2014

Towards practical verifiable computation: verification outsourcing, linear arguments without linearity tests, and repeated structures

Gang Xu
Iowa State University

Towards practical verifiable computation: Verification outsourcing, linear arguments without linearity tests, and repeated structures

by

Gang Xu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Yong Guan, Co-Major Professor

George Amariucai, Co-Major Professor

Cliff Bergman

Doug Jacobson

Jack Lutz

Iowa State University

Ames, Iowa

2014

DEDICATION

To the happy few.

TABLE OF CONTENTS

LIST OF TABLES vii

LIST OF FIGURES viii

ACKNOWLEDGEMENTS ix

ABSTRACT x

CHAPTER 1. INTRODUCTION 1

1.1 Verification for Repeated Structures 2

1.2 Linear Arguments without Linear Tests 4

1.3 Verification Outsourcing 5

1.4 Research Scope 6

1.5 Innovative Claims 7

CHAPTER 2. RELATED WORK 9

CHAPTER 3. PRELIMINARIES 12

3.1 Probabilistically Checkable Proofs (PCPs) 12

3.2 Homomorphic Encryption 13

3.3 Arguments 13

3.4 Efficient Arguments without Short PCPs 13

3.5 Two Recent Efficient Arguments: PEPPER and GINGER 18

3.6 Quadratic Programs 20

3.7 A Recent Efficient Argument System: Zaatar 21

CHAPTER 4. DELEGATION OF COMPUTATION WITH VERIFICATION OUTSOURCING: CURIOUS VERIFIERS 23

4.1 Problem Statement 23

4.1.1 System Model 23

4.1.2 Threat Model 24

4.1.3 Design Goals 24

4.2 Basic Scheme: Verification Without Circuit Information 25

4.2.1 A Building Block: A New Commitment Protocol 25

4.2.2 A Delegation-of-Verification Scheme with Partial Circuit Confidentiality 28

4.2.3 Theoretical Analysis: Correctness and Soundness 29

4.3 The Full Solution to Delegating Verification to a Curious Verifier 30

4.3.1 Outsourcing Computation Phase 30

4.3.2 Constructing Commitments 30

4.3.3 Outsourcing Verification Phase 31

4.3.4 Making A Decision 32

4.3.5 Security Analysis 32

4.4 Practical Use and Complexity Analysis 36

4.4.1 Amortized Query Costs 36

4.4.2 Complexity Analysis 37

4.5 Mathematical Proof of the Commitment 38

4.6 Conclusions and Future Directions 47

CHAPTER 5. VERIFIABLE COMPUTATION WITH REDUCED INFORMATIONAL COSTS AND COMPUTATIONAL COSTS 49

5.1 Motivation 49

5.1.1 Our Contributions 51

5.2 System Model 52

5.3 A Technique: A Commitment Providing Inherent Linearity Tests 52

5.4 A Reduced-Investment Verifiable Computation Protocol: RIVER 54

5.4.1 PCP Querying 55

5.4.2 PCP Querying of RIVER 55

5.4.3 Commit, Decommit and Consistency Verification of RIVER . . . 57

5.5 Performance Analysis 60

5.5.1 The Verifier 61

5.5.2 The Prover 64

5.6 Conclusions 65

CHAPTER 6. BLOCK PROGRAMS: IMPROVING EFFICIENCY OF VERIFIABLE COMPUTATION FOR CIRCUITS WITH REPEATED SUBSTRUCTURES 68

6.1 Problem Statement 68

6.2 Our Theoretical Results: Block Programs (BPs) 69

6.3 Our Scheme: Interactive Verification for Loops 74

6.3.1 Theoretical Results for Loop Circuits 74

6.3.2 Our Interactive Verification for Loops 79

6.3.3 Security Analysis 82

6.3.4 Improving the Performance through Batching 83

6.4 Performance Evaluation 84

6.4.1 Performance Analysis and Comparison 84

6.4.2 An Example for Performance Comparison 87

6.4.3 Further Discussion 88

6.5 Mathematical Proofs 89

6.5.1 Complete Proof of Theorem 17 89

6.5.2 Proof of Soundness 91

6.5.3 Proof of Completeness 96

6.6 Cost Analysis 97

6.6.1 Amortized Cost Calculation 97

6.6.2 Non-amortized Cost 99

6.6.3 Prover's Cost 100

6.7 Conclusions 100

CHAPTER 7. FUTURE WORK: DISTRIBUTED VERIFIABLE COMPUTATION WITH VERIFICATION OUTSOURCING 102

7.1 Motivation and Problem Statement 102

7.2 Methods and the Key Contribution 104

BIBLIOGRAPHY 107

LIST OF TABLES

Table 3.1 Commitment Protocol of IKO [7] 16

Table 3.2 The Single-Commit-Multi-Decommit Design [9] 19

Table 4.1 Comparison of Commitment Protocols 26

Table 4.2 Comparison of Complexity for Each Instance 37

Table 5.1 The First Round of Our QAP-based Linear PCP 57

Table 5.2 The Second Round of Our QAP-based Linear PCP 58

Table 5.3 The Second Round of Our QAP-based Linear PCP 59

Table 5.4 The Decision Making Stage of Our QAP-based Linear PCP . . . 60

Table 5.5 Decommit Design for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D . . 60

Table 5.6 Comparison of Cost for Verifier in Each Instance 66

Table 5.7 Comparison of Cost for Prover in Each Instance 67

Table 6.1 BP of the circuit 74

LIST OF FIGURES

Figure 3.1	Zaatar's linear PCP based on QAPs	22
Figure 5.1	PCP querying	56
Figure 6.1	Computing through Blocks	70
Figure 6.2	Notations	70
Figure 6.3	A Loop Circuit	75
Figure 6.4	One Single Block	76
Figure 6.5	Our Verification Protocol	82
Figure 6.6	Comparison of Costs in Each Instance	86
Figure 6.7	An Loop Example and Its Cost	88
Figure 7.1	A basic scenario of distributed delegation of verification with distributed delegation of verification.	103
Figure 7.2	Delegation of computation with delegation of verification in a simple star topology.	105

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis.

