checks independently. For the second check we will the same circuit for all triples, whereas for the first one we we will use a circuit that can only verify the logic of the particular instruction I_i. Below, we describe in detail how these circuits are implemented.

Ensuring correct propagation of values. We define a circuit C_{time} that takes as input a triple S_i, I_i, S_{i+1}, and verifies that the value of the destination register in S_{i+1} is equal to O_i, all other registers in S_{i+1} remain unchanged, and pc_{i+1} was updated appropriately. Similar to Section 5.1.2, C_{time} also checks that S_i's step number is indeed i and that pc_i is equal to the alleged location of I_i in P (as encoded in I_i by

the prover). However, *unlike* Section 5.1.2, we stress that C_{time} *does not verify* that O_i is the correct output after executing I_i.

Verifying instruction execution. Let J be the number of instruction types supported by the RAM architecture. We include in the witness w an additional trace tr^{**} that is the result of sorting the pairs $(S_i, I_i) \in tr$ by the instruction type of I_i. Define a circuit $C_{val,j}$ which takes as input a pair $(S_i^{**}, I_i^{**}) \in tr^{**}$ and checks that S_i^{**} is a valid state for the instruction I_i^{**} of type j (i.e., O_i is correctly computed by executing I_i^{**} on S_i^{**}). In this way, $C_{val,j}$ is specialized to a specific instruction type. Moreover, since tr^{**} is sorted by instruction type, the copies of $C_{val,j}$ will also appear in C sorted by j. In this way, C can be succinctly described by $(k_1, \ldots, k_J)$, where k_j (for $j = 1, \ldots, J$) denotes the number of times instruction type j appears in trace tr when program P is executed on input x (where $\sum_j k_j = T$).

5.2.2 Verifying Instruction Fetches

As described above, [18] ensures program consistency by first storing the program to memory during the machine's booting phase. Next, each instruction is sequentially loaded from memory for execution. These operations are treated the same as regular memory stores and loads, and are checked by $T + \ell$ copies of C_{mem}. Here, we explain how the correctness of these operations can be checked more efficiently assuming instructions in the program are fixed and known to the verifier (i.e., if we assume that P does not contain self-modifying code, similar to [16]).

Unlike the reduction of Section 5.1.2, note that the trace tr does not include a

boot sequence. Instead, we observe that for each triple S_i, I_i, S_{i+1}, the circuit C_{time} already checks that pc_i is equal to the line number of I_i in P (as encoded in I_i by the prover). All that remains is to verify that I_i is the instruction in P with the same line number. Equivalently, let $\{P_1, \cdots, P_\ell\}$ be the set of instructions in P where each P_i is augmented to also contain its line number within P (as defined in Section 5.1.2). Then we only need to check that the sequence $\{I_1, \cdots, I_{T-1}\}$ is a multiset of $\{P_1, \cdots, P_\ell\}$ (the multiplicity of some P_i may be 0 to account for non-executed instructions). To that end, we add a circuit C_{fetch} that validates this multiset relation and leverages the interactive property of our argument scheme from Section 3. The circuit takes the sequence $I_1, \cdots, I_{T-1}$ from tr and a random value r (provided by the verifier) as input. C_{fetch} outputs the evaluation of its characteristic polynomial at point r, i.e., $\prod_{i=1}^{T-1}(I_i - r)$. The verifier also receives from the prover the multiplicity k_j of P_j in $\{P_1, \cdots, P_\ell\}$. Thus, he can compute himself the value $\prod_{j=1}^{\ell}(P_j - r)^{k_j} \stackrel{\text{def}}{=} \prod_{i=1}^{T-1}(I_i - r)$ and test whether it corresponds to the value output by the circuit. By the Schwartz-Zippel lemma, the probability the verifier accepts if the two polynomials are not the same (i.e., $\{I_1, \cdots, I_{T-1}\}$ is not a multiset of $\{P_1, \cdots, P_\ell\}$) is negligible. We stress that this is only secure if we ensure that the prover commits to the entire witness (including $I_1, \cdots, I_{T-1}$) before seeing r, as is the case in our construction in Section 3. In this way, we have replaced $T + \ell$ copies of C_{mem} with a smaller circuit C_{fetch} evaluating the characteristic polynomial at a random value which leads to concrete efficiency improvement.

5.2.3 Ensuring Memory Accesses

Similar to Section 5.1.2, in order to verify memory accesses (ensuring (3)) we include in w a trace $tr^* = (A_1, \cdots, A_T)$ sorted by the memory address being accessed (again with ties broken by step number and non-memory instructions located at the end of tr^*). Since the correctness of instruction fetches is already ensured (as described above), we only sort the states S_i in tr, and the length of tr^* now becomes T. For every two adjacent entries $A_i, A_{i+1} \in tr^*$ with outputs O_i, O_{i+1}, step numbers t_i, t_{i+1} and accessing addresses a_i, a_{i+1}, respectively, the circuit C_{mem} checks the same two conditions as in Section 5.1.2. Finally, note that the number of instruction that actually perform memory operations may be smaller than T, but we still include T copies of C_{mem} in C to account for the worst case. In practice however, it is almost certain that not every cycle will perform a memory access. E.g., even for a program that consists of a single for loop that simply loads a memory location per repetition, the total percentage of memory accesses is 25% (one instruction for the memory load, plus three for counter increase, loop bound check, and jump). Motivated by this, we exploit the circuit-independent pre-processing of our argument to modify C so that it only contains αT copies of C_{mem} where α is the percentage of general memory accesses over the total steps.

In order to achieve this, we split the witness to two separate parts. The first contains tr and tr^{**} sorted by time and instruction type, and the second contains tr^*. Recall that after the optimization above, tr^* only contains $A_{T+\ell}, \cdots, A_{2T+\ell}$, which is a permutation of $S_1, \cdots, S_T$ sorted by accessed memory addresses. Then,

by our design, if I_i is not a memory load/store instruction, we set the accessed memory address of S_i as 0 and all the values in S_i as 0s before sorting. In this way, the first $(1-\alpha)T$ states in $A_{T+\ell}, \cdots, A_{2T+\ell}$ are all zeros (assuming the real memory address starts from 1) and there is no need to check anything for these states, as they are not memory operations. Because of this layout, now the prover only includes $A_{T+\ell+\alpha T}, \cdots, A_{2T+\ell}$ in tr^*, and tells the verifier the number of non-memory operations. With these information, it is sufficient to validate the new tr^* is a permutation of non-zero states in tr using CMT on circuit C', and the technique is described in Section 4.5.2 for handling circuits that receive inputs at different levels. With this optimization, we manage to reduce the number of C_{mem} further from T to αT, which is a significant improvement in practice. However, the verifier now needs to run two VPD instances (once for each part of the witness).

5.2.4 Checking Consistency

Finally, it remains to check that tr^* and tr^{**} are indeed permutations of tr. Previous works [15, 16, 18] achieve this task by using routing networks, yielding a circuit of size $O((T+\ell)\log(T+\ell)$, for a T-step RAM program of size ℓ, and correspondingly increasing the prover's asymptotic running time from linear to quasilinear. Using routing networks to achieve this would yield a circuit of size $O((T+\ell)\log(T+\ell)$, for a T-step RAM program of size ℓ, which would correspondingly increase the prover's asymptotic running time from linear to quasilinear. Following the approach of Section 4.5.3, we leverage the interactive nature of our

argument in order to avoid the use of routing networks, replacing them with a simple interactive protocol that is similar to the one used above for verifying instruction fetches. The result is that our prover's running time is only $O(T + \ell)$, i.e., asymptotically the same as simply evaluating the program.

More specifically, assume the prover holds lists $x_1, \ldots, x_m$ and $x'_1, \ldots, x'_m$ and wants to convince the verifier that they are a permutation of each other. Consider a circuit C_{perm} that takes $x_1, \ldots, x_m$ and $x'_1, \ldots, x'_m$ (provided by the prover) and a random point r (provided by the verifier) and outputs the result of $\prod_{i=1}^{m}(x_i - r) - \prod_{i=1}^{m}(x'_i - r)$. If the two lists are permutations of each other the output is always zero, otherwise by the Schwartz-Zippel lemma it is zero with negligible probability.[4] Finally, evaluating this polynomial requires $O(m)$ gates. For our argument, we use two executions of this interactive protocol, one for the pair tr, tr^* and one for tr, tr^{**}, in a way that ensures that C outputs zero only if C_{perm} outputs zero both times. From the above analysis, each of these circuits consists of $O(T + \ell)$ gates. We stress that it is crucial to have the prover commit to the two lists ahead of time, in particular before seeing r, for security purposes. This is enforced by our argument as $\mathcal{P}$ commits to the entire witness w in the first step of the protocol (cf. Construction 3, Evaluation Phase, Step 1).

[4] As a state (e.g., A in tr^*) contains multiple values such as O and t and we want to ensure they are permuted together, we pack the values before the check (e.g., for W-bit values (a, b, c), we set $x = a \times 2^{2W} + b \times 2^W + c$). If the result of a single pack overflows the field, we pack the values multiple times with respect to the first value. In our implementation, we use a 254-bit prime field, which allows packing of 7 32-bit numbers. We also use the same technique to ensure that S_i^{**} and I_i^{**} in tr^{**} are permuted together.

Finally, we apply our new argument system presented in Construction 3 on the circuit presented above as a backend. This leads to a VC protocol for arbitrary RAM programs. We are now ready to state the following result:

Theorem 7. *Let ℓ be a program length parameter, T be a time bound and let n be an input bound. Assuming that Construction 2 is an extractable verifiable polynomial delegation protocol, then combining the results of Section 5.2 with Construction 3 we obtain an argument system for the relation $RAM_{\ell,n,T}$ (as per Definition 5). Moreover, as the sizes of C_{time}, C_{val} and C_{mem} are constants which are independent of n, T, ℓ, the running time of $\mathcal{P}$ is $O(n+T+\ell)$ and that of $\mathcal{V}$ is $O(n+\ell+poylog(T))$. This yields a* succinct *argument with* polylog $(n+\ell+T)$ *rounds of interaction.*

The theorem directly follows Theorem 5, and we omit the proof here.

5.3 Experimental Results

Software and hardware. We implemented our constructions (including the RAM reduction, circuit generator, CMT protocol, and VPD protocol) in C++. We use the GMP library [3] for field arithmetic and OpenSSL's [8] SHA-256 implementation for hashing. For the bilinear pairing we use the ate-paring library [1] on a 254-bit elliptic curve.

We run our experiments on an Amazon EC2 m4.2xlarge machine having 32 GB of RAM and an Intel Xeon E5-2686v4 CPU with eight 2.3 GHz virtual cores. Our implementations are *not* parallelized and only use a single CPU core.

5.3.1 Comparison with vnTinyRAM and Buffet

In this section, we compare the performance of our system to existing systems for verifiable RAM. We compare to Buffet [83], a verifiable RAM system with program-specific prepossessing (where the parameters generated by the trusted preprocessing can only be used to verify one specific program on different inputs) and vnTinyRAM [18], a universal verifiable RAM system (where the parameters generated by the trusted preprocessing can be used to verify any program up to some bound on the number of steps). We also measure the performance of our system against naive unverified execution of the RAM program. Finally, in Section 5.3.2 we also discuss comparisons to other verifiable RAM systems.

Benchmark. As a benchmark, we evaluate the RAM programs from [83] (see Table 5.2). Following that work, we benchmark our system using programs of three types.

1. **Circuit friendly.** The function computed by these programs has a very efficient circuit representation. We use *matrix multiplication* as an example.

2. **Fixed memory access and instruction patterns.** These programs do not exploit the full generality of RAM machines, i.e., their memory-access patterns and control flow do not depend on the program's inputs. This allows for a tighter RAM-to-circuit reduction since it can be determined ahead of time which instruction will be executed at each time step. Thus, the produced circuit only needs to handle a specific instruction per cycle. We use *pointer chasing* and

Benchmark	Input Size	# of Cycles	Native
1: Matrix Mult.	n=215	96M	42ms
2: Pointer Chasing	$n = 16634$	50K	22μs
3: Merge Sort	$n = 512$	65K	28μs
4: KMP Search	$n = 2900, k = 256$	30K	13μs
5: Sparse Mat-Vec Mult.	$n = 1150, k = 2300$	27K	12μs

Table 5.2: Benchmarks in our experiments. We report the input size, the number of CPU cycles and the native running time on verifier for the instances we used in Table 5.3 and 5.4.

merge sort as examples of such programs.

3. **Input-dependent memory access and instruction patterns.** Such RAM programs use the full generality of RAM machines since they have input-dependent control flow and memory-access patterns. In particular, the circuit generated by the RAM reduction must be able to handle multiple possible instructions at every step. We use *KMP string matching* [58] and *CSR sparse matrix-vector multiplication* [47] as examples of such programs.

Buffet evaluation methods. Buffet's front-end takes a RAM program and outputs a circuit that verifies its execution and its back-end uses a circuit-based VC system based on Pinocchio [71]. We evaluate Buffet using the released code [2].

vnTinyRAM evaluation methods. We evaluate vnTinyRAM [18] using the

code at [6]. As the code that takes a TinyRAM program and outputs the traces for vnTinyRAM is not available, we are unable to produce vnTinyRAM traces corresponding to the execution of any benchmark RAM program. Instead, we estimate the cost of vnTinyRAM by running the prover on traces of appropriate length resulting from execution random machine instructions. Since the performance of vnTinyRAM only depends on the total number of CPU steps and not on the instruction being executed at each step, this estimate is accurate.[5]

Using a different back-end for vnTinyRAM and buffet. Both Buffet and vnTinyRAM can be re-factored to use the more recent construction of [53] as their back-end. This would result in an approximate improvement of 30% in their setup, prover time and public key size as well as 50% improvement in their proof and verification key sizes. This would also improve verification time by 3×, as per the benchmarks of [6].

vRAM evaluation methods. For vRAM, we implemented our own TinyRAM simulator to output the program traces used by our prover and verifier backend. We then adapted the assembly code for the programs in the Buffet benchmark, and ran them in our TinyRAM simulator to obtain execution traces, which we provided to prover-verifier backend. In order to measure the cost of our system vs. naive unverified execution, we estimate the execution time of random instructions on a

[5] A version of vnTinyRAM that removes unnecessary instructions in each step after running the particular program to be verified was released by Wahby et al. [83]. However, since the prover in this program-specific version is unable to handle arbitrary RAM programs, it is not appropriate for our comparison.

single-threaded 2.3 GHz CPU core.

Experimental results. The results of the comparison are summarized in Tables 5.2, 5.3, 5.4 and 5.5 as well as in Figure 5.2. We executed each program on the largest input size reported in [83]. Table 5.2 summarizes their input size, number of CPU cycles and the native running time if executed on the verifier locally. As vnTinyRAM cannot handle such large parameters, we estimate its cost by extrapolation, assuming linear growth. This yields a conservative estimate since the overhead of vnTinyRAM's prover grows quasilinearly (rather than linearly) with the number of RAM instructions. We report setup time, prover and verifier time, proof size and the size of the circuit verifying the RAM program. In Figure 5.2, we show the prover time and memory consumption of the three systems versus the number of CPU steps. In vRAM, these are mainly determined by the number of CPU steps executed by the benchmark, rather then the specific choice of instructions executed in these steps. Consequently, we show the performance of pointer chasing as a representative example, with other programs behaving similarly. Since Buffet optimizes the circuit generated based on a particular benchmark program, we report two cases: one is pointer chasing, which is a fixed-RAM program, and the other is string search, which is a data dependent RAM program.

Comparison with vnTinyRAM. Both our system and vnTinyRAM can verify the execution of arbitrary programs with a single setup. As shown in Table 5.3, 5.4 and Figure 5.2 (left), for all benchmarks except matrix multiplication, our system achieves an approximate 8× improvement in setup time and 9× improvement in

	Setup Time (min)			Prover Time (min)		
	TinyRAM	Buffet	vRAM	TinyRAM	Buffet	vRAM
#1	460000*	16.6	38.7	290000*	14.4	0.65
#2	310*	20.0		150*	11.2	17.3
#3		16.1		200*	9.6	21.21
#4		22.9		90*	12.6	9.2
#5		20.8		82*	11.8	10.2

Table 5.3: Comparison of the performance of vRAM versus Buffet and vnTinyRAM (Setup time and prover time. * denotes simulation due to memory exhaustion).

prover time compared to vnTinyRAM. Note that vnTinyRAM is unable to exploit the fact that matrix multiplication is circuit-friendly, leading to large circuit size, setup, prover and verifier times. Since our system uses a preprocessing phase that only depends on the input size and is otherwise agnostic to the program representation, for circuit-friendly benchmarks we are able to directly use the program's circuit representation and thereby obtain an improvement of more than 4 orders of magnitude for setup time and 5 orders of magnitude for proving time compared to vnTinyRAM.[6]

[6]Note that in order to support all the benchmarks in Table 5.3, vnTinyRAM only needs to execute a single preprocessing phase which is as large as the largest instance, i.e. matrix multiplication. However, for fair comparison, we report a separate setup time for the 4 RAM-friendly programs and compare the performance of our system to this number.

	|C| (Millions of gates)				Verification Time (ms)		
	TinyRAM	Buffet	vRAM (mult/total)		TinyRAM	Buffet	vRAM
#1	240000*	9.9	9.9	19.8	422*	401	26
#2	125*	8.6	38.5	150.8	56*	69	93
#3	164*	7.9	36.2	148.3	9*	8	91
#4	75*	10.5	18.2	72.4	15*	20	84
#5	68*	9.4	18.1	74.3	20*	15	85

Table 5.4: Comparison of the performance of vRAM versus Buffet and vnTinyRAM (Number of gates and verification time. * denotes simulation due to memory exhaustion).

The speedup obtained by vRAM is due to (1) the better RAM-to-circuit reduction from Section 5.2; and (2) the faster argument system from Section 3. To isolate the effect of (1), in Table 5.4 we report the number of gates in the circuits produced by our reduction. Note that unlike vnTinyRAM and Buffet, in vRAM all types of gates (numbers reported in the last column) contribute to the prover time, instead of multiplication gates only.[7] Thus, to facilitate the comparison between vnTinyRAM's circuit reduction and our circuit reduction, we also report the number of multiplication gates in the table. As shown in Table 5.4, the number of

[7]Both vnTinyRAM and Buffet use the notion of quadratic constraints with each constraint verifying that the product of the outputs of two unbounded fan-in gates equals to the output of a third unbounded fan-in add gate.