First and foremost, Dr. Yong Guan and Dr. George Amariucai for their guidance, patience and support throughout this research and the writing of this thesis. Their insights and words of encouragement have often inspired me and renewed my knowledge for completing my graduate education.

I would also like to thank my committee members for their efforts and contributions to this work: Dr. Cliff Bergman, Dr. Doug Jacobson, and Dr. Jack Lutz. Their valuable advices and assistance helped me to improve the quality of my thesis.

I am also thankful to Dr. David C. Jiles for his encouragement and valuable advices. I additionally like to thank all the instructors of the courses I took during my Ph.D program, from whom I learned a lot. I would like to thank all my labmates and friends at Iowa State University.

ABSTRACT

Cloud Computing represents a new trend in modern computing. Since computation can be purchased as a service, companies and individual users can cut down their computing assets and outsource any burdensome computational workload. In addition to savings in computing infrastructure, the Cloud may also provide expert technical consulting. But while outsourcing computation provides appealing benefits, one must fully consider a critical security issue: there is no guarantee on the correctness of the results. That is, the Cloud servers should be considered error-prone and may or may not be fully trustworthy. Thus an immediate need for result assurance naturally arises. This need motivates a growing body of research on verification of outsourced computation. Researchers strive for verifying the result of general computation, not limited to a specific computational task. Extending classical proof systems, interactive proof (IP) systems and probabilistically checkable proof (PCP) systems provide basic theoretical models and meaningful tools for applications. Unfortunately, PCPs and hence arguments are wildly impractical: traditional PCPs are too expensive to instantiate at the prover or query from the verifier. While state-of-the-art PCP schemes are asymptotically efficient, the constants on their running times are large, and they seem too intricate to be implemented easily.

This dissertation focuses on the verifiable computation, taking steps towards bringing it closer to practicality. We argue that since the verification may be tedious and expensive, users are likely to outsource (again) the verification workload to a third party. Other scenarios such as auditing and arbitrating may also require the use of third-party verification. Outsourcing verification will introduce new security challenges. One such challenge is to protect the computational task and the results from the untrusted third party verifier. In this work, we address this problem by proposing an efficient verification outsourcing scheme. To our knowledge, this is the first solution to the verification outsourcing problem. We show that, without using expensive fully-homomorphic encryption, an honest-but-curious third party can help to verify the result of an outsourced computational task without having to learn either the computational task or the result thereof. We have implemented our design by combining a novel commitment protocol and an additive-homomorphic encryption in the argument system model. The total cost of the verification in our design is less than the verifiers cost in the state-of-the-art argument systems that rely only on standard cryptographic assumptions.

Besides the introduction of the verification outsourcing paradigm, we also bring improvements to the state-of-the-art verification protocol designs. We firstly investigate the linearity tests, which overwhelmingly occupy the bandwidth of the interaction part of the state-of-the-art designs based on linear PCP. Our results show that under certain assumptions, if this Single- Commit-Multi-Decommit protocol further combines with the linear PCP, the linearity tests in the combined linear PCP become redundant. Our theoretical result immediately results in RIVER, a new linear-PCP-based argument system which achieves lower cost. Then, we focus on the computations with repeated sub-structures and design a novel verification protocol, that takes advantage of these particular features. We notice the state of the art involve a considerable cost, including the verifiers amortized cost, (i.e., the cost that needs to be amortized over a large number of work instances), and the provers cost of proof generation. The most efficient argument systems still incur an amortized cost that is linear in the size of the circuit. We address reducing this cost for those outsourced computations which contain repeated substructures (e.g. loops). Since loops play a pivotal role in the real world of computing (not only compute-intensive computations but also data-intensive computations such as big data applications), we take loops as a typical example, propose the first verification protocol that is specific for computations with repeated structures and show that the circuit generated from computation with loops can indeed lead to a lower amortized cost and a lower cost of proof generation. Using the theory of arithmetic circuit complexity we prove that for most programs our design results in very significant savings.

CHAPTER 1. INTRODUCTION

The surging popularity of the cloud computing paradigm has rendered a new type of service: computation as a commodity. While using this service, companies and individual users must no longer maintain expensive computing assets. They just outsource any burdensome computational workload to the cloud and enjoy additional perks, like expert technical consulting. But while outsourcing computation provides appealing benefits, one must fully consider a critical security issue: there is no guarantee on the correctness of the results returned by the cloud server, which may be error-prone or otherwise not entirely trustworthy. Thus an immediate need for result assurance naturally arises.

This need motivates a growing body of research on *verifiable computation*, and in particular, works focused on verification protocols for general-purpose computation. Since verifying the result of general computation can be abstracted to classical problems in the theory of computation, such as interactive proof (IP) systems [1] and probabilistically checkable proof (PCP) systems [2, 3], the security community naturally turned to the model of these classical proof systems, attempting to refine theory toward implementation. In their designs, the server plays the role of a *prover* trying to convince the client, who plays the role of a *verifier*, that the result is correct. A recent line of work strives for verifying computation based on *argument systems* [4, 5, 6], a notable variant of the PCP model. They hold a more practical assumption that, in addition to the verifier being polynomial-time probabilistic, the prover is also computationally bounded. Breakthroughs [7, 8, 9, 10, 11, 12] in argument systems have made PCP-based approaches more practical. Another important line of work [13, 10, 14, 15] makes attempts to adopt the recent finding of a new characterization of the NP complexity class – the *Quadratic Span Programs (QSPs)* (and *Quadratic Arithmetic Programs (QAPs)*) [13]. Based on theoretical innovations of QAP [13], a nearly practical verifiable computation system called Pinocchio was introduced in [14].

The ultimate goal of all these methods is to ensure that the amount of verification workload performed by the client is less than the workload of performing the same computation from scratch. Although recent solutions show encouraging results, making verification closer to practicality than ever, the workload of verification remains quite expensive, especially for those cases requiring large-scale verification (such as when large amounts of computation need to be outsourced, and then the results verified). But the average users may not be willing to spend their valuable time and resources on verification work, even though this new computational task is much less demanding than the original one, and neglect verification altogether. We can hardly imagine a hand-held device user devote CPU time and wireless bandwidth to verification.

To achieve practical verifiable computation, one direct idea is to strive for efficient verification protocols, taking steps to pursue more practical argument systems. While part of the work of this dissertation follows this idea, we also propose another idea towards practical verifiable computation: verification outsoucing.