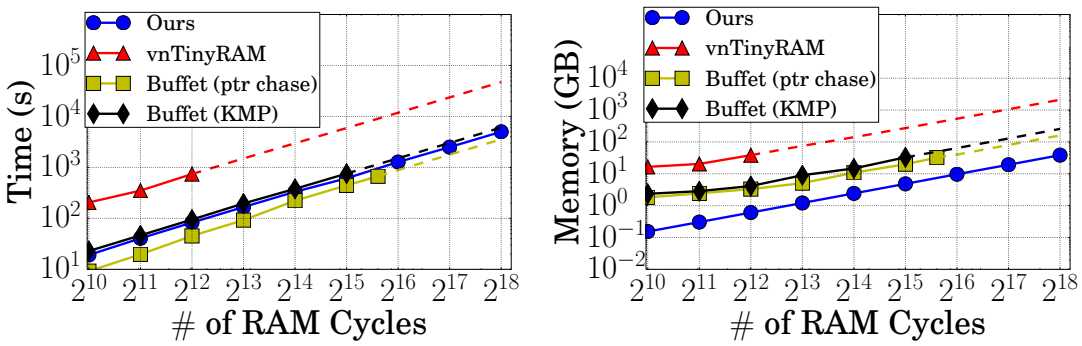

Figure 5.2: Prover time (left) and memory consumption (right) of our construction vs vnTinyRAM and Buffet for various number of CPU steps.

multiplication gates in our system is 3.3–4.5× less than in vnTinyRAM. Regarding (2), the performance of our argument system is demonstrated in more detail in Section 5.3.4, where we show that the per-gate cost of our system is lower than that of QAP-based systems.

Comparison with Buffet. The main advantage of our system compared to Buffet is that it can support arbitrary programs with a single setup. As shown in Table 5.3, the setup time for our system is 38.7 minutes for any program that runs for up to 65K CPU steps. Although the setup time of Buffet for the indicated programs is lower, an independent setup would have to be run for each different program to be verified (and the set of programs being verified must be known at the time setup is run). Moreover, we note that Buffet's setup time would likely be larger than ours if used for a program running for 65K CPU steps (which none of the benchmarks do).

Overall, the prover time of our system is comparable to that of Buffet. On one hand, for programs with fixed memory access and instruction patterns (such as pointer chasing and merge sort) Buffet can perform numerous optimizations, since

the instruction to be executed in each CPU step is pre-determined. This allows Buffet to highly customize the resulting circuit. Nonetheless, our system is still only around 2× slower than Buffet while avoiding program-dependent preprocessing. On the other hand, for programs with input-dependent memory and instruction patterns (such as KMP string search and sparse matrix-vector multiplication), our system actually *outperforms* Buffet, despite the fact that the latter can optimize the circuit during preprocessing. Moreover, as mentioned in [83, Section 4.3], if a program has deep nesting of data dependent loops or complex conditions (e.g., a state machine), the compiler of Buffet may have to incur a significantly higher overhead, since the amount of applicable optimizations will be limited. However, the performance of our construction is not adversely affected by such programs therefore our speedup compared to Buffet can be higher.

Finally, we note that when the program is circuit-friendly, e.g., matrix multiplication, Buffet can also represent the computation using a circuit. In this case, the circuit is exactly the same in both systems, and the prover time of our system is 22× faster than Buffet, since our argument system outperforms Buffet's Pinocchio-based argument [71].

Memory consumption. Another advantage of our system is that it uses much less memory in order to prove the same statement. As shown in Figure 5.2 (right), the memory consumption of our system is 55–110× less than vnTinyRAM, yielding a two orders of magnitude improvement. The memory consumption is also $4 - 8\times$ less than Buffet. In particular, on a desktop machine with 32GB of RAM, we can

	#1	#2	#3	#4	#5
Proof Size (KB)	4	256	255	236	235
Memory Usage (GB)	3.6	7.6	7.7	3.8	3.8

Table 5.5: Proof size and memory usage of vRAM.

execute 2^{18} CPU steps, while vnTinyRAM can only reach 2^{12} steps, and Buffet can reach $2^{15} - 2^{16}$ steps. We also report the memory consumption for the benchmarks we run in Table 5.5. The improvement is largely due to our reliance on the CMT protocol which imposes a minimal memory overhead for the non-input part of the circuit. In fact, although the circuit size is much larger than the input size, the memory usage of our VPD protocol and the CMT protocol are on the same order. In addition, in the VPD protocol, the memory is mainly used for storing the public key, thus the usage is roughly the same in the setup and the evaluate phase of VPD.

Verification time and proof size. We next compare the verification time and communication cost of our system with vnTinyRAM and Buffet, both of which outperform our system. In particular, the verification time is 9–56ms for vnTinyRAM and 8–35ms for Buffet (except matrix multiplication). Also, vnTinyRAM and Buffet inherit a proof size of 288 Bytes from QAP-based SNARKS. For comparison, the verification time and the overall communication cost for our construction varies on different sizes of circuits. As shown in Table 5.4 and 5.5, the verification time is 84–93ms and the communication is 235–256KB for different programs. However, we believe that these are very modest quantities for any modern machine.

Proving 2 million instructions. To demonstrate the ability of our construction to handle the task of verifying programs that run for large amounts of CPU steps, we also ran our system on an Amazon EC2 m4.16xlarge machine featuring 256GB of RAM and an Intel Xeon E5-2676v3 CPU with 64 virtual cores running at 2.4GHz. Using this machine, we executed our system for programs consisting of 2^{21} instructions. The reported prover's time is 51000s, the memory consumption grows to 252 GB and the total number of gates in the circuit is 4.8 billion. While these numbers are concretely large, we stress that, to the best of our knowledge, this is by far the largest reported successfully performed instance of verifiable RAM computation. In particular, this instance is about 65× larger than the largest instance reported in [18] (which was achieved by using a 256GB solid state drive as additional memory space). Finally, the reported verification time was less than 105ms and the total communication cost was 336.5KB.

5.3.2 Comparison to Other RAM-based VC systems

In this section, we briefly discuss the performance of our system compared to other RAM-based VC systems.

Pantry and SNARKs for C. Pantry [29] and SNARKs for C [16] are two VC schemes that predate Buffet and vnTinyRAM, with their performance subsumed by those systems (see [83, Figure 10] and [18, Figure 3]).

Exploiting data parallel structure via bootstrapping. Geppetto [37] is a VC system that takes a large circuit, splits it into sub-circuits, and preprocesses each

sub-circuit with a SNARK separately. An additional SNARK is then applied to aggregate and verify the outputs of all sub-circuits in a "bootstrapping" step. Though verifiable RAM is not explicitly considered in [37], the system can be potentially applied to circuits checking the correctness of a RAM program, such as ones in Sections 5.1.2 and 5.2. Due to the data parallel structure of these circuits, Geppetto can reduce the setup time asymptotically (e.g., only one setup for the sub-circuit C_{mem}, C_{time} etc.). However, it introduces a big concrete overhead for both setup and prover time because of the bootstrapping phase. For example, it requires $\sim 30,000$-$100,000$ gates to bootstrap one small sub-circuit of just 500 gates [37, Section 7.3.1].

Constant or no preprocessing. Two alternative approaches for RAM-based VC are suggested in [13, 17] by Ben-Sasson et al. The first uses composition of elliptic curves to recursively apply a SNARK in a sequence of T fixed-size circuits, each of which validates the state of a single previous CPU step, executes the next CPU step, and outputs the new state. In this way, the resulting setup time is constant. The second constructs a RAM-based VC without any preprocessing by using PCPs. Both these systems incur a very large concrete overhead on the prover. It takes 35.5 seconds/cycle for the first system [17, Figure 1], which is about 3000× slower than ours. For the second one, it takes 0.33 seconds/cycle using 64 threads in parallel [13, Figure 1], which roughly corresponds to 21.1 seconds/cycle using single thread [13, Section 2]. This is compared to our single threaded implementation which achieves 0.015 seconds/cycle. We leave the task of achieving a speedup for our system via parallelization as future work.

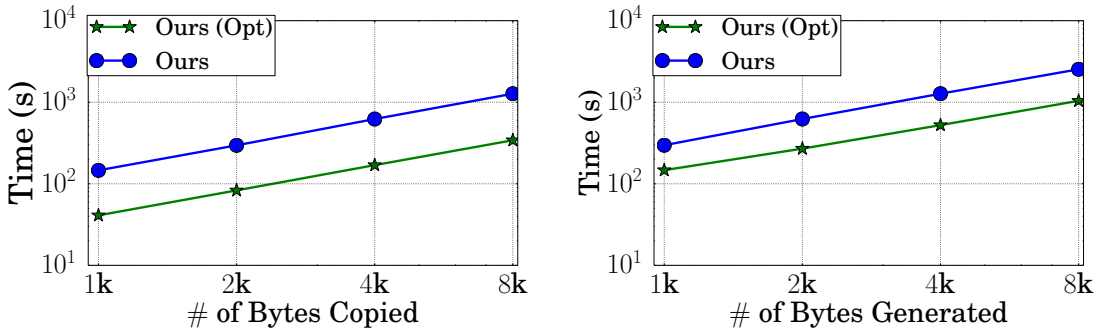

Figure 5.3: Prover time for evaluating memcpy (left), RC4 (right) using vRAM with (green) and without (blue) the optimizations of Section 5.3.3. For memcpy we vary the size of the copied memory block and for RC4 we vary the number of pseudorandom bytes generated.

5.3.3 Just-in-Time Architecture

Next, we use the architecture-independent preprocessing property of our scheme to improve performance for specific tasks. Common just-in-time compilation methods are used to optimize the executed code for a specific architecture. The circuit independent preprocessing feature of our construction allows us to take this approach further and modify the machine's architecture in order to better fit a specific program *after* executing it, when the program's exact behavior on its inputs is known. We illustrate this using two benchmarks from [16, 18]. We stress that since our protocol has architecture-independent preprocessing we are able to change the architecture without rerunning the preprocessing phase. In particular, the following results were achieved with a single preprocessing execution. In all cases, the verifier's runtime remained below 150ms.

Improving performance by adding instructions. Figure 5.3(left) shows prover's time for evaluating a program which copies consecutive blocks of memory from one location to another (e.g., `memcpy`). We achieve a 3.6× improvement by introducing a memory instruction which (1) copies a byte from memory address A to memory address B and (2) increments A and B by 1 for the next loop iteration. This reduces the number of gates in the obtained circuit, thus yielding lower prover time. In this case, we did not modify any of the machine's other parameters (e.g., number of registers and register size).

Improving performance by changing register sizes. Next, Figure 5.3(right) shows prover's time for evaluating a RC4 pseudorandom generator on a highly specialized architecture. More specifically, we modified the machine to contain 3 8-bit registers, a 32-bit address register for memory accesses and a 32-bit program counter. Each RC4 round was implemented using 16 instructions operating over the 8-bit registers. Notice the 2.4× speedup compared to the non-optimized version, which again results from the overall reduction of necessary gates in order to generate one pseudorandom byte.

5.3.4 Microbenchmarks

Finally, we report here the individual performance of our construction's main building blocks.

Verifiable polynomial delegation. Table 5.6 shows the prover time of our implementation of the VPD from Section 3.1. The prover time is about $12\mu s$ per input

Benchmark	VPD Time/Input	CMT Time/Gate	Buffet Time/Gate	vnTinyRAM Time/Gate
#2	11.32	5.42	77.50	72.20
#3	13.17	6.36	68.34	72.15
#4	13.23	6.18	74.97	72.33
#5	12.90	5.77	72.10	72.37

Table 5.6: Per gate (input) prover time for our VPD and CMT, Buffet and vnTinyRAM for the last 4 RAM programs in the benchmark (same order and size as in Table 5.3 and 5.4). Time reported in μs.

gate, which is about 8× faster than that of [88]. This is due to (i) our improved VPD construction (amounting to around 2-4×) and (ii) due to the fact that 85% of the inputs used in our RAM reduction are field elements that encode single bit values (due to bit decomposition, register indices and flags), which leads to faster exponentiation times for the VPD prover.

CMT protocol. Next, we evaluate the performance of our CMT protocol. As can be seen in Table 5.6, the average time required per gate for the CMT prover is about $6\mu s$, which is about 4× slower than the $1.7\mu s$ number reported in [88]. This is because we implemented our new CMT protocol supporting circuits with different copies of sub-circuits in Section 3.2, while the CMT protocol for regular circuits is used in [88]. Both the per-input time for the VPD protocol and the per-gate time for the CMT protocol are much faster than the per-gate time for Buffet

and vnTinyRAM.

Circuit generator. Finally, we report the number of gates required by our reduction to verify a TinyRAM cycle. We measure this by dividing the total number of gates of the circuits produced by the experimental evaluation of Section 5.3.1 over the number of TinyRAM steps. For our tested programs this circuit contained about 2500 gates, 600 of which are multiplications (for comparison, vnTinyRAM takes roughly 2000 multiplication gates, as reported in [18]. Notice that the work of [18] only needs to report the number of multiplication gates while we must report on the total number of gates.

Chapter 6: Zero Knowledge

Our argument scheme presented in Section 3 lacks one crucial property: it is not *zero-knowledge*. Moreover, while state-of-the-art SNARKs from quadratic-arithmetic programs can be made zero-knowledge by simply randomizing proof elements, this approach is not directly compatible with our argument scheme.

In this chapter, we show a zero-knowledge version of our argument scheme. At a high level, in order to obtain a zero-knowledge property we replace both of the underlying components, the VPD and the CMT protocols, with zero-knowledge variants. For the CMT component, this can be achieved by running the entire protocol inside homomorphic commitments (as first observed in a a more general context by Cramer and Damgård [38]). For the VPD part, we devise a new construction that we call zk-VPD which is used to allow the prover to produce a proof about the correctness of a *commitment* to the correct evaluation of a polynomial (rather than proving correctness of the evaluation itself).

Asymptotically, our protocol has the same performance as that in Section 3 and has a preprocessing phase that only depends on an upper bound on the size of the input, but not on the specific circuit to be evaluated. In practice, we would expect it to have a slightly larger overhead for both parties (due to the increased

number of cryptographic operations).

In this chapter, we first go through building blocks we use to achieve zero-knowledge in Section 6.1, and present the zero-knowledge variants of the VPD protocol and CMT protocol in Section 6.2 and 6.3. We then present the construction of our new zero-knowledge argument system in Section 6.4.

6.1 Building Blocks

Linearly homomorphic commitment scheme. We assume the existence of a commitment scheme $\mathsf{Comm} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ that has $\mathbb{Z}_q$ (for prime q) as its message space. This could be instantiated by the Pedersen commitment scheme [73], for example. We assume:

- $\mathsf{Setup}(1^\lambda)$ outputs public commitment parameters cp.
- $\mathsf{Com}(\mathsf{cp}, \mathsf{m}, \mathsf{r})$ on input a message $m \in \mathbb{Z}_q$ and randomness r outputs a commitment com.
- $\mathsf{Open}(\mathsf{cp}, \mathsf{com}, \mathsf{m}, \mathsf{r})$ accepts iff $\mathsf{Com}(\mathsf{cp}, \mathsf{m}, \mathsf{r}) = \mathsf{com}$.

We also require that there exists an efficient algorithm $\mathsf{Evaluate}(\mathsf{cp}, \mathsf{com}_1, \ldots, \mathsf{com}_n, x_1, \ldots, x_n)$ that on input n valid commitments (for some randomness values r_i) for $m_1, \ldots, m_n$ and coefficients $x, \ldots, x_n \in \mathbb{F}$, outputs new commitment com' such that $\mathsf{Open}(\mathsf{cp}, \mathsf{com}', \sum_1^{\mathsf{n}} \mathsf{x_i m_i}, \mathsf{r}')$ accepts, where r' is computed as a function of $(r_1, \ldots, r_n, x_1, \ldots, x_n)$. For Pedersen commitments, this can be easily achieved by having $\mathsf{Evaluate}(\mathsf{cp}, \mathsf{com}_1, \ldots, \mathsf{com}_n, x_1, \ldots, x_n)$ outputs $\mathsf{com}' = \prod_1^n \mathsf{com}_i^{x_i}$ and $r' = \sum_1^n x_i r_i$.

Zero-knowledge argument for commitment-preimage equality. We assume the existence of a zero-knowledge argument $\mathcal{ZK}_{eq}$ for proving that two commitments produced with Comm have the same pre-image. Somewhat informally, we write $\mathcal{ZK}_{eq}(m, r_1, r_2; \mathsf{com}_1, \mathsf{com}_2) \rightarrow 1/0$ to denote the interaction between a prover that holds $\mathsf{cp}, m, r_1, r_2, \mathsf{com}_1, \mathsf{com}_2$ such that $\mathsf{Open}(\mathsf{cp}, \mathsf{com_i}, \mathsf{m}, \mathsf{r_i})$ accepts for $i = 1, 2$, and a verifier that holds $\mathsf{cp}, \mathsf{com}_1, \mathsf{com}_2$ will eventually accept if he believes they have the same preimage and he will reject otherwise. For completeness, we require that the verifier accepts with probability 1 for a valid statement. For soundness, we require that for any probabilistic polynomial-time (cheating) prover algorithm, the verifier will accept a false statement with probability negligible in λ. Zero-knowledge dictates that the verifier learns nothing about m_1, m_2. For the Pedersen commitment scheme, such a protocol can be instantiated by first using a sigma-protocol (e.g., the one from [25]) and then using standard techniques to make it full zero-knowledge (e.g., [44]).

Zero-knowledge argument for product of preimages. We assume the existence of a zero-knowledge argument $\mathcal{ZK}_{prod}$ for proving that for three commitments $\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3$ produced with Comm it holds that the preimage of the last is the product (in $\mathbb{F}$) of the preimages of the first two. We write $\mathcal{ZK}_{eq}(m_1, m_2, r_1, r_2, r_3; \mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3) \rightarrow 1/0$ to denote the interaction between a prover that holds $\mathsf{cp}, m_1, m_2, r_1, r_2, r_3, \mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3$ such that $\mathsf{Open}(\mathsf{cp}, \mathsf{com_i}, \mathsf{m_i}, \mathsf{r_i})$ accepts for $i = 1, 2$, and $\mathsf{Open}(\mathsf{cp}, \mathsf{com}_3, \mathsf{m}_1 \cdot \mathsf{m}_2, \mathsf{r}_3)$ accepts, and a verifier that holds $\mathsf{cp}, \mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3$. For completeness, we require that the verifier accepts with probability 1 for

a valid statement. For soundness, we require that for any probabilistic polynomial-time (cheating) prover algorithm, the verifier will accept a false statement with probability negligible in λ. Zero-knowledge dictates that the verifier learns nothing about m_1, m_2. For the Pedersen commitment scheme, this can again be instantiated via a standard combination of [25, 44].