1.1 Verification for Repeated Structures

Although encouraging results have been emerging, the high costs still stymie their practicality. In particular, the biggest performance concerns are around further reducing the *amortized cost* of verification, and reducing the *cost of proof generation*. Zaatar [10] and Pinocchio [14] are two representatives of the state of the art in verifiable computation. The verification protocols in both Zaatar and Pinocchio can be viewed as argument systems with *amortizing*, namely, proof systems that require a high cost for the verifier that needs to be amortized over a large number of work instances. In Pinocchio, the verifier needs to publish a public key and hold a matching verification key. Both the cost of constructing this verification key and the public key are amortized over all possible work instances of the same circuit. In Zaatar, the verifier must invest in the expensive construction of the commitment and all the PCP queries before he performs the actual verification operations for each instance. This amortized cost grows linearly with the circuit size in both Zaatar and Pinocchio, and it can be prohibitive, especially for large circuits, which are widely used in many practical scenarios. Meanwhile, the costs of proof generation in Zaatar and Pinocchio grow at least linearly with the circuit size. Assuming the circuit size of the computation task is $\mathcal{S}$, then the cost of proof generation in Zaatar is asymptotically $O(\mathcal{S} \cdot log^2(\mathcal{S}))$ and the cost of proof generation in Pinocchio is asymptotically $O(\mathcal{S})$.

From a theoretical perspective, one natural question is whether it is possible to have sub-linear time (in the size of the circuit) amortized cost and cost of proof generation. To the best of our knowledge, this remains an open question. From a practical perspective, the first requirement is to further reduce both amortized cost and cost of proof generation, which act as chief obstacles in verifiable computation.

To tackle these problems, instead of merely focusing on optimizing current verification algorithms regardless of the structure of the computation tasks, like in most recent works, in this dissertation we take into account the structure of the circuit based on which the computation tasks are verified.

We observe that *repeated structures* play a pivotal role in the arithmetic circuit based on which verification protocol is performed. The compilers of Zaatar and Pinocchio analyze any high-level program piece by piece and generate the corresponding circuit in a straightforward way: for instance, loops are unrolled in a naive way. We notice that almost every computation (e.g. Big Data!) employs loops. Moreover, these computations are the most likely tasks to be outsourced to the cloud server. Repeated structures show up frequently in the circuit generated by Zaatar's and Pinocchio's compilers. Meanwhile, vnTinyRAM universally transforms any C program with fixed number of execution steps into one single arithmetic circuit, which also contains lots of repeated subcircuits such as C_{mem}, C_{exe} and the multiplexers in the Waksman networks [16].

Repeated structures (e.g. looping structures) are not well addressed in the current research on verification protocols. As observed in [9, 12], looping can not be handled concisely. Hence, if we can take advantage of repeated structures in the circuits of these computations and handle the verification better, we could make verifiable computation more efficient.

Since we use terminology of *loop* extensively in this dissertation, we formalize it here. In this dissertation, the *loop body* is the piece of code describing the executions inside the loop. One *loop iteration* refers to one execution of the loop body.

In this dissertation, we address the problems proposed above, argue that verifiable computation can be made cheaper by taking advantage of computations whose circuits contain repeated substructures (e.g. loops), and achieve cheaper verifiable computation through efficient loop-handling.

1.2 Linear Arguments without Linear Tests

In the line of linear-PCP fashion verifiable computation designs, once the prover is committed to a proof, the verifier has to perform laborious linearity tests to ensure the proof is linear. In fact, the number of queries required to perform linearity tests dominate the number of overall queries of the protocol. Thus, the cost caused by linearity test is still one of the bottlenecks of current protocols. Up to now, in the context of Single-Commit-Multi-Decommit protocol, whether the linearity tests are necessary was still an open question. In this section, we propose our theoretical result, showing that under a particular assumption, the Single-Commit-Multi-Decommit protocol will provide inherent linearity testing. (Specifically, we assume that the commitment information that the prover holds is computed by a linear function.) Thus, if linear PCP is combined with this commitment protocol, the linearity tests are obsolete. We will adopt these theoretical results in our verification protocol design and thus achieve cost savings.

1.3 Verification Outsourcing

In the spirit of outsourcing computation, a natural idea is to also outsource the verification. For this purpose, the client may delegate the verification to a third party – *the verifier*. The verifier does not need to be as powerful as the server doing the original computation. In the pay-per-use paradigm, the client should pay the verifier far less than the prover.

In addition to this novel verification-outsourcing paradigm, third-party verification may benefit other, equally-important applications. For example, disputes between the server and the client can be solved by an arbitrator who plays the role of the third-party verifier. Similar verifications may be required by government agencies, nonprofit organizations, and consumer safety organization, for the purpose of quality evaluation, project management, etc.

However, outsourcing verification is not trivial to implement. Several challenges emerge when outsourcing verification to untrusted verifiers. One of these, and the main focus of this dissertation, is the confidentiality concern. The results of computing are often confidential. Moreover, in many instances, even the details of the computation task itself may constitute sensitive material.

To the best of our knowledge, there is no feasible solution to these challenges. The two-party verification schemes cannot be directly adopted, either. Recomputing requires the verifier to have the same resources as the prover. If the prover provides a traditional NP-proof, the verifier is able to verify the result with fewer resources than required to recompute. But he still needs to read the entire proof, which costs polynomial time in the size of the computational task. Apart from the high cost, recomputing or checking NP-proofs provides little defence against curious verifiers.

In IP-based or PCP-based two-party verification schemes, it is necessary for the verifier to have perfect knowledge of the computation task and the result (in the context of computational complexity, the verifier needs to know the instance of the problem). If the third party simply runs the verifier's algorithms according to these two-party designs, the computation task and the result cannot be protected unless an expensive fully-homomorphic encryption system (e.g. [17]) is deployed.

The challenge here is how a third party can verify the correctness of the result *without* knowing the computation task and the result and without using expensive fully-homomorphic encryption [17]. In this dissertation, we describe a secure third-party confidentiality-preserving verification scheme.

Our work is related to, but different from delegation of computation to two or more servers [18] [19] [20] [21], where multiple servers with the same computational power compute individually and compete to convince the client to accept their results. In our design, without performing the same computation as the prover, the third party only needs fewer resources to verify the result from the prover.

1.4 Research Scope

In the context of verifiable computation, there are two stages: one in which the outsouced computation task, which is a piece of code written in the form of a high-level language (e.g. C), is transformed into an arithmetic circuit, and another in which the actual verification protocol is performed to check that the prover correctly evaluated the circuit generated in the first stage. The core of the first transformation stage is a specific compiler, also known as a circuit generator. State-of-the-art compilers includes vnTinyRAM [15, 16] and the compilers in Zaatar and Pinocchio. These compiler techniques are beyond the scope of this dissertation, and we review these compilers for completeness in Chapter 2. In this dissertation, we are interested in the second stage, namely, the actual verification protocol. In the rest of the dissertation, we assume that the underlying circuit representation has been generated using the aforementioned circuit generators.

1.5 Innovative Claims

Our work is framed in the context of recent results from delegation of computation. Our research will investigate completely novel paradigms, and has the potential to generate an abundance of related research. Our main innovative contributions are broadly categorized below.

1. Our results show that computation involving loop structures can indeed lead to a lower set-up cost namely, a set-up cost which is linear in the degree of the loop body (i.e., the degree of the polynomial that describes the loop body), instead of its size (i.e., the number of multiplication gates in the circuit description of the loop body). For most programs, this results in very significant savings.

2. We investigate the linearity tests, which overwhelmingly occupy the communication of the total cost of the state-of-the-art designs based on linear PCP. Our results show that under certain assumptions, if this Single-Commit-Multi-Decommit protocol further combines with the linear PCP, the linearity tests in the combined linear PCP are redundant. Our theoretical result immediately results in RIVER, a new linear-PCP-based argument system which achieves lower cost.

3. Our research shows, for the first time, that verification can be outsourced to an untrusted third-party, who can verify the correctness of the result *without* knowing the

computation task and the result and without using expensive fully-homomorphic encryption [17].

4. We propose to investigate a new paradigm: secure delegation of computation with verification outsourcing. At the time of this research, we are not aware of any results on “delegation of verification”. In fact, we are the first to ever discuss the idea of multiple outsourcing in the context of delegation of computation.

CHAPTER 2. RELATED WORK

Extensive research has been motivated by the problem of verifying computation, However, much of the prior work focuses on specific problems and exploits properties of these problems for efficient verification. [39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]. Meanwhile, more work strives for verifying the result of general computation. Verifying the result of general computation can be viewed as originating from similar but more abstract problems in the theory of computation, such as interactive proof (IP) systems [1] and probabilistically checkable proof (PCP) systems [2, 3]. Early results [23, 2, 3, 19, 25, 25, 5, 6, 26] were viewed as important findings only in theoretical computer science. The current focus is on the efficiency [27, 28], length [27, 29, 30, 31] and soundness error [32, 33, 34, 35] of PCPs. Until recently, the security community has been extending and refining classical proof systems, attempting to make theoretical cryptographic protocols practical.

Extending provides basic theoretical models and meaningful tools for applications. Existing verifiable computation schemes fall into three broad categories. The first line of verifiable computation systems [8, 9, 10] is based on the IKO argument system which is first proposed by Ishai et al. [7]. In these systems, the proof for the result correctness is formulated into a linear PCP with a commitment which is constructed in the preprocessing phase. This line of work made PCP-based approaches more practical– very efficient if batching over a large number of computation instances, while requiring only standard cryptographic assumptions.

Parno et al. start another line of work [10, 14, 15] which is based on the recent finding of a new characterization of the NP complexity class –QSPs/QAPs [13], such as Pinocchio, which supports public verifiability and zero-knowledge. Similarly, by compiling programs to an innovative circuit representation [51], Ben-Sasson et al. provides another publicly verifiable and zero-knowledge scheme BCGTV [15, 16]. Our work inherits the property of the linear-PCP style designs and does not provide zero-knowledge, either.

Motivated by the delegation of computation (GKR [52]), IP systems provides the third line of work to assure the client that an untrusted prover has actually performed the correct computation (CRR [21], CMT [53]). In this line of work, Thaler also finds that circuits that satisfy a specific condition can have much lower cost on the pre-processing stage [54]. However, their circuits need to satisfy either a so-called "regular" wiring pattern condition or the "data-parallel" structure requirement (namely, a sub-computation is applied *independently* to different pieces of data). Besides, Thaler's work still has verifier's cost linear in the size of the circuit. All GKR-style systems typically require far more interaction, introducing much more bandwidth costs.

As an interesting hybrid-architecture protocol, Allspice [12] integrates both Zaatar and CMT in such a way that it automatically determines which one would be more efficient to verify the computation and runs the better of the two. Another study [55] has been pursuing argument systems that avoid a pre-processing phase for the verifier. Those argument systems are based on short PCPs, and existing work on this topic is still only theoretical.

Since state-of-the-art verification protocols are based on arithmetic circuits, the compiler which transforms the outsource task (typically a program written in high level languages such as C) into a circuit representation plays a very important role in this area. Zaatar's and Pinocchio's compilers map a large class of computations into corresponding arithmetic circuits, using special-purpose encodings. Another line of compilers, TinyRAM [15] and vnTinyRAM [16], use an innovative technique [51] and compile general C programs that have the same number of steps of executions to one same circuit.

This compiler has better performance for programs consisting of lots of memory accesses and control flow than Zaatar's and Pinocchio's, while the compilers of Zaatar and Pinocchio do better in programs which are close to “circuit” forms [16]. Walfish et al. evaluate these compilers in [56] and claim TinyRAM's circuit representation is orders of magnitude larger than the representation in Pinocchio and Zaatar.

CHAPTER 3. PRELIMINARIES

This is the opening chapter to my thesis which explains in general terms the concepts and assumptions which will be used in my thesis.

3.1 Probabilistically Checkable Proofs (PCPs)

In the PCP model, the verifier, a probabilistic polynomial-time (PPT) algorithm $\mathcal{V}$ can be convinced by a prover $\mathcal{P}$ that a string x belongs to a language L in an interactive way: $\mathcal{V}$ has random access to the proof π which is constructed by $\mathcal{P}$. By querying π (accessing the proof and reading several values), $\mathcal{V}$ will either accept or reject.