Exractability. Finally, we want the commitment scheme to be *extractable* in the manner described in [21] for the case of collision-resistant functions, i.e., it should not be possible to output a valid commitment without knowing a corresponding pre-image. Somewhat informally, this is captured by the existence of an adversary-specific extractor that (given access to the adversary's code, random tape and auxiliary input) can output a pre-image for any commitment value the adversary produces with all but negligible probability. For the Pedersen commitment scheme this can be achieved, under Assumption 2, via the following modifications. (1) Parameters cp also include value g^{β} for $\beta \in \mathbb{F}$ chosen uniformly at random. (2) Commitments consist of a pair of values from $\mathsf{com}, \mathsf{com}' \in \mathbb{G}$ such that $\mathsf{com}' = \mathsf{com}^{\beta}$. (3) Upon receiving such a commitment $\mathsf{com}, \mathsf{com}'$, the receiving party must check the relation $e(\mathsf{com}, g^{\beta}) = e(\mathsf{com}, g)$ and abort if the check fails. To ease notation, in the following when describing a commitment value we will only refer to com and we will omit the above validity check from the description of our protocols.

6.2 Zero-Knowledge Polynomial Commitment

In this section we present our zero knowledge polynomial delegation scheme. At a high level, the main idea is to modify the polynomial delegation scheme of [88] to output a commitment to the evaluation instead of the evaluation itself. That is, instead of having $\mathsf{Evaluate}$ output the value y of the polynomial f when evaluated on the input x together with a suitable proof π, the zero-knowledge polynomial delegation commitment scheme outputs a statistically hiding and computationally binding commitment com_y to the value of y (in addition to the proof π). This hides the value of y but still supports verifying that com_y is indeed a commitment to $f(x)$.

We first present our definition of a zero-knowledge polynomial delegation scheme.

Definition 6. *Let $\mathbb{F}$ be a finite field, $\mathcal{F}$ be a family of ℓ-variate polynomials over $\mathbb{F}$, and d be a variable-degree parameter.* $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Evaluate}, \mathsf{CheckCom}, \mathsf{Ver})$ *constitute a* zero-knowledge verifiable polynomial-delegation protocol *for $\mathcal{F}$ if:*

- ***Perfect completeness.*** *For any polynomial $f \in \mathcal{F}$ and value t, the following probability is 1:*

$$\Pr_{r_f, r_y}\left[\begin{array}{c} (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d) \\ \mathsf{com}_f \leftarrow \mathsf{CommitPoly}(f, r_f, \mathsf{pp}) \\ (\mathsf{com}_y, \pi) \leftarrow \mathsf{CommitValue}(f, t, f(t), r_f, r_y, \mathsf{pp}) \end{array} : \begin{array}{c} \mathsf{CheckCom}(\mathsf{com}_f, \mathsf{vp}) = 1 \wedge \\ \mathsf{Ver}(\mathsf{com}_f, t, \mathsf{com}_y, \pi, \mathsf{vp}) = 1 \end{array}\right]$$

- ***Binding.*** *For any* PPT *adversary* Adv *and benign auxilary inputs z_1, z_2 the fol-*

lowing probability is negligible:

$$\Pr\left[\begin{array}{l|l} & \mathsf{CheckCom}(\mathsf{com}_f^*, \mathsf{vp}) = 1 \wedge \\ (\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d) & \mathsf{Ver}(\mathsf{com}_f^*, t^*, \mathsf{com}_y^*, \pi^*, \mathsf{vp}) = 1 \wedge \\ (\pi^*, \mathsf{com}_f^*, \mathsf{com}_y^*, \mathsf{state}) \leftarrow \mathsf{Adv}(1^\lambda, z_1, \mathsf{pp}) & \mathsf{com}_f^* = \mathsf{CommitPoly}(f^*, r_f^*, \mathsf{pp}) \wedge \\ (f^*, t^*, y^*, r_f^*, r_y^*) \leftarrow \mathsf{Adv}(1^\lambda, z_2, \mathsf{state}, \mathsf{pp}) & (\mathsf{com}_y^*, \pi) = \mathsf{CommitValue}(f^*, t^*, y^*, r_f^*, r_y^*, \mathsf{pp}) \\ & \wedge\, f^*(t^*) \neq y^* \end{array}\right].$$

- ***Zero Knowledge.*** *For security parameter λ, polynomial f, adversary* Adv*, and simulator* Sim *consider the two experiments* $\mathsf{Real}_{\mathsf{Adv},f}(1^\lambda), \mathsf{Ideal}_{\mathsf{Adv}}(1^\lambda)$*, defined as follows.*

$\underline{\mathsf{Real}_{\mathsf{Adv},f}(1^\lambda)}$*:*

1. $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, \ell, d)$
2. *Generate r_f uniformly at random*
3. $\mathsf{com}_f \leftarrow \mathsf{CommitPoly}(f, r_f, \mathsf{pp})$
4. $k \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, \mathsf{vp})$
5. *For $i = 1, \ldots, k$ repeat:*

(a) *Generate r_i uniformly at random*

(b) $t_i \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, \mathsf{com}_{y_1} \ldots, \mathsf{com}_{y_{i-1}}, \pi_1, \ldots, \pi_{i-1}, \mathsf{vp})$

(c) $(\mathsf{com}_{y_i}, \pi_i) \leftarrow \mathsf{CommitValue}(f, t_i, f(t_i), r_f, r_i, \mathsf{pp})$

6. $b \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, (\mathsf{com}_{y_1} \ldots, \mathsf{com}_{y_k}, \pi_1, \ldots, \pi_k), \mathsf{vp})$
7. *Output b*

$\underline{\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)}$:

1. $(\mathsf{com}_f, \mathsf{pp}, \mathsf{vp}, \sigma) \leftarrow \mathsf{Sim}(1^\lambda, \ell, d)$
2. $k \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, \mathsf{vp})$
3. *For* $i = 1, \ldots, k$ *repeat:*
   (a) $t_i \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, \mathsf{com}_{y_1} \ldots, \mathsf{com}_{y_{i-1}}, \pi_1, \ldots, \pi_{i-1}, \mathsf{vp})$
   (b) $(\mathsf{com}_{y_i}, \pi_i, \sigma) \leftarrow \mathsf{Sim}(t_i, \sigma, \mathsf{pp})$
4. $b \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{com}_f, (\mathsf{com}_{y_1} \ldots, \mathsf{com}_{y_k}, \pi_1, \ldots, \pi_k), \mathsf{vp})$
5. *Output* b

We require that for any PPT *adversary* Adv *and all* $f \in \mathbb{F}$*, there exists a simulator* Sim *such that the following is negligible*

$$\left|\Pr\left[\mathsf{Real}_{\mathsf{Adv},f}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda) = 1\right]\right|.$$

Finally, we say that (KeyGen, Commit, Evaluate, CheckCom, Ver) *are an* extractable *zero-knowledge verifiable polynomial-delegation protocol for* $\mathcal{F}$ *if* (KeyGen, Commit, Evaluate, CheckCom, Ver) *satisfy the following extraction requirements instead of the above defined soundness requirement.*

- ***Polynomial extractability.*** *For any* PPT *adversary* Adv *there exists a polynomial-time algorithm* $\mathcal{E}$ *with access to* Adv′*s random tape such that for all benign auxil-*

iary inputs $z \in \{0,1\}^{poly(\lambda)}$ *the following probability is negligible:*

$$\Pr\left[\begin{array}{l} (\mathsf{pp},\mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda,\ell,d); \\ \mathsf{com}_f^* \leftarrow \mathsf{Adv}(1^\lambda,\mathsf{pp},z); \\ (f,r_f) \leftarrow \mathcal{E}(1^\lambda,\mathsf{pp},z) \end{array} : \begin{array}{c} \mathsf{CheckCom}(\mathsf{com}^*,\mathsf{vp}) = 1 \wedge \\ \mathsf{com}_f^* \neq \mathsf{CommitPoly}(f,r_f,\mathsf{pp}) \end{array}\right].$$

- ***Evaluation extractability.*** *For any* PPT *adversary* Adv *there exists a polynomial-time algorithm* $\mathcal{E}$ *with access to* Adv′*s random tape such that for all benign auxiliary inputs* $z \in \{0,1\}^{poly(\lambda)}$ *the following probability is negligible:*

$$\Pr\left[\begin{array}{l} (\mathsf{pp},\mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda,\ell,d) \\ (t^*,\pi^*,\mathsf{com}_f^*,\mathsf{com}_y^*) \leftarrow \mathsf{Adv}(1^\lambda,\mathsf{pp},z) \\ (f,r_f,y,r_y) \leftarrow \mathcal{E}(1^\lambda,\mathsf{pp},z) \end{array} : \begin{array}{c} \mathsf{CheckCom}(\mathsf{com}_f^*,\mathsf{vp}) = 1 \wedge \\ \mathsf{Ver}(\mathsf{com}_f^*,t^*,\mathsf{com}_y^*,\pi^*,\mathsf{vp}) = 1 \wedge \\ (f(t^*) \neq y \ \vee \ \mathsf{com}_f^* \neq \mathsf{CommitPoly}(f,r_f,\mathsf{pp}) \vee \\ (\pi^*,\mathsf{com}_y^*) \neq \mathsf{CommitValue}(f,t^*,y,r_f,r_y,\mathsf{pp})) \end{array}\right].$$

Our construction of zero-knowledge verifiable polynomial-delegation protocol is as following:

Construction 5 (Zero-knowledge Verifiable Polynomial-Delegation Protocol). *Let* $\mathbb{F}$ *be a prime-order finite field,* ℓ *be a variable parameter, and* d *be a variable-degree parameter such that* $O(\binom{\ell(d+1)}{\ell d})$ *is polynomial in* λ*. Consider the following protocol for the family* $\mathcal{F}$ *containing* ℓ*-variate polynomials of variable-degree* d *over* $\mathbb{F}$*.*

1. $\mathsf{KeyGen}(1^\lambda,\ell,d)$***:*** *Select* $\alpha,\beta,s_1,\ldots,s_\ell,s_{\ell+1} \in \mathbb{F}$ *uniformly at random, run* $\mathsf{bp} \leftarrow \mathsf{BilGen}(1^\lambda)$ *and compute* $\mathbb{P} = \{g^{\prod_{i\in W} s_i},\ g^{\alpha\cdot\prod_{i\in W} s_i}\}_{W\in\mathcal{W}_{\ell,d}}$*. The public parameters are set to be* $\mathsf{pp} = (\mathsf{bp},\mathbb{P},g^\alpha,g^\beta,g^{s_{\ell+1}},g^{\alpha s_{\ell+1}},g^{\beta s_{\ell+1}})$ *and the verifier parameters are set to be* $\mathsf{vp} = (\mathsf{bp},g^{s_1},\cdots,g^{s_\ell},g^{s_{\ell+1}},g^\alpha,g^\beta)$*.*

2. $\mathsf{CommitPoly}(f, r_f, \mathsf{pp})$: *If* $f \notin \mathcal{F}$ *output null. Else, compute* $c_1 = g^{f(s_i,\ldots,s_\ell)+r_f s_{\ell+1}}$ *and* $c_2 = g^{\alpha\cdot(f(s_i,\ldots,s_\ell)+r_f s_{\ell+1})}$*, and output the commitment* $\mathsf{com}_f = (c_1, c_2)$.

3. $\mathsf{CheckCom}(\mathsf{com}_f, \mathsf{vp})$: *On input a commitment* $\mathsf{com}_f = (\mathsf{com}_{f,1}, \mathsf{com}_{f,2})$*, check whether it is well-formed, i.e., output* 1 *if* $e(\mathsf{com}_{f,1}, g^\alpha) = e(\mathsf{com}_{f,2}, g)$ *and output* 0 *otherwise.*

4. $\mathsf{CommitValue}(f, t, y, r_f, r_y, \mathsf{pp})$: *Choose* $r_1, \ldots, r_\ell \in \mathbb{F}$ *uniformly at random. Next, using Lemma 1 compute polynomials* q_i *such that*

$$f(x_1, \ldots, x_\ell) + r_f x_{\ell+1} - (y + r_y x_{\ell+1}) =$$

$$\sum_{i=1}^{\ell} (x_i - t_i) \cdot (q_i(x_i, \ldots, x_\ell) + r_i x_{\ell+1}) + x_{\ell+1}(r_f - r_y - \sum_{i=1}^{\ell} r_i(x_i - t_i)).$$

Set $\mathsf{com}_y \leftarrow (g^{y+r_y s_{\ell+1}}, g^{\beta y+\beta r_y s_{\ell+1}})$*. For* $i = 1, \ldots, \ell$*, compute* $\mathsf{com}_i \leftarrow \mathsf{CommitPoly}(q_i, r_i, \mathsf{pp})$*. Compute* $\mathsf{com}_{\ell+1} \leftarrow (g^{r_f - r_y - \sum_{i=1}^{\ell} r_i(s_i - t_i)}, g^{\alpha(r_f - r_y - \sum_{i=1}^{\ell} r_i(s_i - t_i))})$*. Output* com_y *and the proof* $\pi := (\mathsf{com}_1, \ldots, \mathsf{com}_{\ell+1})$.

5. $\mathsf{Ver}(\mathsf{com}_f, t, \mathsf{com}_y, \pi, \mathsf{vp})$: *Parse the proof* π *as* $(\mathsf{com}_1, \ldots, \mathsf{com}_{\ell+1})$*. For* $i = 1, \ldots, \ell + 1$ *run* $\mathsf{CheckCom}(\mathsf{com}_i, \mathsf{pp})$*. If any of them outputs* 0*, output* 0*. Otherwise, parse* com_f *as* $(\mathsf{com}_{f,1}, \mathsf{com}_{f,2})$ *and* com_y *as* $(\mathsf{com}_{y,1}, \mathsf{com}_{y,2})$ *and for* $i = 1, \ldots, \ell + 1$ *parse* com_i *as* $(\mathsf{com}_{i,1}, \mathsf{com}_{i,2})$*. If* $e(\mathsf{com}_{y,1}, g^\beta) \stackrel{?}{=} e(\mathsf{com}_{y,2}, g)$ *and* $e(\mathsf{com}_{f,1}/\mathsf{com}_{y,1}, g) \stackrel{?}{=} e(g^{s_{\ell+1}}, \mathsf{com}_{\ell+1,1}) \prod_{i=1}^{\ell} e(g^{s_i - t_i}, \mathsf{com}_i)$ *output* 1*, otherwise output* 0.

Next, consider the following theorem.

Theorem 8. *Under Assumptions 1 and 2, Construction 5 is a zero-knowledge extractable verifiable polynomial-delegation protocol. Moreover, for a variable-degree-d ℓ-variate polynomial $f \in \mathcal{F}$ containing m monomials, algorithm* KeyGen *runs in time $O(\binom{\ell(d+1)-1}{\ell d})$,* CommitPoly *in time $O(m)$,* CommitValue *in time $O(\ell d m)$,* Ver *in time $O(\ell)$ and* CheckCom *in time $O(1)$. If $d = 1$,* CommitValue *can be made to run in time $O(2^\ell)$. The commitment produced by* CommitPoly *consists of $O(1)$ group elements, and the proof produced by* CommitPoly *consists of $O(\ell)$ elements of $\mathbb{G}$.*

Proof. Completeness follows by close inspection of the algorithms. Next, we prove the rest of the properties of Definition 6.

Polynomial extractability. Let Adv be a PPT adversary that on input $(1^\lambda, \mathsf{pp})$, where $(\mathsf{pp}, \mathsf{vp})$ is the output of $\mathsf{KeyGen}(1^\lambda, \ell, d)$, outputs commitment com_f^* such that $\mathsf{CheckCom}(\mathsf{com}_f^*, \mathsf{vp})$ accepts. This implies that $e(\mathsf{com}_{f,1}, g^\alpha) = e(\mathsf{com}_{f,2}, g)$. By Assumption 2, there exists PPT extractor $\mathcal{E}$ for Adv such that upon the same input as Adv, and with access to same random tape, outputs $a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|}, b \in \mathbb{F}$ such that $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_W \prod_{i \in W} s_i} g^{b s_{\ell+1}} = \mathsf{com}_{f,1}$, except with negligible probability. Note that, the coefficients $(a_0, \ldots, a_{|\mathcal{W}_{\ell,d}|}, b)$ can always be encoded as an $(\ell+1)$-variate polynomial that consist of the sum of two polynomials: an ℓ-variate one with degree-variable d that is defined over variables $x_1, \ldots, x_\ell$ and has values a_i as its monomial coefficients, and the univariate, degree-1 polynomial $b x_{\ell+1}$.

Binding. Next, we prove the binding property. Let Adv be a PPT adversary that wins the binding game with non-negligible probability. For $i = 1, \ldots, \ell+1$ we define adversary Adv_i that receives the same input as Adv and executes the same

code, but outputs only $\mathsf{com}_i \in \pi^*$ (where π^* is the proof output by Adv). Moreover, since Adv is PPT, all these adversaries are also PPT. Thus, for $i = 1, \ldots, \ell + 1$, from Assumption 2 there exists PPT $\mathcal{E}_i$ (running on the same random tape as Adv_i) which on input $(1^\lambda, \mathsf{pp})$ outputs $a_{0,i}, \ldots, a_{|\mathcal{W}_{\ell,d}|,i}, b_i \in \mathbb{F}$ such that the following holds: If $e(\mathsf{com}_{i,1}, g^\alpha) = e(\mathsf{com}_{i,2}, g)$ then $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_{W,i} \prod_{j \in W} s_j} \cdot g^{b_i s_{\ell+1}} \neq \mathsf{com}_{i,1}$, except with negligible probability. By the same reasoning as above, the coefficients $(a_{0,i}, \ldots, a_{|\mathcal{W}_{\ell,d}|,i} b_i)$ for each $i = 1, \ldots, \ell$ can always be encoded as an $(\ell+1)$-variate q'_i that can be expressed as the sum of an ℓ-variate polynomial with variable-degree d that is defined over variables $x_1, \ldots, x_\ell$ and a univariate degree-1 polynomial defined over $x_{\ell+1}$.