Correctness: If $x \in L$, $\mathcal{P}$ can always construct a proof π such that $\mathcal{V}$ will accept that $x \in L$. We call π the *correct proof* for x.

Soundness: If $x \notin L$ then for any π, the probability that $\mathcal{V}$ wrongly accepts is less than a constant ϵ. Let L be any language. The PCP theorem [22] [3] [2] [23] [24] guarantees that, if $L \in NP$, then with only a constant number of queries, $\mathcal{V}$ can verify $x \in L$ with negligible error probability (soundness).

Early results of PCP [23] [2] [3] [19] [25] [25] [5] [6] [26] were viewed as important discoveries only in the theory of computational complexity. Recent research focuses on the efficiency [27] [28], length [27] [29] [30] [24] [31] or soundness error [32] [33] [34] [35] of PCPs.

3.2 Homomorphic Encryption

The commitment protocols of existing efficient argument systems are all based on homomorphic encryption. This homomorphic encryption does not refer to fully-homomorphic encryption [17]. Only additive homomorphism is used in current argument systems: that is, the ciphertext of the result of adding two plaintexts can be efficiently computed from the ciphertexts of the two plaintexts. Formally, for any valid ciphertexts $c_1 = \texttt{Enc}(pk, m_1)$ and $c_2 = \texttt{Enc}(pk, m_2)$, there is an efficient algorithm $\mathcal{H}$ such that $\mathcal{H}(c_1, c_2) = \texttt{Enc}\ (pk, m_1 + m_2)$, where pk is the public key and m_1, m_2 are plaintexts. In general this does not mean $\texttt{Enc}(m1 + m2) = \texttt{Enc}(m1) + \texttt{Enc}(m2)$, and it is generally not feasible to have this relation without compromising security. In this dissertation, the underlying homomorphic encryption is assumed to be semantically secure [36].

3.3 Arguments

Arguments [4] are interactive proof systems, consisting of two PPT algorithms: the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. For an NP language L with soundness error $\epsilon(\cdot)$, an argument is both *complete* and *sound* if it satisfies the following conditions: (a) Completeness: for any $x \in L$ and corresponding NP witness w, the interaction between $\mathcal{V}(x)$ and $\mathcal{P}(x, w)$ leads $\mathcal{V}$ to accept the proof as true. (b) Soundness: for any $x \notin L$, and any efficient prover $\mathcal{P}^*$, the interaction between $\mathcal{V}(x)$ and $\mathcal{P}^*(x)$ leads $\mathcal{V}$ to accept the proof with probability less than $\epsilon(|x|)$.

3.4 Efficient Arguments without Short PCPs

To make argument systems efficient, current implementations rely on PCPs. However, PCP algorithms assume that the proof is computed by the prover, and fixed before the interaction with the verifier begins. The same assumption cannot be made in the context of argument systems. To bridge the gap between arguments and PCPs, an additional protocol is required, in which the prover commits to the proof before starting the PCP protocol with the verifier. Consequently, an argument is generally formed by joining together two protocols: a *PCP* and a *commitment*. Since the commitment protocol should maintain the efficiency of the argument, it is generally not feasible to require $\mathcal{P}$ to send the entire PCP proof to $\mathcal{V}$ due to the length of the proof. Two solutions can be implemented to overcome this obstacle: (1) make the PCP proof short, and (2) use cryptographic techniques to enable even shorter commitments to these short proofs. One of the first efforts in the latter direction is that of [26], which proposed to use a Merkle hash-tree construction to enable the prover to efficiently commit to the proof. Implicitly, the security of the protocol is bound to the security of the underlying hash function. To avoid the need for convoluted short PCP proofs, as well as the uncertain security of practical hashing primitives, [7] takes a new approach to argument systems: maintain a large (exponential-size) proof, and base the commitment on (computationally) provably-secure encryption primitives – public-key primitives.

The protocols of [7] are restricted to linear PCPs ([23], Section6). It is shown how SAT problems (Propositional Satisfiability Problems), formulated in the context of a boolean circuit, can be readily addressed by a simple linear PCP [7]. To form the argument system, [7] complemented the linear PCP with the notion of *commitment with linear decommitment*, which is instantiated with a simple public-key-based protocol. Since our work is closely related to that of [7], we will provide both the definition of *commitment with linear decommitment*, and a brief sketch of its instantiation in this section.

Definition 1. *Commitment with Linear Decommitment ([7]) A* commitment with linear decommitment *(in the context of argument systems) is a protocol between the prover $\mathcal{P}$ and verifier $\mathcal{V}$ – both modeled as interactive PPT algorithms – consisting of a commitment phase, and a decommitment phase, and aiming to securely commit the prover to a linear function $f_d : \mathbb{F}^n \to \mathbb{F}$ expressed as $f_d(z) = \langle d, z\rangle$, where $d, z \in \mathbb{F}^n$, and $\langle d, z\rangle$ is the natural inner (dot) product over vector spaces. In the commitment phase, an environment $\mathcal{E}$ gives*

$\mathcal{P}$ inputs d and $\mathbb{F}$, and gives $\mathcal{V}$ inputs $\mathbb{F}$ and the arity n. The interaction between $\mathcal{P}$ and $\mathcal{V}$ results in decommitment information z_P and z_V, respectively. In the decommitment phase, $\mathcal{E}$ gives $\mathcal{P}$ a decommitment query $q \in \mathbb{F}^n$. After further interaction between $\mathcal{P}$ and $\mathcal{V}$, the verifier $\mathcal{V}$ outputs either a value $a \in \mathbb{F}$, or the symbol $\perp$ (reject).

A commitment with linear decommitment *has the following properties. (a) Correctness: for any n and $\mathcal{E}$ generating d, q, at the end of the decommitment phase, the verifier outputs $a = f_d(q)$. (b) Binding: for the same decommitment information z_P, z_V (obtained after the commitment phase) and environment inputs q in the decommitment phase, the probability that at the end of the protocol the verifier outputs two different values (a_1, a_2) is negligible in n.*

Ishai et al. [7] took L as the satisfiability problem over an arithmetic circuit to show how to construct a correct proof for any arithmetic circuit and how to verify this circuit is satisfiable. Since this problem is NP-complete, every other NP problems can be deterministically and efficiently reduced to it. The PCP theorem guarantees that, if $L \in NP$ then with only constant number of queries, $\mathcal{V}$ can verify $x \in L$ with negligible error probability (soundness).

The instance x is an arithmetic circuit in this problem. For $x \in L$, there is a correct assignment z of the inputs to all gates in x. z can be also viewed as values of both the input of x and intermediate results. The correct proof is an exponential size PCP, which consists of two substrings. Each of the substrings can be viewed as a linear function: $\pi^{(1)} : \mathbb{F}^n \mapsto \mathbb{F}$ and $\pi^{(2)} : \mathbb{F}^{n^2} \mapsto \mathbb{F}$ where n is the length of a correct assignment z, $\pi^{(1)}(\cdot) = \langle z, \cdot \rangle$ and $\pi^{(2)}(\cdot) = \langle z \otimes z, \cdot \rangle$. Here, $\langle u, v \rangle$ denotes the inner product of two vectors u and v, and $u \otimes v$ denotes the outer product of two vectors u and v. The outer product is equivalent to a matrix multiplication uv^T, provided that u and v are both represented as a column vector. The whole proof string can be viewed as one single linear function $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$ such that $\pi(\cdot) = \langle z || z \otimes z, \cdot \rangle$ where $z || z \otimes z$ is the concatenation of the two vectors z and $z \otimes z$. When $\mathcal{V}$ sends the query q to π, he will get back $\pi(q)$. For

Table 3.1 Commitment Protocol of IKO [7]

Prover's Input: a vector $d \in \mathbb{F}^{n^2+n}$,
a linear function $\pi : \mathbb{F}^{n^2+n} \rightarrow \mathbb{F}$ where $\pi(q) = \langle q, d \rangle$.
Verifier's Input: arity $n^2 + n$, a PCP query q,
security parameter k for the homomorphic encryption.

Commitment Phase
Step 1: Verifier generates the key pair $(pk, sk) \leftarrow \texttt{Gen}(1^k)$.
Verifier randomly generates a vector $r \in_R \mathbb{F}^{n^2+n}$,
$r = (r_1, r_2, \cdots, r_{n^2+n})$, $r_i \in \mathbb{F}, i \in [n^2 + n]$.
Verifier encrypts each entry of the vector r using pk.
He sends to the prover:
$\texttt{Enc}(pk, r_1), \cdots, \texttt{Enc}(pk, r_{n^2+n})$ and pk.
Step 2: Prover makes use of the homomorphism of `Enc`
and gets $e = \texttt{Enc}(pk, \langle r, d \rangle)$. Prover sends e to Verifier.
Step 3: Verifier decrypts e and gets $s = \langle r, d \rangle = \texttt{Dec}(sk, e)$.
(s, r) will be kept for future decommitment.

Decommitment Phase
Step 4: Verifier picks at random a secret $\alpha \in_R \mathbb{F}$.
Step 5: Verifier sends $q, r + \alpha q$ to the prover.
Step 6: Prover responds with 2 values that are in $\mathbb{F}$:
(a, b) where a is supposed to be $\pi(q)$
and b is supposed to be $\pi(r + \alpha q)$.
Step 7: Verifier will determine whether $b = s + \alpha a$.
If it holds, the verifier will accept and output a;
otherwise it will reject and output $\perp$.

$q \in \mathbb{F}^n$, $\pi(q) = \langle z || z \otimes z, q || 0^{n^2} \rangle = \langle z, q \rangle$. For $q \in \mathbb{F}^{n^2}$, $\pi(q) = \langle z || z \otimes z, 0^n || q \rangle = \langle z \otimes z, q \rangle$. For $q \in \mathbb{F}^{n^2+n}$, $\pi(q) = \langle z || z \otimes z, q \rangle$.

As in Table 3.1, the commitment protocol was designed in [7], where a commitment to a proof is constructed and $\mathcal{V}$ can verify that the proof is indeed a linear function.

Once the proof is committed, $\mathcal{V}$ will check the proof in the linear PCP fashion. The verification consists of three kinds of tests. The first is the linearity test. $\mathcal{V}$ picks at random $q_1, q_2 \in \mathbb{F}^n$ and verifies $\pi(q_1) + \pi(q_2) = \pi(q_1 + q_2)$. The second is the quadratic consistency test. $\mathcal{V}$ picks at random $q_3, q_4 \in \mathbb{F}^n$ and verifies $\pi(q_3) \cdot \pi(q_4) = \pi(q_3 \otimes q_4)$. The third is the circuit correctness test. Each gate implies a constraint. For each constraint f_u, $(u = 1, 2, \cdots n)$, $\mathcal{V}$ picks at random a weight δ_u and constructs the weighted sum

$\sum_{u=1}^{n} \delta_u f_u$. The sum can be rewritten as $\pi(q_5) + c = 0$, $c \in \mathbb{F}$. If each constraint is satisfied, the weighted sum of the constraints $\pi(q_5) + c = 0$ is also satisfied. If there are some constrains not satisfied, the probability that the $\pi(q_5) + c = 0$ is $1/|\mathbb{F}|$. All these tests can be performed several times to drive the error probability down.

If we view the proof to be a function, then these commitment protocols ensure that once the prover commits to a certain function, later the answers to the verifier's queries are bounded to this function. However, it is not clear whether this function is linear. Since the correctness and soundness of all the linear PCP based argument systems– Pepper [8], Ginger [37], and Zaatar [10]– hold on the assumption that the committed proof is linear, mandatory tests for linearity have to be performed before genuinely checking whether the proof assures the correctness of the returned results. In the so-called linearity tests, the verifier picks at random queries q_1 and q_2, and verifies $\pi(q_1) + \pi(q_2) = \pi(q_1 + q_2)$, where $\pi(\cdot)$ is the committed function. It is pretty clear that if $\pi(\cdot)$ is linear, the equation must hold; if $\pi(\cdot)$ is not linear (viewed as being δ-close to a linear function) , the verifier will miss the non-linear part of $\pi(\cdot)$ with only a certain error probability. This test can be performed several times to drive the error probability down. A typical usage of linearity tests can be found in the first and the fourth steps of Zaatar as shown in Table 3.1, where $\mathcal{V}$ queries $\mathcal{P}$ with q_5, q_6, q_7 where $q_7 = q_5 + q_6$, and q_8, q_9, q_{10} where $q_{10} = q_8 + q_9$, expecting that the following holds: $\pi_W(q_7) = \pi_W(q_6) + \pi_W(q_5)$ and $\pi_W(q_{10}) = \pi_W(q_9) + \pi_W(q_8)$.