We now proceed to build an adversary $\mathcal{B}$ that breaks Assumption 1 for parameter $(\ell+1) \cdot d$. Upon input $(1^\lambda, \mathsf{bp}, g^s, g^{s^2}, \ldots, g^{s^{(\ell+1)\cdot d}})$, $\mathcal{B}$ proceeds as follows:

Parameter Generation. $\mathcal{B}$ implicitly sets $s_1 = s$ and for $i = 2, \ldots, \ell + 1$ he chooses $\rho_i \in \mathbb{F}$ uniformly at random and sets (also implicitly) $s_i = s \cdot \rho_i$. Then he chooses uniformly at random values $\alpha, \beta \in \mathbb{F}$. Next $\mathcal{B}$ needs to generate the terms in $\mathbb{P} = \{g^{\prod_{i \in W} s_i},\ g^{\alpha \cdot \prod_{i \in W} s_i}\}_{W \in \mathcal{W}_{\ell,d}}$. Since the exponent of each term is a product of at most $\ell \cdot d$ factors where each factor is one of the values $s_i = s \cdot r_i$ (for $i = 1, \ldots, \ell$), it can be written as a polynomial in s with degree at most $\ell \cdot d$. Therefore, $\mathcal{B}$ can compute these terms from the values $g, g^s, g^{s^2}, \ldots, g^{s^{\ell \cdot d}}$ and α. Then, he computes $g^{s_{\ell+1}}, g^{\alpha s_{\ell+1}}, g^{\beta s_{\ell+1}}$. Finally, $\mathcal{B}$ runs Adv on input $(1^\lambda, \mathsf{pp})$, where $\mathsf{pp} = (\mathsf{bp}, \mathbb{P}, g^\alpha, g^\beta, g^{s_{\ell+1}}, g^{\alpha s_{\ell+1}}, g^{\beta s_{\ell+1}})$.

Query Evaluation. Upon eventually receiving $(f^*, t^*, y^*, \pi^*, \mathsf{com}^*_f, \mathsf{com}^*_y, r^*_f, r^*_y)$ from Adv, $\mathcal{B}$ first checks whether $\mathsf{CommitPoly}(f^*, r^*_f, \mathsf{pp}) = \mathsf{com}^*_f$ and $\mathsf{CommitValue}(f^*, t^*,$

$r_y^*, \mathsf{pp}) = \mathsf{com}_f^*$ and aborts if any of the checks fails. Then, he runs $\mathsf{Ver}(\mathsf{com}_f^*, t^*, \mathsf{com}_y^*, \pi^*, \mathsf{vp})$ where $\mathsf{vp} = (\mathsf{bp}, g^{s_1}, \ldots, g^{s^{\ell+1}}, g^{\alpha}, g^{\beta})$. If Ver rejects $\mathcal{B}$ aborts, else he runs extractors $\mathcal{E}_1, \ldots, \mathcal{E}_{\ell+1}$ (defined above) on the same input as Adv and receives polynomials $q'_1, \ldots, q'_{\ell+1}$.

If for the output of any of the $\mathcal{E}_i$ it holds that $\prod_{W \in \mathcal{W}_{\ell,d}} g^{a_{W,i} \prod_{j \in W} s_j} g^{b_i s_{\ell+1}} \neq \mathsf{com}_{i,1}$, $\mathcal{B}$ aborts. Let $\delta = y^* - f^*(t^*)$. If $\delta = 0$ (i.e., $y^* = f^*(t^*)$), $\mathcal{B}$ aborts.

Otherwise, let $K(\mathbf{x}) \stackrel{\text{def}}{=} f^*(\mathbf{x}) - \sum_{i=1}^{\ell}(x_i - t_i)q'_i(\mathbf{x}) - x_{\ell+1}q'_{\ell+1}(\mathbf{x}) + (r_f - r_y)x_{\ell+1} - f^*(t^*)$. Note that by setting $s_1 = s, s_2 = \rho_2 \cdot s, \ldots, s_{\ell+1} = \rho_{\ell+1} \cdot s$, we implicitly set variables $x_2, \ldots, x_{\ell+1}$ to $\rho_2 \cdot x_1, \ldots, x_\ell = \rho_{\ell+1} \cdot x_1$. Thus, $K(\mathbf{x})$ can be interpreted as an (efficiently computable) univariate polynomial of degree at most $(\ell+1) \cdot d$ over variable x_1, which we refer to as $K'(x_1)$.

$\mathcal{B}$ then proceeds as follows. He chooses $\tau \in \mathbb{F}$ uniformly at random. If $g^{\tau} = g^{-s}$, he aborts. Else, he computes univariate polynomial Q of degree at most $(\ell+1) \cdot d$ and value $R \in \mathbb{F}$ such that $K'(x_1) = (x_1 + \tau)Q(x_1) + R$. We then distinguish two cases. (1) If $R = \delta$ then $\mathcal{B}$ factorizes the polynomial K' and let $Y \subset \mathbb{F}$ be the set of its roots ($|Y| \leq (\ell+1) \cdot d$). For each $y \in Y$, $\mathcal{B}$ tests whether $g^y = g^s$. If so, he outputs $(\tau, e(g,g)^{\frac{1}{y+\tau}})$ as a challenge tuple for Assumption 1 and halts. If all these checks fail, he aborts. (2) Else, (if $R \neq \delta$) he outputs $(\tau, e(g,g)^{Q(s_1) \cdot (\delta - R)^{-1}})$ as a challenge tuple for Assumption 1 and halts. Recall that, (as explained above) the expression in the exponent is a $(\ell+1) \cdot d$ degree polynomial thus the challenge value is computable in polynomial time from $(1^{\lambda}, p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, g^{s^2}, \ldots, g^{s^{(\ell+1) \cdot d}})$.

$\mathcal{B}$ is clearly PPT since all of $\mathcal{E}_i$ are PPT and he performs polynomially many operations in $\mathbb{F}, \mathbb{G}, \mathbb{G}_T$. Next, we analyze the success probability of $\mathcal{B}$. Recall that, by

assumption Adv succeeds in breaking the binding property of the scheme with non-negligible probability ϵ. We observe that, *conditioned on not aborting*, $\mathcal{B}$ perfectly emulates the binding game to $\mathcal{A}$ and moreover $\mathcal{B}$'s output is always a valid tuple for breaking Assumption 1. Let us argue why this is true.

Since verification succeeded, it holds that

$$e(\mathsf{com}_{f,1}/\mathsf{com}_{y,1}, g) = e(g^{s_{\ell+1}}, \mathsf{com}_{\ell+1,1}) \prod_{i=1}^{\ell} e(g^{s_i - t_i}, \mathsf{com}_{i,1})$$

and since extraction succeeded this can be replaced with

$$e(g,g)^{f^*(s_1,\ldots,s_\ell)+r_f x_{\ell+1}-y^*-r_y x_{\ell+1}} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})} \prod_{i=1}^{\ell} e(g,g)^{(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})}$$

$$e(g,g)^{f^*(s_1,\ldots,s_\ell)-y^*} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})} \prod_{i=1}^{\ell} e(g,g)^{(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})} e(g,g)^{(r_y-r_f)x_{\ell+1}}$$

$$e(g,g)^{f^*(s_1,\ldots,s_\ell)-y^*} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})+(r_y-r_f)x_{\ell+1}+\sum_{i=1}^{\ell}(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})}$$

$$e(g,g)^{-y^*} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})+(r_y-r_f)x_{\ell+1}+\sum_{i=1}^{\ell}(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})-f^*(s_1,\ldots,s_\ell)}$$

$$e(g,g)^{-\delta-f^*(t^*)} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})+(r_y-r_f)x_{\ell+1}+\sum_{i=1}^{\ell}(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})-f^*(s_1,\ldots,s_\ell)}$$

$$e(g,g)^{-\delta} = e(g,g)^{s_{\ell+1}q'_{\ell+1}(s_1,\ldots,s_{\ell+1})+(r_y-r_f)x_{\ell+1}+\sum_{i=1}^{\ell}(s_i-t_i)q'_i(s_1,\ldots,s_{\ell+1})-f^*(s_1,\ldots,s_\ell)+f^*(t^*)}$$

$$e(g,g)^{\delta} = e(g,g)^{K(s_1,\ldots,s_{\ell+1})}$$

$$e(g,g)^{\delta} = e(g,g)^{K'(s_1)} = e(g,g)^{(x_1+tau)Q(s_1)+R}$$

In order for the last substitution to be possible, it must the case that $K'(x_1)$, and correspondingly $K'(\mathbf{x})$ is non-constant polynomial (i.e., with degree > 0). Recall, that for polynomials defined over finite fields division is always possible assuming that the dividend's degree is at least as large as that of the divisor's. Moreover, the degree of the quotient is at most that of the dividend's and that of the remainder is strictly smaller than that of the divisor (i.e., R is a constant in this case).

Let us assume that $K'(\mathbf{x})$ is a constant polynomial. Since, $e(g,g)^{\delta} = e(g,g)^{K(s_1,\ldots,s_{\ell+1})}$ and $e(g,g)$ is a generator or $\mathbb{G}_T$, it must be that $K'(\mathbf{x}) \stackrel{\text{def}}{=} \delta$ therefore we can write

$$f^*(x_1,\ldots,x_\ell) - \delta - f^*(x_1,\ldots,x_\ell) =$$

$$x_{\ell+1}q'_{\ell+1}(x_1,\ldots,x_{\ell+1}) + (r_y - r_f)x_{\ell+1} + \sum_{i=1}^{\ell}(x_i - t_i)q'_i(x_1,\ldots,x_{\ell+1})$$

$$f^*(x_1,\ldots,x_\ell) - y^* =$$

$$x_{\ell+1}q'_{\ell+1}(x_1,\ldots,x_{\ell+1}) + (r_y - r_f)x_{\ell+1} + \sum_{i=1}^{\ell}(x_i - t_i)q'_i(x_1,\ldots,x_{\ell+1})$$

$$f^*(x_1,\ldots,x_\ell) - (r_y - r_f)x_{\ell+1} - y^* =$$

$$x_{\ell+1}q'_{\ell+1}(x_1,\ldots,x_{\ell+1}) + \sum_{i=1}^{\ell}(x_i - t_i)q'_i(x_1,\ldots,x_{\ell+1}).$$

Now let f' be the $\ell+1$ variable polynomial defined as $f'(x_1,\ldots,x_{\ell+1}) \stackrel{\text{def}}{=} f^*(x_1,\ldots,x_\ell) - (r_y - r_f)x_{\ell+1}$ and let $t' \in \mathbb{F}^{\ell+1}$ defined as $t' = (t_1^*,\ldots,t_\ell^*,0)$. From the above relation it follows that $f'(x_1,\ldots,x_{\ell+1}) - y^* = \sum_{i=1}^{\ell+1}(x_i - t'_i)q'_i(x_1,\ldots,x_{\ell+1})$, therefore t' is a root of the polynomial $f'' \stackrel{\text{def}}{=} f'(x_1,\ldots,x_{\ell+1}) - y^*$, i.e., $f''(t') = 0$ which implies that

$$f'(t_1,\ldots,t_{\ell+1}) - y^* = 0$$

$$f^*(t_1,\ldots,t_\ell) - (r_y - r_f)\cdot 0 - y^* = 0$$

$$f^*(t_1,\ldots,t_\ell) = y^*$$

which implies that y^* is the correct evaluation of f^* on t^*, i.e., $\delta = 0$. If that is the case, $\mathcal{B}$ has already aborted, therefore conditioned on not aborting this will never happen.

In all other cases, the polynomial division is possible therefore we can write

$$e(g,g)^{\delta} = e(g,g)^{(s_1+\tau)Q(s_1)+R}$$

$$e(g,g)^{\frac{\delta}{s_1+\tau}} = e(g,g)^{Q(s_1)+\frac{R}{s_1+\tau}}$$

$$e(g,g)^{\frac{\delta-R}{s_1+\tau}} = e(g,g)^{Q(s_1)}.$$

If $\delta = R$ (case (1) above), then it follows that $e(g,g)^0 = e(g,g)^{Q(s_1)}$, i.e., $s_1 = s$ is root of Q. Therefore, $s = y$ for some $y \in Y$ (and therefore $|Y| > 0$). Since factorization can be done in deterministic polynomial time $\mathcal{B}$ always succeeds in computing this y and $e(g,g)^{\frac{1}{y+\tau}} = e(g,g)^{\frac{1}{s+\tau}}$ thus $\mathcal{B}$ succeeds in breaking Assumption 1 in this case. If $\delta \neq R$ (case (2) above), from the above it holds that

$$e(g,g)^{\frac{\delta-R}{s_1+\tau}} = e(g,g)^{Q(s_1)}$$

$$e(g,g)^{\frac{1}{s_1+\tau}} = e(g,g)^{Q(s_1)\cdot(\delta-R)^{-1}}$$

therefore, in this case too, $\mathcal{B}$ succeeds in breaking Assumption 1 in this case.

Since the two cases are complementary, $\mathcal{B}$ always succeeds, conditioned on not aborting. Thus, it remains to bound the probability of aborting. $\mathcal{B}$ can only abort in three cases. If extraction fails, if $y^* = f^*(t^*)$, or if $\tau = -y$. The former can only happen with negligible probability. This holds since, if verification succeeds it must be that $e(\mathsf{com}_{i,2}, g) = e(\mathsf{com}_{i,1}, g^{\alpha})$ for $i = 1, \ldots, \ell+1$ and by Assumption 2, extraction for any of $\mathcal{E}_1, \ldots, \mathcal{E}_{\ell+1}$ fails with negligible probability. Since ℓ is polynomial in λ it follows that the probability any of them fails (which by a union bound is at most equal to the sum of each individual failure probability) is also negligible. The second happens by assumption with probability at most $1 - \epsilon$ (as Adv wins with

probability at least ϵ), whereas the third happens with negligible probability $O(2^{-\lambda})$ as τ is chosen uniformly at random from $\mathbb{F}$. By a union bound, the abort probability is at most $(1-\epsilon)+\mathsf{neg}(\lambda)$. Thus the success probability of $\mathcal{B}$ is $\epsilon - \mathsf{neg}(\lambda)$ which is non-negligible as we assumed that ϵ is non-negligible. Since $\mathcal{B}$ succeeds in breaking Assumption 1 this contradicts our original assumption and our proof is complete.

Evaluation extractability. This follows almost directly from soundness and polynomial extractability. In particular, let Adv be an PPT adversary that plays the evaluation extractability game. Let Adv_f, Adv_y be two adversaries that on input the same input as Adv, run Adv's code internally but only output com_f^*, com_y^* respectively and then halt. Clearly, both adversaries are PPT. Moreover, whenever $\mathsf{CheckCom}(\mathsf{com}_f^*, \mathsf{vp})$ and $\mathsf{Ver}(\mathsf{com}_f^*, t^*, \mathsf{com}_y^*, \pi^*, \mathsf{vp})$ output 1, it follows that: (1) by polynomial extractability there exist extractor $\mathcal{E}_f$ with access to the code and random tape of Adv_f that with all but negligible probability outputs f, r_f such that $\mathsf{CommitPoly}(f, r_f, \mathsf{pp}) = \mathsf{com}_f^*$, and (2) by Assumption 2, since Ver accepted (and recall that as a sub-routine, Ver checks that $e(\mathsf{com}_{y,1}, g^{\beta}) = e(\mathsf{com}_{y,2}, g)$) there exists PPT extractor with access to the code and random tape of Adv_y that with all but negligible probability, outputs $y, r_y \in \mathbb{F}$ such that $g^{y + r_y s_{\ell+1}} = \mathsf{com}_{y,1}$.

It remains to show that the event $E = \{f(t^*) \neq y$, where f is the output of $\mathcal{E}_f$ and y is the output of $\mathcal{E}_y\}$ occurs with negligible probability. For contradiction, assume $\Pr[E] = \epsilon$, for some non-negligible ϵ. Then we can build adversary Adv' that breaks the binding property of our scheme, as follows.

1. On input $(1^\lambda, \mathsf{pp})$, Adv' runs Adv internally and receives $(t^*, \pi^*, \mathsf{com}_f^*, \mathsf{com}_y^*)$.

2. Adv' runs $\mathcal{E}_f, \mathcal{E}_y$ on the same input as Adv to receive f, r_f, y, r_y.

3. Adv' outputs $(f, t^*, y, \pi^*, \mathsf{com}_f^*, \mathsf{com}_y^*, r_f, r_y)$ as a challenge for the soundness game.

Adv' is clearly PPT as $\mathsf{Adv}, \mathcal{E}_f, \mathcal{E}_y$ are all PPT. Note that whenever E occurs, Adv' wins. Assuming that $\Pr[E] = \epsilon$, it follows that Adv' breaks the binding property, for which we proved above that it can only happen with negligible probability. This concludes our proof.

Zero knowledge. We build our simulator Sim that operates as follows.

1. On input $(1^\lambda, \ell, d)$, run $\mathsf{KeyGen}(1^\lambda, \ell, d)$ and receive pp, vp. Set $\sigma = \alpha, \beta, s_1, \ldots, s_{\ell+1}$. Choose $r_f \in \mathbb{F}$ uniformly at random and set $\mathsf{com}_f = (g^{r_f}, g^{\alpha \cdot r_f})$. Send $\mathsf{vp}, \mathsf{com}_f$ to $\mathcal{A}$.

2. Receive k from $\mathcal{A}$.

3. For $i = 1, \ldots, k$ repeat:

   (a) Receive t_i from $\mathcal{A}$.

   (b) Choose $r_{1i}, \ldots, r_{\ell i}, r_{yi} \in F$ uniformly at random.

   (c) Compute $\mathsf{com}_{yi} = (g^{r_{yi} s_{\ell+1}}, g^{\beta r_{yi} s_{\ell+1}})$, $\mathsf{com}_{ji} = (g^{r_{ji} s_{\ell+1}}, g^{\alpha r_{ji} s_{\ell+1}})$ for $j = 1, \ldots, \ell$ and $\mathsf{com}_{\ell+1 i} = (g^{r_f - r_{yi} - \sum_{j=1}^{\ell} r_{ji}(s_j - t_{ij})}, g^{\alpha(r_f - r_{yi} - \sum_{j=1}^{\ell} r_{ji}(s_j - t_{ij}))})$.