To make the test "smooth", a mechanism of self-correction is needed. It is the mechanism of self-correction that, conditioned on a proof being δ-close to a linear function, allows the calculation of the soundness for the argument systems. The mechanism is, when querying $\pi(\cdot)$ with query q, the verifier actually queries with q_0 and $q_0 + q$ instead of a single query q, and $\pi(q)$ is computed from $\pi(q_0 + q) - \pi(q_0)$. It is easy to see that the querying process in Zaatar uses the mechanism of self-correction. For instance, in order to get $\pi_W(q_A)$, the verifier queries the prover with q_5 and $q_1 = q_A + q_5$ instead of

q_A.

3.5 Two Recent Efficient Arguments: PEPPER and GINGER

Several recent works build upon the ideas developed in [7]. Of these, [8] and [9] are the most relevant to our work. To bring the protocol of [7] closer to practicality, [8] introduces a new protocol called PEPPER. It first shows that large savings in both computation and communication overhead can be achieved by expressing the SAT problem in the format of arithmetic circuits with *concise gates* [8] instead of the boolean circuits of [7]. In addition, by batching together multiple queries (to the same committed function), [8] can decommit all of them in a single commit-decommit round, rather than providing separate decommitments for each query.

In Ishai et al.'s original commitment design [7], one query is accompanied by an auxiliary query which is associated to a commitment. This requires many commitments, therefore increases the overhead. In [8], one auxiliary query is made, which is a random linear combination of all the PCP queries and the secret information that is associated to the commitment. In this design, one decommitment can guarantee many PCP queries are bound to the committed function. This sharply reduced the computational cost of generating the commitment information (although remaining cost is still very high.). The Single-Commit-Multi-Decommit design is demonstrated in Table 3.2.

Finally, by batching together multiple computations, [8] only requires a single random commitment query r for all computations involved (rather than a different r for each computation), hence achieving great savings in the encryption process – recall that the query r is transmitted to the prover after it has been encrypted by the homomorphic encryption algorithm.

Building on top of [8], additional improvements are provided in [9], in the context of a more efficient protocol called GINGER. For example, several queries of the quadratic

Table 3.2 The Single-Commit-Multi-Decommit Design [9]

$\mathcal{P}$'s Input: a vector $z \in \mathbb{F}^n$, a linear function $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$
where $\pi(\cdot) = \langle Z\|\|z \otimes z, \cdot\rangle$, n is the length of a correct assignment z.
$\mathcal{V}$'s Input: arity n, security parameter k of the encryption.
Commitment
Step 1: $\mathcal{V}$ generates the key pair: $(pk, sk) \leftarrow Gen(1^k)$.
$\mathcal{V}$ randomly generates a vector: $r = (r_1, r_2, \cdots, r_{n^2+n}) \in_R \mathbb{F}^{n^2+n}$.
$r_i \in \mathbb{F}, i = 1, 2, \cdots, n^2 + n$. $\mathcal{V}$ encrypts each entry of the vector r.
He sends $\texttt{Enc}(pk, r_1), \cdots, \texttt{Enc}(pk, r_{n^2+n})$ to $\mathcal{P}$.
Step 2: Using the homomorphism,
$\mathcal{P}$ gets: $e = \texttt{Enc}(pk, \langle r, z\rangle)$ $\mathcal{P}$ sends e to $\mathcal{V}$.
Step 3: $\mathcal{V}$ decrypts e. He gets $s = \langle r, z\rangle = Dec(sk, e)$.
Decommitment
Step 1: $\mathcal{V}$ picks μ secrets $\alpha_1, \cdots, \alpha_\mu \in \mathbb{F}$
$\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and $t = r + \alpha_1 q_1 + \cdots + \alpha_\mu q_\mu$.
Step 2: $\mathcal{P}$ returns $(a_1, \cdots, a_\mu, b)$
where $a_i = \pi(q_i)$ for $i = 1, \cdots, \mu$ and $b = \pi(t)$
Step 3: $\mathcal{V}$ checks whether $b = s + \alpha_1 a_1 + \cdots \alpha_\mu a_\mu$ holds.
If so, $\mathcal{V}$ outputs $a_1, \cdots, a_\mu$.
Otherwise, he rejects and output $\perp$.

correction test may be omitted [9] from the $\mathcal{V}$-to-$\mathcal{P}$ transmission, as they can be easily computed by the prover from the remaining quadratic correction queries.

Definition 2. A commitment to a function with multiple decommitments (CFMD)(from [8]) *A commitment to a function with multiple decommitments (CFMD) is defined by a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ (a sender and receiver, which correspond to our prover and verifier) anticipating the following experiment with an environment $\mathcal{E}$. $\mathcal{E}$ generates $\mathbb{F}, w$ and $Q = (q_1, \cdots, q_\mu)$. The two phases are:*

- Commitment phase: *$\mathcal{P}$ has w, and $\mathcal{P}$ and $\mathcal{V}$ interact, based on their random inputs.*
- Decommitment phase: *$\mathcal{E}$ gives Q to $\mathcal{V}$, and $\mathcal{P}$ and $\mathcal{V}$ interact again, based on further random inputs. At the end, $\mathcal{V}$ outputs $A = (a_1, \cdots, a_\mu) \in \mathbb{F}^\mu$ or $\perp$.*

A commitment to a function with multiple decommitments (CFMD) *should satisfy the following properties:*

- ***Correctness:*** *at the end of the decommitment phase, $\mathcal{V}$ outputs $\pi(q_i) = \langle w, q_i \rangle$, (for all i), if $\mathcal{P}$ is honest.*