   (d) Output $(\mathsf{com}_{yi}, \pi_i = (\mathsf{com}_{1i}, \ldots, \mathsf{com}_{\ell+1 i}), \sigma)$.

Sim is clearly PPT as all the above steps can be computed in time polynomial in λ. Next, note that since r_f and $r_{1i}, \ldots, r_{\ell i}$, for all i, are chosen uniformly at random, it follows that $\mathsf{com}_f, \mathsf{com}_{1i}, \ldots, \mathsf{com}_{\ell i}$ are indistinguishable from uniformly

chosen elements from $\mathbb{G}$. Moreover, this holds both in the real and the ideal game execution since in the former the discrete logs of these elements are computed as the sum of a polynomial evaluation in $\mathbb{F}$ and an element of $\mathbb{F}$ chosen uniformly at random. Finally, note that in both games, for any i, fixing $\mathsf{com}_f, \mathsf{com}_{1i}, \ldots, \mathsf{com}_{\ell i}$ also fixes a unique element $\mathsf{com}_{\ell+1 i} \in \mathbb{G}$. From the above, it follows that for any (even unbounded) adversary $\mathcal{A}$ and all $f \in \mathbb{F}$, it holds that the view from the execution of $\mathsf{Real}_{\mathsf{Adv},f}(1^\lambda)$ and $\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$ is indistinguishable, thus Construction 5 is perfect zero-knowledge.

□

6.3 Zero-Knowledge CMT Protocol

We start with a zero-knowledge sum-check protocol, which is a major building block in the CMT protocol.

6.3.1 A Sum-Check Protocol over Homomorphic Commitments

As shown in Section 2.3.1, the messages exchanged during the sum check protocol reveal the coefficients of $g_1, \cdots, g_\ell$, thus leaking additional information about the values of g, beyond H. This is problematic since we would like to use the sum-check protocol as part of a zero-knowledge argument system of NP, where leaking additional evaluations of g might leak information about the prover's witness. To that end, we execute the sum-check protocol over an additively homomorphic commitment scheme. In particular, we consider the Pedersen commitment scheme,

modified as described in Section 6.1. The modified protocol starts by having $\mathcal{P}$ commit to the coefficients of g and by providing a commitment com_0 to the value H. Next, at round i, instead of having $\mathcal{P}$ send to $\mathcal{V}$ the coefficients of g_i, we modify the sum-check protocol and have $\mathcal{P}$ send commitments to these coefficients to $\mathcal{V}$. Since Pedersen commitments are linearly homomorphic, $\mathcal{V}$ can locally compute a commitment com^* to $g_i(0)+g_i(1)$ and then check (using the $\mathcal{ZK}_{eq}$ protocol) that com^* commits to the same value as com_{i-1}, thus verifying that indeed $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$. In case this verification succeeds, $\mathcal{V}$ sends a uniformly random challenge r_i to $\mathcal{P}$, and both $\mathcal{P}$ and $\mathcal{V}$ use the homomorphic properties of the Pedersen commitment scheme in order to obtain a commitment com_i to the evaluation of g_i on r_i. $\mathcal{P}$ and $\mathcal{V}$ repeat the above, using com_i and r_i in round $i+1$.

Formally, we present our construction below.

Construction 6 (**Sum-check Protocol Over Homomorphic Commitment Schemes**). *Let $\mathbb{F}$ be a prime-order finite field, and let λ be a security parameter. In addition, let* Comm *be a linearly homomorphic commitment scheme as described in Section 6.1,* $\mathsf{cp} \leftarrow \mathsf{Setup}(1^\lambda)$*, and let $g(b_1, \cdots, b_\ell)$ be an ℓ-variate total-degree-d polynomial over $\mathbb{F}$ which is represented using m coefficients $a_0, \cdots, a_m$. Consider the following protocol between $\mathcal{P}$ and $\mathcal{V}$ for convincing $\mathcal{V}$ that $t_0 = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell)$ is a valid opening to some commitment com_0. That is, that he knows ρ_0 such that $\mathsf{com}_0 = \mathsf{Com}(\mathsf{cp}, \mathsf{t}_0, \rho_0)$.*

1. *For all $i = 1, \cdots, \ell$ perform the following.*

   (a) *Define $g_i(x) = \sum_{b_{i+1}, \cdots, b_\ell \in \{0,1\}} g(r_1, \cdots, r_{i-1}, x, b_{i+1}, \cdots, b_\ell)$ and let $a_0, \cdots, a_m$*

be the coefficients of g_i.

(b) *For every* $0 \leq j \leq m$ $\mathcal{P}$ *computes* $\mathsf{com}_{a_j} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{a_j}, \rho_{\mathsf{a_j}})$ *where* $\rho_{a_j} \in \mathbb{F}$ *is selected uniformly at random and sends* $(\mathsf{com}_{a_0}, \cdots, \mathsf{com}_{a_m})$ *to* $\mathcal{V}$.

(c) $\mathcal{V}$ *computes* $\mathsf{com}^*_{i-1} \leftarrow \mathsf{com}_{a_0} \cdot \prod_{k=0}^{m} \mathsf{com}_{a_k}$ *which is a commitment to* $g_i(0) + g_i(1)$.

(d) $\mathcal{V}$ *and* $\mathcal{P}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, t_{i-1}, \rho_{i-1}, \rho_{a_0} + \sum_{j=0}^{m} \rho_{a_j}; \mathsf{com}_{i-1}, \mathsf{com}^*_{i-1})$.

(e) $\mathcal{V}$ *generates a random value* r_i *and sends it to* $\mathcal{P}$.

(f) *Both* $\mathcal{V}$ *and* $\mathcal{P}$ *compute* $\mathsf{com}_i \leftarrow \mathsf{Evaluate}(\mathsf{cp}, \mathsf{com}_{a_0}, \cdots, \mathsf{com}_{a_m}, 1, r_i, \cdots, r_i^m)$.

(g) $\mathcal{P}$ *sets* $t_i \leftarrow g_i(r_i)$ *and* $\rho_i \leftarrow \sum_{j=0}^{m} \rho_{a_j} r_i^j$

2. $\mathcal{V}$ *computes* $\mathsf{com}^*_\ell \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{g}(\mathsf{r}_1, \cdots, \mathsf{r}_\ell), \rho_{\ell+1})$ *and sends* com^*_ℓ *and* $\rho_{\ell+1}$ *to* $\mathcal{P}$.

3. *Both* $\mathcal{V}$ *and* $\mathcal{P}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, g(r_1, \cdots, r_\ell), \rho_\ell, \rho_{\ell+1}; \mathsf{com}_\ell, \mathsf{com}^*_\ell)$.

We now state the following theorem (we are only interested in proving "regular" soundness and not knowledge soundness).

Theorem 9. *For any* ℓ*-variate, total-degree-*d *polynomial* $g : \mathbb{F}^\ell \to \mathbb{F}$ *with* m *non-zero coefficients, assuming* Comm *is a linearly homomorphic commitment scheme, as described in Section 6.1, and* $\mathcal{ZK}_{eq}$ *is a zero-knowledge non-interactive argument for testing equality of commitments for* Comm*, Construction 6 is an interactive argument with soundness* $d \cdot \ell / |\mathbb{F}|$ *for the following language* L

$$\left\{ (\mathsf{cp}, \mathsf{com}_0, g; \rho_0) : \mathsf{com}_0 \leftarrow \mathsf{Com}\left(\mathsf{cp}, \sum_{\mathsf{b}_1, \cdots, \mathsf{b}_\ell \in \{0,1\}} \mathsf{g}(\mathsf{b}_1, \cdots, \mathsf{b}_\ell), \rho_0\right) \text{ and } \mathsf{cp} \leftarrow \mathsf{Setup}(1^\lambda) \right\}$$

where λ *is the security parameter.*

Proof Sketch. The completeness property immediately follows from Construction 6. We now proceed to argue about the soundness property.

Soundness. Let $g(b_1, \cdots, b_\ell)$ be an ℓ-variate total-degree-d polynomial over a finite field $\mathbb{F}$. We begin by observing that the commitment com_0 and all coefficient commitments $\mathsf{com}_{a_j,i}$ for $i = 1, \ldots, \ell$, $j = 1, \ldots, m$ are extractable. That is, for each of them there exists a polynomial-time extractor that receives the same input as the adversary Adv and outputs with all but negligible probability a valid pre-image from $\mathbb{F}$, whenever Adv succeeds in convincing $\mathcal{V}$. This follows under Assumption 2 using the same argument as in the proof of Theorem 8 (recall that, as explained in Section 6.1, we implicitly assume that whenever $\mathcal{V}$ receives a commitment he checks whether it is well-formed and rejects otherwise).

Next, we distinguish between the following two complementary cases.

1. **There exists $0 \leq i \leq \ell$ such that the extracted pre-images for $\mathsf{com}_i, \mathsf{com}_i^*$ are not equal.** In this case, this directly contradicts the soundness of the $\mathcal{ZK}_{eq}$ protocol executed in Steps 1d and 3 of Construction 6. This means that Adv can be used to construct a black box adversary Adv' which breaks the soundness of the $\mathcal{ZK}_{eq}$ protocol.

2. **For all $0 \leq i \leq \ell$ it holds that the extracted pre-images for $\mathsf{com}_i, \mathsf{com}_i^*$ are equal.** In this case let t_0^* be the extracted pre-image of com_0, and let h_i^* be the extracted pre-image of com_i for all $i = 0, \cdots, \ell - 1$. Also, let g_i^* be the polynomial defined by coefficients $a_0, \cdots, a_m$ which are the pre-images extracted from the commitments $\mathsf{com}_{a_0}, \cdots, \mathsf{com}_{a_m}$ sent by $\mathcal{P}$ in Step 1b of Construction 6

during the $(i+1)$th round. Notice that since com_i and com_i^* have the same pre-image h^*, by construction of com_i^* is holds that $h_i^* = g_i^*(0) + g_i^*(1)$. Due to this, notice that $(t_0^*, g, (r_i, g_i^*, h_i^*)_{i=1,\cdots,\ell})$ is a valid transcript for a (possibly) cheating prover $\mathcal{P}^*$ controlled by Adv trying to convince a verifier $\mathcal{V}$ that indeed $t_0 = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell)$.

Thus, if $t_0 \neq \sum_{b_1 \in \{0,1\}, \ldots, b_\ell \{0,1\}} g(b_1 \ldots, b_\ell)$ then Adv can be used in a black-box manner in order to break the soundness property of the sum-check interactive proof protocol.

□

Moreover, we prove the following lemma that will be helpful for us while proving the zero-knowledge property of our argument.

Lemma 2. *For every verifier $\mathcal{V}^*$ and for every ℓ-variate, total-degree-d polynomial $g : \mathbb{F}^\ell \to \mathbb{F}$ with m non-zero coefficients, there exists a simulator Sim such that Sim is capable of simulating from $\mathsf{cp}, \mathsf{com}_0, m$ and t_0 (without using g) the partial view of $\mathcal{V}^*$ defined by $\mathsf{cp}, \mathsf{com}_0$ as well as the messages obtained during only Step 1 of Construction 6.*[1]

Proof Sketch. We build simulator Sim which simulates the view of $\mathcal{V}$ during Step 1 of Construction 6 as follows. First, Sim receives as input commitment parameters cp, commitment com_0, an upper bound m on the number of coefficients of g, as well as the value $t_0 = \sum_{b_1, \cdots, b_\ell \in \{0,1\}} g(b_1, \cdots, b_\ell)$. Sim proceeds to simulate Step 1 of Construction 6 as follows.

[1]Notice that the partial view does not include the coefficients $a_1, \cdots, a_m$ of g.

1. For $i = 1, \ldots, \ell$:

   (a) Sim chooses coefficients $a_0, \ldots a_{m-1}$ chosen uniformly at random from $\mathbb{F}$ and sets a_m such that $t_{i-1} = a_0 + a_m + \sum_{j=0}^{m-1} a_j$. Let $g_i(x)$ be the polynomial denoted by coefficients $a_0, \ldots, a_m$ and notice that $t_{i-1} = g_i(0) + g_i(1)$.

   (b) For every $0 \leq j \leq m$, Sim computes $\mathsf{com}_{a_j} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{a_j}, \rho_{\mathsf{a_j}})$ where $\rho_{a_j} \in \mathbb{F}$ is selected uniformly at random, and sends $(\mathsf{com}_{a_0}, \cdots, \mathsf{com}_{a_m})$ to $\mathcal{V}^*$.

   (c) Sim computes $\mathsf{com}^*_{i-1} \leftarrow \mathsf{com}_{a_0} \cdot \prod_{k=0}^{m} \mathsf{com}_{a_k}$.

   (d) Let Sim_{eq} be the simulator guaranteed from the zero knowledge property of $\mathcal{ZK}_{eq}$. Sim runs simulator Sim_{eq} on inputs $(\mathsf{cp}, \mathsf{com}^*_{i-1}, \mathsf{com}_{i-1})$ in order to simulate $\mathcal{V}^*$'s view during the execution of $\mathcal{ZK}_{eq}$ the ith round.

   (e) Upon receiving r_i from $\mathcal{V}^*$, Sim computes $\mathsf{com}_i \leftarrow \mathsf{Evaluate}(\mathsf{cp}, \mathsf{com}_{a_0}, \cdots, \mathsf{com}_{a_m}, 1, r_i, \cdots, r_i^m)$ and sets $t_i \leftarrow g_i(r_i)$ and $\rho_i \leftarrow \sum_{j=0}^{m} \rho_{a_j} r_i^j$.

The produced transcript is indistinguishable from the one $\mathcal{V}^*$ gets while interacting with $\mathcal{P}$ since: (i) the coefficients $a_0, \ldots, a_m$ for each round i satisfy the same relation with respect to t_{i-1} in both cases, (ii) Comm is statistically hiding, i.e., each commitment is indistinguishable from a commitment to a random value, and (iii) the output of Sim_{eq} for round i is indistinguishable from the messages received by $\mathcal{V}^*$ while running $\mathcal{ZK}_{eq}$ on the same values. In the following, we consider a slightly modified simulator Sim that outputs as secret state the values (t_ℓ, ρ_ℓ) to be used when building a larger simulator that runs Sim as a black box. $\square$

6.3.2 A CMT Protocol over Homomorphic Commitments

Similarly to the case of the standard sum-check protocol, the messages exchanged during the CMT execution leak information about the intermediate values of the circuit C and thus potentially about the circuit's input (which in our case will include the prover's witness). Thus, similarly to Section 6.3.1, we execute the CMT protocol over homomorphic commitments and use the commitment's hiding property to conceal from $\mathcal{V}$ information regarding the circuit's internal wires.

The modified protocol proceed as follows. First, $\mathcal{P}$ sends to $\mathcal{V}$ commitments $\mathsf{com}_{x_1}, \ldots, \mathsf{com}_{x_n}$ to the n inputs of C and a commitment com_0 to 1 (the claimed output of the circuit when evaluated on x). Next, at round-i, both $\mathcal{P}$ and $\mathcal{V}$ use the first step of the sum-check protocol over homomorphic commitments, resulting in $\mathcal{V}$ obtaining a commitment com'_i on a_i which is claimed by $\mathcal{P}$ to be equal to $\tilde{V}_{i-1}(r'_i)$, where r'_i is uniformly generated by $\mathcal{V}$. Let q_1, q_2 be the last $2s_i$ elements of r'_i. $\mathcal{P}$ then computes $t_1 = \tilde{V}_i(q_1)$, $t_2 = \tilde{V}_i(q_2)$, $t_3 = t_1 \cdot t_2$ and commits to t_1, t_2, t_3 resulting in $\mathsf{com}_{t_1}, \mathsf{com}_{t_2}, \mathsf{com}_{t_3}$. $\mathcal{V}$ then verifies (using the $\mathcal{ZK}_{prod}$ protocol) that indeed com_{t_3} is a commitment to the multiplication of t_1 and t_2 and uses the homomorphic properties of the commitment scheme and the $\mathcal{ZK}_{eq}$ protocol in order to check that the value com'_i provided earlier by $\mathcal{P}$ is indeed a commitment to the evaluation of $\tilde{V}_{i-1}(r'_i)$. Both $\mathcal{V}$ and $\mathcal{P}$ then use the homomorphic properties of the commitment scheme and the $\mathcal{ZK}_{eq}$ protocol in order for $\mathcal{V}$ to obtain a commitment com_i to a value $a_i = \tilde{V}_i(\gamma(r''_i))$, where r''_i is generated by $\mathcal{V}$, and proceed to the next round of the protocol using a_i and random point $r_i = \gamma(r''_i)$. Finally $\mathcal{P}$ reveals x and

r_0 to $\mathcal{V}$ who then checks their consistency with the initial commitments, evaluates the polynomial $\tilde{V}_x$ on the last random point established r_d and both parties use $\mathcal{ZK}_{eq}$ to establish that a commitment to this evaluation has the same pre-image as com_d. We stress that this entire last step (which clearly would violate any notion of zero-knowledge) will not be a step of our final construction; it will instead by replaced with appropriate evocations of zk-VPD.

Formally, consider the protocol presented below.

Construction 7 (**CMT Protocol Over Homomorphic Commitment Schemes**)**.** *Let $\mathbb{F}$ be a prime-order finite field, and let λ be a security parameter,* Comm *be a linearly homomorphic commitment scheme as described in Section 6.1, and let* $\mathsf{cp} \leftarrow \mathsf{Setup}(1^\lambda)$*. In addition, let $C : \mathbb{F}^n \to \mathbb{F}$ be a depth-d layered arithmetic circuit and let $x \in \mathbb{F}^n$ be inputs of C such that $C(x) = 1$. Consider the following protocol between $\mathcal{P}$ and $\mathcal{V}$ for convincing $\mathcal{V}$ that x is a valid opening to a series of commitments* $\mathsf{com}_{x_i} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{x_i}, \rho_{\mathsf{x_i}})$*, where $x_i \in \mathbb{F}$ is the i-th element of input x.*

1. *Both parties set $a_0 = 1$ and $r_0 = 0$. $\mathcal{P}$ generates ρ_0 uniformly at random, computes* $\mathsf{com}_0 \leftarrow \mathsf{Comm}(\mathsf{cp}, \mathsf{a_0}, \rho_0)$ *and sends it to $\mathcal{V}$.*

2. *For all $i = 1, \cdots, d$ perform the following.*

   (a) *$\mathcal{V}$ and $\mathcal{P}$ execute Step 1 of Construction 6 on input a_{i-1}, polynomial $\tilde{V}_{i-1}$ (as per Equation 2.1), and randomness ρ_{i-1} for $\mathcal{P}$ and $\mathsf{com}_{i-1}, r_{i-1}$ for $\mathcal{V}$. As a result, $\mathcal{V}$ obtains a commitment $\mathsf{com}'_i = \mathsf{Com}(\mathsf{t}, \rho_\mathsf{i})$ (where t and ρ_i are known to $\mathcal{P}$ and not to $\mathcal{V}$) where $\mathcal{P}$ claims that $t = \tilde{V}_{i-1}(r'_i)$ with r'_i having been selected uniformly at random by $\mathcal{V}$.*

(b) *Let* (q_1, q_2) *be the last* $2s_i$ *elements of* r'_i. $\mathcal{P}$ *computes* $t_1 = \tilde{V}_i(q_1)$, $t_2 = \tilde{V}_i(q_2)$, *and* $t_3 = t_1 \cdot t_2$. $\mathcal{P}$ *then computes commitments* $\mathsf{com}_{t_1} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{t_1}, \rho_{\mathsf{t_1}})$, $\mathsf{com}_{t_2} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{t_2}, \rho_{\mathsf{t_2}})$, *and* $\mathsf{com}_{t_3} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{t_3}, \rho_{\mathsf{t_3}})$ *which he sends to* $\mathcal{V}$.

(c) $\mathcal{V}$ *and* $\mathcal{P}$ *perform* $\mathcal{ZK}_{prod}(\mathsf{cp}, t_1, t_2, t_3, \rho_{t_1}, \rho_{t_2}, \rho_{t_3}; \mathsf{com}_{t_1}, \mathsf{com}_{t_2}, \mathsf{com}_{t_3})$.

(d) *Using Equation 2.1* $\mathcal{V}$ *can express* $\tilde{V}_{i-1}(r'_i)$ *as a linear function of* $r'_i, \tilde{V}_i(q_1)$, $\tilde{V}_i(q_2), \tilde{V}_i(q_1) \cdot \tilde{V}_i(q_2)$. *Thus, using* Evaluate, $\mathcal{V}$ *can obtain a new commitment* com^*_i *to the evaluation of* $\tilde{V}_{i-1}(r'_i)$ *and let* ρ^*_i *be the corresponding randomness.*

(e) $\mathcal{V}$ *and* $\mathcal{P}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, \tilde{V}_{i-1}(r'_i), \rho_i, \rho^*_i; \mathsf{com}'_i, \mathsf{com}^*_i)$.

(f) *Let* $\gamma : \mathbb{F} \to \mathbb{F}^{s_i}$ *be the line defined by* $\gamma(0) = q_1$ *and* $\gamma(1) = q_2$ *and let* $h(x)$ *be the degree-*s_i *polynomial such that* $h(x) = \tilde{V}_i(\gamma(x))$ *and* $h_0, \ldots, h_{s_i}$ *be its coefficients. For* $j = 0, \ldots, s_i$, $\mathcal{P}$ *computes commitments* $\mathsf{com}_{h_j} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{h_j}, \rho_{\mathsf{h_j}})$ *and sends them to* $\mathcal{V}$.

(g) $\mathcal{V}$ *computes* $\mathsf{com}_{h(0)} \leftarrow \mathsf{com}_{h_0}$ *and* $\mathsf{com}_{h(1)} \leftarrow \prod_{j=0}^{s_i} \mathsf{com}_{h_j}$ *which are commitments to* $h(0)$ *and* $h(1)$ *respectively.*

(h) $\mathcal{V}$ *and* $\mathcal{P}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, t_1, \rho_{t_1}, \rho_{h_0}; \mathsf{com}_{t_1}, \mathsf{com}_{h(0)})$ *and* $\mathcal{ZK}_{eq}(\mathsf{cp}, t_2, \rho_{t_2}, \sum_{j=0}^{s_i} \rho_{h_j}; \mathsf{com}_{t_2}, \mathsf{com}_{h(1)})$.

(i) $\mathcal{V}$ *chooses* $r''_i \in \mathbb{F}$ *uniformly at random, sets* $r_i \leftarrow \gamma(r''_i)$ *and sets* $\mathsf{com}_i \leftarrow \mathsf{Evaluate}(\mathsf{cp}, \mathsf{com}_{h_0}, \cdots, \mathsf{com}_{h_{s_i}}, 1, r''_i, \cdots, r''^{s_i})$.

(j) $\mathcal{V}$ *sends* r''_i *and* com_i *to* $\mathcal{P}$. $\mathcal{P}$ *sets* $r_i \leftarrow \gamma(r''_i)$, $a_i \leftarrow \tilde{V}_i(\gamma(r''_i))$ *and* $\rho_i \leftarrow \sum_{j=0}^{s_i} r''^j \rho_{h_j}$.

3. *$\mathcal{P}$ sends to $\mathcal{V}$ the input x and randomness ρ_0 and ρ_{x_i} for $1 \leq i \leq n$. $\mathcal{V}$ verifies that $\mathsf{com}_{x_i} = \mathsf{Com}(\mathsf{cp}, \mathsf{x_i}, \rho_{\mathsf{x_i}})$ for $1 \leq i \leq n$ and that $\mathsf{com}_0 = \mathsf{Com}(\mathsf{cp}, 1, \rho_0)$.*

4. *Let $\tilde{V}_x$ be the multilinear extension of the polynomial V_x satisfying $V_x(i) = x_i$ for all $i = 1, \cdots, n$. The verifier computes $\mathsf{com}^*_x \leftarrow \mathsf{Com}(\mathsf{cp}, \tilde{\mathsf{V}}_{\mathsf{x}}(\mathsf{r_d}), \rho^*_{\mathsf{d}})$ where ρ^*_d is chosen uniformly at random.*

5. *$\mathcal{V}$ and $\mathcal{P}$ perform $\mathcal{ZK}_{eq}(\mathsf{cp}, \tilde{V}_x(r_d), \rho_d, \rho_x; \mathsf{com}^*_x, \mathsf{com}_d)$. $\mathcal{V}$ accepts if the protocol accepts and rejects otherwise.*

We have the following theorem:

Theorem 10. *Let $C : \mathbb{F}^n \to \mathbb{F}^k$ be a depth-d layered arithmetic circuit over a finite field $\mathbb{F}$. Assuming* Comm *is an linearly homomorphic commitment scheme as described in Section 6.1, $\mathcal{ZK}_{eq}$ is a zero-knowledge argument for testing equality of committed values, and $\mathcal{ZK}_{prod}$ is a zero-knowledge argument for testing the product relation between three commitments in* Comm*, the CMT protocol presented in Construction 7 is an interactive argument with soundness $O(d \cdot \log S/|\mathbb{F}|)$ for for the following relation*

$$\mathcal{R} = \Bigg\{ \Big((\mathsf{cp}, C, \mathsf{com}_{x_1}, \ldots, \mathsf{com}_{x_n}); (x_1, \cdots, x_n, \rho_{x_1}, \ldots, \rho_{x_n}) \Big) : C(x_1, \cdots, x_n) = 1 \wedge \bigwedge_{i=1}^{n} \mathsf{com}_{x_i} = \mathsf{Com}(\mathsf{cp}, \mathsf{x_i}, \rho_{\mathsf{x_i}}) \Bigg\}$$

where $\mathsf{cp} \leftarrow \mathsf{Setup}(1^\lambda)$, λ is the security parameter, n is the input size of C, and S is the maximal number of gates per circuit layer in C.

Proof Sketch. The completeness property immediately follows from Construction 7. We now proceed to argue about the soundness property.

Soundness. Soundness follows by a similar argument as in Theorem 9. Indeed, let $\mathcal{P}^*$ be an cheating prover which convinces $\mathcal{V}$ (with non-negligible probability) of a claim "$1 = C(x)$" for some x and C, such that $1 \neq C(x)$. Using Assumption 2, for each commitment that $\mathcal{V}$ receives from $\mathcal{P}^*$, there exists a polynomial-time extractor with access to $\mathcal{P}^*$'s code and random tape that outputs (with all but negligible probability) a corresponding commitment pre-image.

Next, we define the following events.

1. Event A takes place if the extracted pre-image for com_d is not equal to $\tilde{V}_x(r_d)$ or there exists $1 \leq i \leq d$ such that the extracted pre-images for $\mathsf{com}'_i, \mathsf{com}^*_i$ during Step 2e, or the extracted pre-images for $\mathsf{com}_{t_1}, \mathsf{com}_{h(0)}$ or $\mathsf{com}_{t_2}, \mathsf{com}_{h(1)}$ during Step 2h are not equal.

2. Event B takes place if there exists $1 \leq i \leq d$ such that the extracted pre-image for com_{t_3} is not equal to the product of the extracted pre-images for $\mathsf{com}_{t_1}, \mathsf{com}_{t_2}$ during Step 2c

3. Event C_i (for $0 \leq i \leq d$) takes place if $\tilde{V}_i(r_i) = a_i$ (where $\tilde{V}_i$ is as defined in Equation 2.1) when evaluating C on the extracted pre-image of com_x and a_i is the extracted pre-image of com_i.

Note that assuming $C(x) \neq 1$ is equivalent to assuming $\tilde{V}_0(r_0) \neq a_0$ i.e., that $\neg C_0$ occurred. Next, we study the following (exhaustive) cases.

- **Event A occurs.** We argue about this case in exactly the same manner as in the proof of Theorem 9. That is, this directly contradicts the soundness of the $\mathcal{ZK}_{eq}$

protocol executed in Steps 5, 2e, or 2h of Construction 6 since $\mathcal{P}^*$ can be used in a black box manner to construct an adversary Adv which breaks the soundness of the $\mathcal{ZK}_{eq}$ protocol.

- **Event B occurs.** Again, this directly contradicts the soundness of the $\mathcal{ZK}_{prod}$ protocol executed in Step 2c of Construction 6 since $\mathcal{P}^*$ can thus be used in a black box manner to construct an adversary Adv which breaks the soundness of the $\mathcal{ZK}_{prod}$ protocol.

- **There exist $1 \leq i \leq d$ such that $\neg C_{i-1}$ occurs and events A, B do not.** We will now prove that this case contradicts the soundness property of Construction 6. Note that since $\mathcal{V}$ computes $\tilde{V}_x(r_d) \stackrel{\text{def}}{=} \tilde{V}_d(r_d)$ himself and since event A did not occur, C_d must occur. Therefore in this case it holds $\neg A \wedge \neg B \wedge \neg C_i \wedge \neg C_0 \wedge C_d$. Therefore, there must exist i' such that $\neg C_{i'-1} \wedge C_{i'}$.

  Let $a^*_{i'-1}$ be the extracted pre-image of $\mathsf{com}_{i'-1}$, $a^*_{i'}$ be the extracted pre-image of $\mathsf{com}_{i'}$. Since $\neg C_{i'-1} \wedge C_{i'}$ holds we obtain that $a^*_{i'} = \tilde{V}_{i'}(r_i)$ and that $a^*_{i'-1} \neq \tilde{V}_{i'-1}(r_{i'-1})$. Next, since $a^*_{i'} = \tilde{V}_{i'}(r_i)$ from Steps 2f-2h we obtain that (except with negligible probability) it holds that the extracted pre-images of $\mathsf{com}_{t_1}, \mathsf{com}_{t_2}$ are indeed equal to $\tilde{V}_{i'}(q_1), \tilde{V}_{i'}(q_2)$ (where (q_1, q_2) are the last $2s_i$ elements of r'_i).

  We now claim that $\mathcal{P}^*$ can be used in black-box manner to construct an adversary Adv that succeeds in falsely proving that $a^*_{i'-1}$ is equal to $\tilde{V}_{i'-1}(r_{i'-1})$, thus contradicting the soundness of Construction 6. Indeed, let $\mathcal{V}_{sum-check}$ be a verifier for Construction 7 using the coefficients of $\tilde{V}_{i'-1}$. Adv then performs (using the

code of $\mathcal{P}^*$) Step 1 of Construction 6. Since the verifier $\mathcal{V}$ for Construction 7 did not reject while interacting with $\mathcal{P}^*$ (and in particular, did not reject during Step 2a with $i = i'$), $\mathcal{V}_{sum-check}$ will not reject as well. Notice that, at this point in Construction 6, it is the case that com_ℓ is a commitment to $\tilde{V}_{i'-1}(r'_i)$ and so is $\mathsf{com}'_{i'}$ (defined in Step 2a of Construction 7 with $i = i'$).

Next, $\mathcal{V}_{sum-check}$ and Adv proceed by performing Steps 2 and 3 of Construction 6. We now argue that $\mathcal{V}_{sum-check}$ will not reject during Step 3 of Construction 6. Indeed, since the extracted pre-images t_1, t_2 of $\mathsf{com}_{t_1}, \mathsf{com}_{t_2}$ are equal to $\tilde{V}_{i'}(q_1), \tilde{V}_{i'}(q_2)$ and since $\mathcal{V}$ did not reject during Step 2c and 2d of Construction 7 with $i = i'$, we obtain that $\mathcal{V}$ successfully performed a step which is equivalent to Step 2 of Construction 6. Thus, $\mathcal{V}$ holds a commitment $\mathsf{com}^*_{i'}$ to $\tilde{V}_{i'-1}(r'_i)$ (using the notation of Construction 7) and $\mathcal{V}_{sum-check}$ holds a commitment com^*_ℓ to the same value $\tilde{V}_{i'-1}(r'_i)$. At this point, ass explained above we also have that that the pre-images of com_ℓ and $\mathsf{com}'_{i'}$ are both equal to $\tilde{V}_{i'-1}(r'_i)$ as well. Since event A did not occur, it holds that the pre-images of $\mathsf{com}'_{i'}$ and $\mathsf{com}^*_{i'}$ also have the same pre-image. By transitivity, we obtain that com^*_ℓ is a commitment to the same value as com_ℓ, thus $\mathcal{V}_{sum-check}$ will not reject during Step 3 of Construction 6. Therefore, we have violated the soundness of Construction 6 by allowing Adv to falsely prove that $a^*_{i'-1} = \tilde{V}_{i'-1}(r_{i'-1})$. □

Moreover, we prove the following lemma that will be helpful for us while proving the zero-knowledge property of our argument.

Lemma 3. *For every verifier $\mathcal{V}^*$ and for every depth-d layered circuit $C : \mathbb{F}^n \to \mathbb{F}^k$*

over a finite field $\mathbb{F}$ *there exists a simulator* Sim *such that* Sim *is capable of simulating the view of* $\mathcal{V}^*$ *in steps 1 and 2 of Construction 7 from* C*, without access to* x*.*

Proof Sketch. We build simulator Sim that simulates the view of $\mathcal{V}$ during Steps 1 and 2 of Construction 7. The simulator gets as input commitment parameters cp and a circuit C and proceeds as follows.

1. Sim sets $a_0 = 1$ and $r_0 = 0$, and computes $\mathsf{com}_x \leftarrow \mathsf{Com}(\mathsf{cp}, 0, \rho_x)$ for some ρ_x generated at random. Sim then sends com_x to $\mathcal{V}^*$.

2. Sim generates ρ_0 uniformly at random, computes $\mathsf{com}_0 \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{a}_0, \rho_0)$ and sends it to $\mathcal{V}$.

3. Sim proceeds to simulate Step 2 of Construction 7 as follows. For all $i = 1, \cdots, d$ Sim performs the following.

   (a) Let $\mathsf{Sim}_{sum-check}$ be the simulator from the proof of Lemma 2. Sim runs $\mathsf{Sim}_{sum-check}$ on input $(\mathsf{cp}, \mathsf{com}_{i-1}, s_{i-1}, a_{i-1})$, in order to simulate $\mathcal{V}^*$'s view during the execution of Step 2a. In addition to the final message com'_i sent to $\mathcal{V}^*$, $\mathsf{Sim}_{sum-check}$ also outputs a secret state (a_i, ρ_i) which is not forwarded to $\mathcal{V}^*$. Notice that a_i is the simulated value of $\tilde{V}_{i-1}(r'_i)$ where r'_i was chosen by $\mathcal{V}^*$.

   (b) Let (q_1, q_2) be the last $2s_i$ elements of r'_i. Sim chooses simulated values $t_1, t_2 \in \mathbb{F}$ for $\tilde{V}_i(q_1)$ and $\tilde{V}_i(q_2)$ such that a_i (which is the simulated value $\tilde{V}_{i-1}(r'_i)$), t_1, t_2 (which are the simulated values for $\tilde{V}_i(q_1)$ and $\tilde{V}_i(q_2)$) and r'_i satisfy Equation 2.1.

(c) Sim then computes $\mathsf{com}_{t_j} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{t_j}, \rho_{\mathsf{t_j}})$ for $j = 1, 2, 3$, where $t_3 = t_1 \cdot t_2$ and $\rho_{t_1}, \rho_{t_2}, \rho_{t_3}$ are chosen uniformly at random from $\mathbb{F}$, and forwards them to $\mathcal{V}^*$.

(d) Let Sim_{prod} be the simulator guaranteed from the zero knowledge property of $\mathcal{ZK}_{prod}$. Sim runs simulator Sim_{prod} on input $\mathsf{cp}, \mathsf{com}_{t_1}, \mathsf{com}_{t_2}, \mathsf{com}_{t_3}$ in order to simulate $\mathcal{V}^*$'s view during the execution of Step 2c of Construction 7).

(e) Sim performs Step 2d of Construction 7 using the values r'_i, t_1, t_2, t_3. This results in a commitment com^*_i to the value of $\tilde{V}_{i-1}(r'_i)$.

(f) Sim runs simulator Sim_{eq} on input $\mathsf{cp}, \mathsf{com}'_i, \mathsf{com}^*_i$ in order to simulate $\mathcal{V}^*$'s view during the execution of Step 2e of Construction 7).

(g) Sim computes $\mathsf{com}_{h(0)}$ as a fresh commitment to t_1. For $j = 1, \ldots, s_i - 1$, Sim chooses values $h_j \in \mathbb{F}$ uniformly at random. Moreover, he chooses h_{s_i} such that $\sum_{j=1}^{s_i} h_j + t_1 = t_2$ and for $j = 1, \ldots, s_i$ he computes $\mathsf{com}_{h_j} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{h_j}, \rho_{\mathsf{h_j}})$. Sim then sends $\mathsf{com}_{h_0}, \ldots, \mathsf{com}_{h_{s_i}}$ to $\mathcal{V}^*$.

(h) Sim computes $\mathsf{com}_{h(1)} \leftarrow \mathsf{com}_{h(0)} \cdot \prod_{j=1}^{s_i} \mathsf{com}_{h_j}$.

(i) Sim runs simulator Sim_{eq} on input $\mathsf{cp}, \mathsf{com}_{t_1}, \mathsf{com}_{h(0)}$ and on input $\mathsf{cp}, \mathsf{com}_{t_2}, \mathsf{com}_{h(1)}$ in order to simulate $\mathcal{V}^*$'s view during the execution of Step 2h of Construction 7).

(j) Finally, Sim sets $r_i \leftarrow \gamma(r''_i)$ (where r''_i was sent by $\mathcal{V}^*$ in Step 2j of Construction 7) and $a_i \leftarrow H(r''_i)$ (where H is the degree-s_i polynomial that has as coefficients $t_1 = h_0, h_1, \ldots, h_{s_i}$) and $\rho_i \leftarrow \sum_{j=0}^{s_i} r''^j \rho_{h_j}$. Finally, Sim computes $\mathsf{com}_i \leftarrow \mathsf{Comm}(\mathsf{a_i}, \rho_{\mathsf{i}})$.

We claim that the view of $\mathcal{V}^*$ while interacting with Sim (for Steps 1,2 of Construction 6) is indistinguishable from the view he gets while interacting with the honest prover $\mathcal{P}$ since: (i) All triplets a_i and t_1, t_2 (for each round i) chosen by Sim satisfy Equation 2.1, (ii) All values h_j (for each round i) satisfy the condition $h_0 = t_1$ and $\sum_{j=1}^{s_i} h_j + t_1 = t_2$, (iii) by assumption, the messages received by $\mathcal{V}^*$ by $\mathsf{Sim}_{eq}, \mathsf{Sim}_{prod}, \mathsf{Sim}_{sum-check}$ (forwarded via Sim) are indistinguishable form the ones received while running $\mathcal{ZK}_{eq}, \mathcal{ZK}_{prod}$ and Construction 6 with the honest prover, and (iv) (ii) Comm is statistically hiding. □

Other approaches to make CMT zero-knowledge. Chiesa et al. [34] showed how a large class of algebraic protocols (including sum-check and CMT) can be made zero-knowledge using only information theoretic techniques. While this is a very attractive property, it is not clear how to make their approach compatible with a VPD protocol from Section 3.1. Next, in a concurrent and independent work, Wahby et al. [84] presented an efficient zero-knowledge argument for sufficiently "parallel" circuits that utilizes the CMT protocol and uses the same general approach for making it zero-knowledge as the one used in this work (i.e., running the CMT protocol over homomorphic commitments). Unlike our construction which has a trusted preprocessing phase and relies on non-standard knowledge-of-exponent assumptions, the construction of [84] does not require any preprocessing and its security is based solely on the DDH assumption. However, while our construction achieves communication size and verification time that are polylogarithmic in the size of the witness w for NP-relation being verified, the communication size and verification time of [84] scale

with $O(\sqrt{|w|})$ which might be prohibitive for some applications.

6.4 Zero-Knowledge with Function Independent Preprocessing

In this section we construct our zero knowledge proof system with function independent preprocessing. We run the CMT protocol over homomorphic commitments, as described in 6.3, and by replacing the VPD construction of Section 3 with our zk-VPD construction from Section 6.2. Formally, consider the following protocol and theorem.

Construction 8 (**Zero-knowledge Delegation Protocol**). *Let λ be a security parameter and let $\mathbb{F}$ be a prime order field such that $|\mathbb{F}|$ is exponential in λ. In additional, let n be an input size parameter and let t be a circuit size parameter. In the following, for simplicity of exposition we assume that n is a power of 2. Consider the algorithms $\mathcal{G}, \mathcal{P}, \mathcal{V}$ described below.*

- ***Preprocessing phase.*** *The parameter generator $\mathcal{G}$ on input $1^n, 1^t, 1^\lambda$ runs $(\mathsf{pp}, \mathsf{vp}) \leftarrow \mathsf{KeyGen}(1^\lambda, n, 1)$. The proving key pk is set to be pp and the verification key vk is set to be vp.*

- ***Evaluation phase.*** *Let $C : \mathbb{F}^{n_x+n_w} \to \mathbb{F}$ be a depth-d layered arithmetic circuit over $\mathbb{F}$ such that $|C| \leq t$ and $n_x + n_w \leq n$. In addition, let $x \in \mathbb{F}^{n_x}$ and $w \in \mathbb{F}^{n_w}$ such that $C(x; w) = 1$. Assume that $n_w/n_x = 2^m - 1$ for some $m \in \mathbb{N}$. Consider the following protocol between $\mathcal{P}$ and $\mathcal{V}$.*

1. *Let* $\tilde{V}_d$ *be the multilinear extension of the input layer of* C *evaluated on* $(x; w)$. $\mathcal{P}$ *commits to the values of* $\tilde{V}_d$ *by executing* $\mathsf{com}_{\tilde{V}_d} \leftarrow \mathsf{CommitPoly}(\tilde{V}_d, \rho_{\tilde{V}_d}, \mathsf{pp})$ *where* $\rho_{\tilde{V}_d}$ *is generated uniformly at random.* $\mathcal{P}$ *then sends* $\mathsf{com}_{\tilde{V}_d}$ *to* $\mathcal{V}$.

2. $\mathcal{V}$ *runs* $\mathsf{CheckCom}(\mathsf{com}_{\tilde{V}_d}, \mathsf{vp})$. *In case* CheckCom *rejects,* $\mathcal{V}$ *rejects as well.*

3. $\mathcal{V}$ *and* $\mathcal{P}$ *execute Steps 1 and 2 of Construction 7. In case Construction 7 rejects, so does* $\mathcal{V}$. *Otherwise, at the end of Step 2 of Construction 7* $\mathcal{V}$ *holds a commitment* com_d *of an evaluation of* $\tilde{V}_d$ *at a random point* r_d *chosen by* $\mathcal{V}$ *while* $\mathcal{P}$ *holds the randomness* ρ_d *used to generate* com_d.

4. $\mathcal{P}$ *executes* $(\mathsf{com}^*_d, \pi) \leftarrow \mathsf{CommitValue}(\tilde{V}_d, r_d, \tilde{V}_d(r_d), \rho_{\tilde{V}_d}, \rho_{\tilde{V}_d(r_d)}, \mathsf{pp})$ *where* $\rho_{\tilde{V}_d(r_d)}$ *is generated uniformly at random and sends* (com^*_d, π) *to* $\mathcal{V}$.

5. *Upon receiving* (com^*_d, π), $\mathcal{V}$ *executes* $\mathsf{Ver}(\mathsf{com}^*_d, r_d, \mathsf{com}_{\tilde{V}_d}, \pi, \mathsf{vp})$. *In case* Ver *rejects, so does* $\mathcal{V}$.

6. $\mathcal{P}$ *and* $\mathcal{V}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, \tilde{V}_d(r_d), \rho_d, \rho_{\tilde{V}_d(r_d)}; \mathsf{com}^*_d, \mathsf{com}_d)$. *(Note that* cp *is a subset of* vp.*) In case* $\mathcal{ZK}_{eq}$ *rejects so does* $\mathcal{V}$.

7. $\mathcal{V}$ *computes the multilinear extension* $\tilde{x}$ *of the input* x, *generates a random point* $r_x \in (\mathbb{F}^{\log n_x} \times 0^{\log n_w})$ *and sends* r *to* $\mathcal{P}$.

8. *Upon receiving* r_x, $\mathcal{P}$ *executes* $(\mathsf{com}^*_x, \pi_x) \leftarrow \mathsf{CommitValue}(\tilde{V}_d, r_x, r_{\tilde{V}_d}, \rho_{r_x}, \mathsf{pp})$ *where* ρ_{r_x} *is generated uniformly at random and sends* $(\mathsf{com}^*_x, \pi_x)$ *to* $\mathcal{V}$. *Next,* $\mathcal{V}$ *executes* $\mathsf{Ver}(\mathsf{com}^*_x, r_x, \mathsf{com}_{\tilde{V}_d}, \pi_x, \mathsf{vp})$. *In case* Ver *rejects, so does* $\mathcal{V}$.

9. $\mathcal{V}$ *computes* $\mathsf{com}_x \leftarrow \mathsf{Com}(\mathsf{vp}, \tilde{\mathsf{x}}(\mathsf{r}'_{\mathsf{x}}), \rho'_{\mathsf{x}})$ *where* ρ'_x *is generated uniformly at random and* r'_x *is defined to be the first* $\log n_x$ *elements of* r_x. $\mathcal{V}$ *sends* ρ'_x *to* $\mathcal{P}$.

10. *Both* $\mathcal{P}$ *and* $\mathcal{V}$ *perform* $\mathcal{ZK}_{eq}(\mathsf{cp}, \tilde{V}_d(r_x), \rho_{r_x}, \rho'_x; \mathsf{com}_x, \mathsf{com}^*_x)$. *In case* $\mathcal{ZK}_{eq}$ *rejects so does* $\mathcal{V}$.

Theorem 11. *For any circuit size parameter* t*, input size* n *and finite field* $\mathbb{F}$*, Construction 8 is a zero-knowledge argument system for the relation*

$$\mathcal{R} = \{(C, x; w) : C \in \mathcal{C}_{\mathbb{F}} \wedge |C| \leq t \wedge \mathsf{inp}(C) \leq n \wedge C(x; w) = 1\}.$$

Moreover, for every $(C, x; w) \in R$ *the running time of* $\mathcal{P}$ *is* $O(|C| \cdot \log(\mathsf{width}(C)))$ *and if* C *is log-space uniform then the running time of* $\mathcal{V}$ *is* $O(|x| + d \cdot \mathrm{poylog}(|C|))$*. Finally* $\mathcal{P}$ *and* $\mathcal{V}$ *interact* $O(d \log(\mathsf{width}(C)))$ *rounds where* d *is the depth of* C*. In case* d *is* $\mathrm{polylog}(|C|)$*, the above construction is a* succinct *argument.*

Proof Sketch. The completeness property immediately follows from Construction 8. We now proceed to argue about the knowledge soundness property.

Knowledge soundness. Let Adv be a reduced version of $\mathcal{P}^*$ that aborts right after outputting $\mathsf{com}_{\tilde{V}_d}$. By the polynomial extractability property of Construction 5, there exists extractor $\mathcal{E}'$ that upon the same input as $\mathcal{P}^*$ and the same random tape, outputs a n-variable degree-variable 1, polynomial f and randomness ρ_f such that $\mathsf{CommitPoly}(f, \rho_f, \mathsf{pp}) = \mathsf{com}_{\tilde{V}_d}$ with all but negligible probability. We are now ready to build our extractor $\mathcal{E}$ as follows:

1. Run $\mathcal{E}'(1^\lambda, \mathsf{pp})$ and receive polynomial f and randomness ρ_f. If f is not a n-variable degree-variable 1, polynomial f, abort.

2. Output $w = (f(n_x), \ldots, f(n_w - 1))$.

We now argue that assuming $\mathcal{P}^*$ successfully convinced a verifier $\mathcal{V}$, it is indeed the case that $C(x, w) = 1$.

First, notice that com_x^* (produced via $\mathsf{CommitValue}$) and com_x are of the same format, i.e., regular Pedersen commitments under the same cp parameters (as described in Section 6.1). Thus, by the soundness property of the $\mathcal{ZK}_{eq}$ protocol we obtain that com_x and com_x^* are commitments to the same pre-image. Next, let $\mathcal{E}_{vpd,x}$ be the extractor for $\mathcal{P}^*$ (limited to Steps 1 and 8 of Construction 8) guaranteed by the evaluation extractability of property of Construction 5, as per Definition 6. Since $\mathcal{P}^*$ convinces $\mathcal{V}$ we obtain that $\mathcal{E}_{vpd,x}$ on the same inputs as $\mathcal{P}^*$ outputs $f(r_x)$ as the pre-image of com_x^* (with high probability).

We now argue that $(f(0), \cdots, f(n_x - 1)) = x$. Indeed, notice that f is an n-variate variable-degree-1 polynomial and it is thus a multilinear extension. In addition, by construction of $\tilde{x}$ (Step 7 of Construction 8) it holds that $(\tilde{x}(0), \cdots, \tilde{x}(n_x - 1)) = x$. Next, since com_x and com_x^* are commitments to the same value, we obtain that $\tilde{x}(r'_x) = f(r_x)$. Thus, by the properties of multilnear extensions we obtain that with high probability it holds that $(f(0), \cdots, f(n_x - 1)) = (\tilde{x}(0), \cdots, \tilde{x}(n_x - 1)) = x$.

We now proceed to argue that $C(x, w) = 1$. Let $x' = (x, w)$. We now show how to construct a prover $\mathcal{P}^*_{cmt}$ which will convince a verifier $\mathcal{V}_{cmt}$ from Construction 7 that $C(x') = 1$. Using the soundness property of Construction 7 we shall obtain that $C(x, w) = C(x') = 1$ with high probability. Indeed, let $x' = (x'_1, \cdots, x'_n)$, $\mathcal{P}^*_{cmt}$ starts by computing $\mathsf{com}_{x'_i} \leftarrow \mathsf{Com}(\mathsf{cp}, \mathsf{x}'_\mathsf{i}, \rho_{\mathsf{x}'_\mathsf{i}})$ where $\rho_{x'_i}$ is generated uniformly at random and cp was given to $\mathcal{P}^*$ by the parameter generator $\mathcal{G}$. $\mathcal{P}^*_{cmt}$ then sends $\mathsf{com}_{x'_1}, \cdots, \mathsf{com}_{x'_n}$ to $\mathcal{V}_{cmt}$ and proceeds as follows.

1. $\mathcal{P}^*_{cmt}$ sets $a_0 = 1, r_0 = 0$, generates ρ_0 uniformly at random, computes $\mathsf{com}_0 \leftarrow \mathsf{Comm}(\mathsf{cp}, \mathsf{a}_0, \rho_0)$ and sends it to $\mathcal{V}_{cmt}$. $\mathcal{P}^*_{cmt}$ then emulates $\mathcal{G}$ and runs P^* until Step 3 of Construction 8, discarding messages sent to $\mathcal{V}_{cmt}$.

2. Using $\mathcal{P}$ (restricted to Step 3 of Construction 8), $\mathcal{P}^*_{cmt}$ now interacts with $\mathcal{V}_{cmt}$ during Step 2 of Construction 7 by forwarding messages between $\mathcal{V}_{cmt}$ and $\mathcal{P}$. At the end of this step $\mathcal{P}^*_{cmt}$ and $\mathcal{V}_{cmt}$ hold a commitment com_d and a random point r_d chosen by $\mathcal{V}^*_{cmt}$.

3. $\mathcal{P}^*_{cmt}$ then sends x' and the randomness $\rho_{x'_1}, \cdots, \rho_{x'_n}$ to $\mathcal{V}_{cmt}$.

4. $\mathcal{P}^*_{cmt}$ then runs P^* until Step 6 of Construction 8 again discarding messages sent to $\mathcal{V}_{cmt}$.

5. Using $\mathcal{P}$ (restricted to Step 6 of Construction 8), $\mathcal{P}^*_{cmt}$ now interacts with $\mathcal{V}_{cmt}$ during Step 5 of Construction 7 by forwarding messages between $\mathcal{V}_{cmt}$ and $\mathcal{P}$.

We now proceed to argue that since $\mathcal{P}$ convinces $\mathcal{V}$ it is the case that $\mathcal{P}^*_{cmt}$ convinces $\mathcal{V}_{cmt}$. Indeed, first notice that since Step 2 of Construction 8 involve running running Steps 1 and 2 of Construction 7 and since $\mathcal{V}$ did not reject we have that $\mathcal{V}^*_{cmt}$ will not reject as well. Next, since the commitments $\mathsf{com}_{x'_1}, \cdots, \mathsf{com}_{x'_n}$ to $\mathcal{V}_{cmt}$ sent by $\mathcal{P}^*_{cmt}$ to $\mathcal{V}_{cmt}$ are honestly computed commitments to the values of x' using the randomness $\rho_{x'_1}, \cdots, \rho_{x'_n}$, we obtain that $\mathcal{V}^*_{cmt}$ will not reject during Step 3 of Construction 7. It remains to show that $\mathcal{V}^*_{cmt}$ will not reject during Step 5 of Construction 7.

Indeed, notice that f is the unique multilinear extension of $x' = (x, w)$. Thus we have that the polynomial $\tilde{V}_x$ defined in Step 4 of Construction 7 actually equals

f. Let $\mathcal{E}_{vpd,d}$ be the extractor for $\mathcal{P}^*$ (limited to Steps 1 and 4 of Construction 8) guaranteed by the evaluation extractability of property of Construction 5, as per Definition 6. Since $\mathcal{P}$ convinces $\mathcal{V}$ we have that with high probability $\mathcal{E}_{vpd,d}$ on the same inputs as $\mathcal{P}^*$ outputs $f(r_d)$ as the pre-image of com^*_d. Next, by uniqueness property of multilinear extensions, we have that the multilinear extension $\tilde{V}_{x'}$ of x' computed in Step 2 of Construction 7 equals f. This implies that commitment $\mathsf{com}^*_{x'}$ computed in Step 4 of Construction 7 (executed on input x') is also to a commitment to $\tilde{V}_{x'}(r_d) = f(r_d)$. Overall, since com_d is produced the same way in Construction 7 and Construction 8, we obtain that the $\mathcal{ZK}_{eq}$ protocol is executed on commitments to the same values. Thus, if $\mathcal{P}$ convinces $\mathcal{V}$ we obtain that $\mathcal{V}_{cmt}$ will also be convinced by $\mathcal{P}^*_{cmt}$.

Zero knowledge. Let Sim_{vpd} be the simulator from Theorem 8 and Sim_{cmt} be the simulator from Lemma 3. Consider the simulator Sim which is defined as follows.

1. On input $(1^\lambda, C, x)$, Sim runs Sim_{vpd} on input $(1^\ell, n, 1)$ where n is the input size of C and receives commitment $\mathsf{com}_{\tilde{V}_d}$, parameters pp, vp, and state σ. Note that pp contains commitment parameters cp for the Pedersen commitment scheme, as defined in Section 6.1. Sim sends vp to $\mathcal{V}^*$.

2. Sim runs Sim_{cmt} on input (cp, C), in order to simulate $\mathcal{V}^*$'s view during the execution of Step 3 of Construction 8. Let com_d be the corresponding output forwarded to $\mathcal{V}^*$ and r_d be the last random point chosen by $\mathcal{V}^*$.

3. In order to simulate $\mathcal{V}^*$'s view during Step 4 of Construction 8, Sim runs Sim_{vpd} on input $(r_d, \sigma, \mathsf{pp})$ and receives commitment com^*_d, proof π, and new state σ.

Sim then forwards (com_d^*, π) to $\mathcal{V}^*$.

4. Sim runs simulator Sim_{eq} on input $\mathsf{com}_d, \mathsf{com}_d^*$ in order to simulate $\mathcal{V}^*$'s view during the execution of Step 6 of Construction 8.

5. Upon receiving r_x from $\mathcal{V}^*$, Sim simulates $\mathcal{V}^*$'s view during Step 8 of Construction 8. To that end, Sim runs Sim_{vpd} on input $(r_x, \sigma, \mathsf{pp})$ and receives commitment com_x^*, proof π_x, and new state σ. Sim then forwards $(\mathsf{com}_x^*, \pi_x)$ to $\mathcal{V}^*$.

6. Upon receiving ρ'_x from $\mathcal{V}^*$, Sim computes $\mathsf{com}_x \leftarrow \mathsf{Comm}(\mathsf{vp}, \tilde{\mathsf{x}}(\mathsf{r}_\mathsf{x}), \rho'_\mathsf{x})$. Next, Sim runs simulator Sim_{eq} on input $\mathsf{com}_x, \mathsf{com}_x^*$ in order to simulate $\mathcal{V}^*$'s view during the execution of Step 10 of Construction 8.

We claim that the view of $\mathcal{V}^*$ while interacting with Sim is indistinguishable from the view he gets while interacting with the honest prover $\mathcal{P}$ since: (i) Comm is statistically hiding, (ii) the messages received by $\mathcal{V}^*$ by Sim_{eq} (forwarded via Sim) are indistinguishable from the ones received while running $\mathcal{ZK}_{eq}$ with the honest prover, (iii) the messages received by $\mathcal{V}^*$ by Sim_{cmt} (forwarded via Sim) are indistinguishable from the ones received while running Construction 7 with the honest prover, and (iv) the messages received by $\mathcal{V}^*$ by Sim_{vpd} (forwarded via Sim) are indistinguishable from the ones received while running Construction 5 with the honest prover. Note that the values of commitments $\mathsf{com}_d, \mathsf{com}_d^*, \mathsf{com}_x, \mathsf{com}_x^*$ are independent of each other (modulo the common commitment parameters cp) in both the real and the ideal execution. In particular, the messages exchanged during Step 3 of Construction 5 do not depend on the value of com_x ($\mathsf{com}_{\tilde{V}_d}$ in the real execution). □

Asymptotic complexity. Firstly, we note that Pedersen commitments, as well as protocols $\mathcal{ZK}_{eq}$, $\mathcal{ZK}_{prod}$, require a constant number of exponentiations and field operations (when instantiated as explained in Section 6.1). Then, the analysis of the asymptotic complexity of our argument follows in a straight forward manner from: (i) the analysis of CMT in Theorem 2, (ii) the analysis of the standard VPD in Theorem 3, (iii) the fact that the zk-VPD protocol of Section 6.2 has the same asymptotic behavior as the plain VPD in Section 3.1.

Chapter 7: Conclusions and Future Work

7.1 Conclusions

In this thesis, we introduce a new construction of an argument system. Our construction combines techniques developed in the literature of interactive proofs with an extractable verifiable polynomial delegation scheme, supporting proving relations in NP. Compared to the most commonly used existing techniques, i.e. SNARKs, our construction only requires a setup phase that is independent of the relation, and can be used later to prove any relation without a separate setup. In addition, we significantly improve the prover efficiency, by reducing the number of expensive cryptographic operations on the prover side from proportional to the size of the circuit representing the relation to the size of the input (and the witness).

We apply our new construction to build a verifiable database system that supports validating arbitrary SQL queries. The prover time is improved by up to 2 orders of magnitude compared to SNARK-based solutions and is comparable to those customized systems that support only a subset of SQL queries (e.g., IntegriDB). We also use our new argument systems to build a verifiable RAM program and show that it improves the prover time by 1-2 orders of magnitude and the mem-

ory consumption by 120× compared to prior work, and scales to prove a program with 2 million CPU instructions, beating the 32K instructions achieved by the best existing result.

Finally, we present a variant of our argument system that achieves an additional zero-knowledge property.

7.2 Future Directions

Efficient zero-knowledge argument without trusted setup. Though our new argument system removes the function-dependent setup phase, it still requires a trusted party to perform the key generation with a trapdoor, and the soundness and zero-knowledge will be broken if the trapdoor is leaked.

Many recent work [11, 14, 26, 30, 84] try to remove this trusted setup phase. However, they either produce a proof that is not succinct (square root of the size of the witness [11, 26, 84]), or require a linear verification time [11, 26, 30]. A remaining open problem is to construct an efficient and succinct argument system without trusted setup.

Privacy-preserving smart contracts. One key application of zero-knowledge proof is to construct privacy-preserving smart contracts. Smart contract systems on blockchain and crypto-currencies enforce the correct execution of digital contracts among many parties without a trusted third party. However, existing smart contracts have no privacy guarantee. All information of the contracts, such as the amount of money and the sender and the receiver of the transactions, are exposed

on the blockchain.

Kosba et. al. [59] proposed a system, Hawk, for privacy-preserving smart contracts using zero-knowledge proof. Instead of posting the contract in the clear on the blockchain, a contract manager would instead post a zero-knowledge proof that the contract is correctly executed. In this way, all parties of the blockchain can validate the contract without learning any of its information. However, zk-SNARK is used to construct the zero-knowledge proof in [59], which requires a trusted setup phase for every different contract. Because of this, a trusted party needs to be present to generate the parameters of zero-knowledge proof for every contract in Hawk, which deviates from the original goal of decentralization in blockchain.

A future direction is to apply our new zero-knowledge proof system to address this issue. With our new protocol, the public parameters are only generated once by a trusted authority, and can be used to generate proofs of difference smart contracts later. How to reduce the proof size and verification time of our protocol, and how to build compliers to translate a smart contract to our argument system are left as future work.

Bibliography

[1] Ate pairing. `https://github.com/herumi/ate-pairing`.

[2] Buffet. `https://github.com/pepper-project/releases`.

[3] The GNU multiple precision arithmetic library. `https://gmplib.org/`.

[4] Integridb. `https://github.com/integridb/Code`.

[5] jsnark. `https://github.com/akosba/jsnark`.

[6] libsnark. `https://github.com/scipr-lab/libsnark`.

[7] NTL library. `http://www.shoup.net/ntl/`.

[8] OpenSSL toolkit. `https://www.openssl.org/`.

[9] TPC-H benchmark. `http://www.tpc.org/tpch/`.

[10] Aiello, W., Bhatt, S. N., Ostrovsky, R., and Rajagopalan, S. Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for NP. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings* (2000), pp. 463–474.

[11] Ames, S., Hazay, C., Ishai, Y., and Venkitasubramaniam, M. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 2087–2104.

[12] Backes, M., Barbosa, M., Fiore, D., and Reischuk, R. M. ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data. In *S&P 2015*, pp. 271–286.

[13] Ben-Sasson, E., Bentov, I., Chiesa, A., Gabizon, A., Genkin, D., Hamilis, M., Pergament, E., Riabzev, M., Silberstein, M., Tromer, E., and Virza, M. Computational integrity with a public random string from quasi-linear PCPs. In *Advances in Cryptology—Eurocrypt 2017* (2017), pp. 551–579.

[14] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[15] Ben-Sasson, E., Chiesa, A., Genkin, D., and Tromer, E. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013* (2013), pp. 401–414.

[16] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*. 2013, pp. 90–108.

[17] Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO 2014*. 2014, pp. 276–294.

[18] Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security 2014* (2014).

[19] Benabbas, S., Gennaro, R., and Vahlis, Y. Verifiable delegation of computation over large datasets. In *CRYPTO 2011*, pp. 111–131.

[20] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS 2012* (2012), pp. 326–349.

[21] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012* (2012), pp. 326–349.

[22] Bitansky, N., Canetti, R., Paneth, O., and Rosen, A. On the existence of extractable one-way functions. STOC 2014, pp. 505–514.

[23] Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., and Paneth, O. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings* (2013), pp. 315–333.

[24] Boneh, D., and Boyen, X. Short signatures without random oracles. In *EUROCRYPT 2004* (2004), pp. 56–73.

[25] Bootle, J., Cerulli, A., Chaidos, P., and Groth, J. Efficient zero-knowledge proof systems. In *Foundations of Security Analysis and Design VIII - FOSAD 2014/2015/2016 Tutorial Lectures* (2016), pp. 1–31.

[26] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., and Petit, C. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2016), Springer, pp. 327–357.

[27] Boyle, E., Goldwasser, S., and Ivan, I. Functional signatures and pseudorandom functions. In *PKC 2014*, pp. 501–519.

[28] Boyle, E., and Pass, R. Limits of extractability assumptions with distributional auxiliary input. In *ASIACRYPT 2015*, pp. 236–261.

[29] Braun, B., Feldman, A. J., Ren, Z., Setty, S. T. V., Blumberg, A. J., and Walfish, M. Verifying computations with state. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013* (2013), pp. 341–357.

[30] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), vol. 00, pp. 319–338.

[31] Canetti, R., Chen, Y., Holmgren, J., and Raykova, M. Adaptive succinct garbled RAM or: How to delegate your database. In *TCC 2016-B*, pp. 61–90.

[32] Canetti, R., Paneth, O., Papadopoulos, D., and Triandopoulos, N. Verifiable set operations over outsourced databases. In *PKC 2014*, pp. 113–130.

[33] Catalano, D., Fiore, D., Gennaro, R., and Nizzardo, L. Generalizing homomorphic MACs for arithmetic circuits. In *PKC 2014*, pp. 538–555.

[34] Chiesa, A., Forbes, M. A., and Spooner, N. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017. `http://eprint.iacr.org/2017/305`.

[35] Chiesa, A., Tromer, E., and Virza, M. Cluster computing in zero knowledge. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II* (2015), pp. 371–403.

[36] Cormode, G., Mitzenmacher, M., and Thaler, J. Practical verified computation with streaming interactive proofs. In *ITCS 2012* (2012), pp. 90–112.

[37] Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., and Zahur, S. Geppetto: Versatile verifiable computation. In *S&P 2015* (2015), pp. 253–270.

[38] Cramer, R., and Damgård, I. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings* (1998), pp. 424–441.

[39] Danezis, G., Fournet, C., Groth, J., and Kohlweiss, M. Square span programs with applications to succinct NIZK arguments. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I* (2014), pp. 532–550.

[40] Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., and Stubblebine, S. G. Flexible authentication of XML documents. In *CCS 2001*, pp. 136–145.

[41] Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., and Parno, B. Hash first, argue later: Adaptive verifiable computations on outsourced data. Cryptology ePrint Archive, 2016.

[42] Fiore, D., and Gennaro, R. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *CCS 2012*, pp. 501–512.

[43] Fournet, C., Kohlweiss, M., Danezis, G., and Luo, Z. ZQL: A compiler for privacy-preserving data processing. In *USENIX Security 2013*, pp. 163–178.

[44] Garay, J. A., MacKenzie, P. D., and Yang, K. Strengthening zero-knowledge protocols using signatures. *J. Cryptology 19*, 2 (2006), 169–209.

[45] Gennaro, R., Gentry, C., and Parno, B. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings* (2010), pp. 465–482.

[46] Gennaro, R., Gentry, C., Parno, B., and Raykova, M. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings* (2013), pp. 626–645.

[47] Ghuloum, A. M., and Fisher, A. L. Flattening and parallelizing irregular, recurrent loop nests. In *ACM SIGPLAN Notices* (1995), vol. 30, ACM, pp. 58–67.

[48] Goldwasser, S., Kalai, Y. T., and Rothblum, G. Delegating computation: interactive proofs for muggles. In *STOC 2008* (2008), pp. 113–122.

[49] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof-systems. In *STOC 1985*, pp. 291–304.

[50] Groth, J. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security* (2010), Springer, pp. 321–340.

[51] Groth, J. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings* (2010), pp. 321–340.

[52] Groth, J. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II* (2016), pp. 305–326.

[53] Groth, J. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016* (2016), pp. 305–326.

[54] Ishai, Y., Kushilevitz, E., and Ostrovsky, R. Efficient arguments without short pcps. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA* (2007), pp. 278–291.

[55] Kalai, Y. T., and Paneth, O. Delegating RAM computations. In *TCC 2016-B,*, pp. 91–118.

[56] Kate, A., Zaverucha, G. M., and Goldberg, I. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, pp. 177–194.

[57] Kilian, J. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada* (1992), pp. 723–732.

[58] Knuth, D. E., Morris, Jr, J. H., and Pratt, V. R. Fast pattern matching in strings. *SIAM journal on computing 6*, 2 (1977), 323–350.

[59] Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on* (2016), IEEE, pp. 839–858.

[60] Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., abhi shelat, and Shi, E. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. `http://eprint.iacr.org/2015/1093`.

[61] Kosba, A. E., Papadopoulos, D., Papamanthou, C., Sayed, M. F., Shi, E., and Triandopoulos, N. TRUESET: Faster verifiable set computations. In *USENIX Security 2014*, pp. 765–780.

[62] Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L. Dynamic authenticated index structures for outsourced databases. In *SIGMOD 2006*, pp. 121–132.

[63] Lipmaa, H. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography Conference* (2012), Springer, pp. 169–189.

[64] Lipmaa, H. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I* (2013), pp. 41–60.

[65] Lund, C., Fortnow, L., Karloff, H., and Nisan, N. Algebraic methods for interactive proof systems. *J. ACM 39*, 4 (1992), 859–868.

[66] Micali, S. Computationally sound proofs. *SIAM J. Comput. 30*, 4 (2000), 1253–1298.

[67] Papadopoulos, D., Papadopoulos, S., and Triandopoulos, N. Taking authenticated range queries to arbitrary dimensions. In *CCS 2014*, pp. 819–830.

[68] Papadopoulos, D., Papamanthou, C., Tamassia, R., and Triandopoulos, N. Practical authenticated pattern matching with optimal proof size. *VLDB 2015*, 750–761.

[69] Papamanthou, C., Shi, E., and Tamassia, R. Signatures of correct computation. In *TCC 2013* (2013), pp. 222–242.

[70] Papamanthou, C., Tamassia, R., and Triandopoulos, N. Optimal verification of operations on dynamic sets. In *CRYPTO 2011*, pp. 91–110.

[71] Parno, B., Howell, J., Gentry, C., and Raykova, M. Pinocchio: Nearly practical verifiable computation. In *S&P 2013* (2013), pp. 238–252.

[72] Parno, B., Raykova, M., and Vaikuntanathan, V. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings* (2012), pp. 422–439.

[73] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings* (1991), pp. 129–140.

[74] Setty, S., Braun, B., Vu, V., Blumberg, A. J., Parno, B., and Walfish, M. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys 2013*, pp. 71–84.

[75] Setty, S. T., McPherson, R., Blumberg, A. J., and Walfish, M. Making argument systems for outsourced computation practical (sometimes). In *NDSS 2012*, p. 17.

[76] Setty, S. T. V., Vu, V., Panpalia, N., Braun, B., Blumberg, A. J., and Walfish, M. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium 2012*, pp. 253–268.

[77] Tamassia, R. Authenticated data structures. In *European Symposium on Algorithms* (2003), Springer, pp. 2–5.

[78] Thaler, J. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO 2013* (2013), pp. 71–89.

[79] Thaler, J. A note on the GKR protocol, 2015. Available at `http://people.cs.georgetown.edu/jthaler/GKRNote.pdf`.

[80] Thaler, J. R. *Practical verified computation with streaming interactive proofs*. PhD thesis, 2013.

[81] Vu, V., Setty, S., Blumberg, A. J., and Walfish, M. A hybrid architecture for interactive verifiable computation. In *S&P 2013* (2013), pp. 223–237.

[82] Wahby, R. S., Ji, Y., Blumberg, A. J., Shelat, A., Thaler, J., Walfish, M., and Wies, T. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 2071–2086.

[83] Wahby, R. S., Setty, S. T., Ren, Z., Blumberg, A. J., and Walfish, M. Efficient ram and control flow in verifiable outsourced computation. In *NDSS* (2015).

[84] Wahby, R. S., Tzialla, I., abhi shelat, Thaler, J., and Walfish, M. Doubly-efficient zkSNARKs without trusted setup.

[85] Walfish, M., and Blumberg, A. J. Verifying computations without reexecuting them. *Commun. ACM 58*, 2 (2015), 74–84.

[86] Wang, X., Malozemoff, A. J., and Katz, J. EMP-toolkit: Efficient multiparty computation toolkit. `https://github.com/emp-toolkit`.

[87] Yang, Y., Papadias, D., Papadopoulos, S., and Kalnis, P. Authenticated join processing in outsourced databases. In *SIGMOD 2009*, pp. 5–18.

[88] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., and Papamanthou, C. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE Symposium on Security and Privacy (S&P) 2017* (2017).

[89] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., and Papamanthou, C. A zero-knowledge version of vsql. Tech. rep., Cryptology ePrint Archive, Report 2017/1146, 2017. https://eprint. iacr. org/2017/1146, 2017.

[90] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., and Papamanthou, C. vram: Faster verifiable ram with program-independent preprocessing. In *Proceeding of IEEE Symposium on Security and Privacy (S&P)* (2018).

[91] Zhang, Y., Katz, J., and Papamanthou, C. IntegriDB: Verifiable SQL for outsourced databases. In *CCS 2015* (2015), pp. 1480–1491.

[92] Zhang, Y., Katz, J., and Papamanthou, C. An expressive (zero-knowledge) set accumulator. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on* (2017), IEEE, pp. 158–173.

[93] Zhang, Y., Papamanthou, C., and Katz, J. Alitheia: Towards practical verifiable graph processing. In *CCS 2014*, pp. 856–867.

[94] Zheng, Q., Xu, S., and Ateniese, G. Efficient query integrity for outsourced dynamic databases. In *CCSW 2012*, pp. 71–82.