- ϵ_B***-Binding:***. *Consider the following experiment. The environment $\mathcal{E}$ produces two (possibly distinct) μ-tuples of queries: $Q = (q_1, \cdots, q_\mu)$ and $\hat{Q} = (\hat{q}_1, \cdots, \hat{q_\mu})$. $\mathcal{V}$ and a cheating $\mathcal{P}^*$ run the commitment phase once and two independent instances of the decommitment phase. In the two instances $\mathcal{V}$ presents the queries as Q and $\hat{Q}$, respectively. We say that $\mathcal{P}^*$ wins if $\mathcal{V}$'s outputs at the end of the respective decommit phases are $A = (a_1, \cdots, a_\mu)$ and $\hat{A} = (\hat{a}_1, \cdots, \hat{a_\mu})$, and for some i, j, we have $q_i = \hat{q}_j$ but $a_i \neq \hat{a}_j$. The protocol holds the ϵ_B-Binding property if for all $\mathcal{E}$ and for all efficient $\mathcal{P}^*$, the probability of $\mathcal{P}^*$ winning is at most ϵ_B. The probability is taken over three sets of independent randomness: the commitment phase and the two runnings of the decommitment phase.*

3.6 Quadratic Programs

Recently Gennaro, Gentry, Parno and Raykova introduced a new characterization of the NP complexity class – the *Quadratic Span Programs (QSPs)* (and *Quadratic Arithmetic Programs (QAPs)*) [13, 14]. They showed that NP can be defined as the set of languages with proofs that can be efficiently verified by QSPs (or QAPs). Similarly to PCPs – another characterization of NP, which has already been widely used to obtain verifiable computation schemes – QSPs/QAPs are considered to be well-suited for verifiable computation and zero-knowledge schemes. One limitation of QSPs is that they inherently compute boolean circuits. But since arithmetic circuits are more natural and efficient in real-world computation tasks, we focus on QAPs, the counterpart of QSPs dealing with arithmetic circuit evaluation.

Definition 1. (Quadratic Arithmetic Programs [13]) *A QAP Q over field $\mathbb{F}$ contains 3 sets of $W' + 1$ polynomials: $\{A_w(t)\}$, $\{B_w(t)\}$, $\{C_w(t)\}$, for $w \in \{0, 1, \cdots, W'\}$, and* *a target polynomial $D(t)$. For function $\Psi : \mathbb{F}^N \mapsto \mathbb{F}^{N'}$, we say Q computes Ψ if the following holds: $(z_1, \cdots, z_{N+N'}) \in \mathbb{F}^{N+N'}$ is a valid assignment of Ψ's inputs and outputs, iff there exist coefficients $z_{N+N'+1}, \cdots, z_{W'}$ such that $D(t)$ divides $P(t)$, where $P(t) = \left(\sum_{w=1}^{W'} z_w \cdot A_w(t) + A_0(t)\right) \cdot \left(\sum_{w=1}^{W'} z_w \cdot B_w(t) + B_0(t)\right) - \left(\sum_{w=1}^{W'} z_w \cdot C_w(t) + C_0(t)\right)$. Namely, there exists a polynomial $H(t)$ such that $D(t) \cdot H(t) = P(t)$.*

Given an arithmetic circuit computing function Ψ, its corresponding QAP can be constructed by polynomial interpolation. Consider the set of circuit wires corresponding to the inputs and outputs of the circuit, and also the outputs of all multiplication gates. Each one of these wires is assigned three interpolation polynomials in Lagrange form, encoding whether the wire is a left input, right input, or output of each multiplication gate [13, 14]. The resulting set of polynomials is a complete description of the original circuit.

3.7 A Recent Efficient Argument System: Zaatar

The very recent work of [38] observes that QAPs can also be viewed as linear PCPs. By re-designing the PCP query generation and replacing the quadratic consistency checks and circuit correctness checks with the divisibility check of a QAP, Setty et al. successfully fit QAPs into the framework of Ginger [9]. The result is the novel protocol *Zaatar*, which significantly reduces the prover's workload. The key observation of Zaatar is that the evaluation of the polynomial $P(t)$ at the point $t = \tau$ can be simply written as: $P(\tau) = (\langle Z, q\rangle + A_0(\tau)) \cdot (\langle Z, q'\rangle + B_0(\tau)) - (\langle Z, q''\rangle + C_0(\tau))$, where $Z = (z_1, z_2, \cdots, z_{W'})$, $q = (A_1(\tau), A_2(\tau), \cdots, A_{W'}(\tau))$, $q' = (B_1(\tau), B_2(\tau), \cdots, B_{W'}(\tau))$, and $q'' = (C_1(\tau), \cdots, C_{W'}(\tau))$. Thus, $P(\tau)$ can be evaluated through three standard PCP queries to the dot product oracle $\pi_Z(\cdot) = \langle Z, \cdot\rangle$. If we represent the polynomials $H(t)$ explicitly: $H(t) = h_{|C_Z|}t^{|C_Z|} + \cdots + h_1 t + h_0$ (where C_Z is the set of constraints in Zaatar), similar observations on $H(\tau)$ can be made: $H(\tau) = \langle K_H, q_H\rangle$ where

$\mathcal{V}$ queries an oracle $\pi_R(\cdot) = \langle R, \cdot \rangle$, where $R = (z_{N+N'+1}, \cdots, z_{W'})$ is the intermediate results of the circuit computation.

–loop ρ times from 0 to 4:

0. Linearity queries generation. $\mathcal{V}$ selects $q_5, q_6 \in \mathbb{F}^{W'-(N+N')}$ and $q_8, q_9 \in \mathbb{F}^{|C_Z|+1}$. He takes $q_7 \leftarrow q_5 + q_6$ and $q_{10} \leftarrow q_8 + q_9$. Perform ρ_{lin} iterations in total.
1. QAP queries generation. $\mathcal{V}$ selects $\tau \in \mathbb{F}$ and takes $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^{|C_Z|})$, and $q_4 \leftarrow q_H + q_8$ and:

- $q_A \leftarrow (A_{(W')}(\tau), A_{(W'-1)}(\tau), \cdots, A_{(N+N'+1)}(\tau))$, and $q_1 \leftarrow (q_a + q_5)$.
- $q_B \leftarrow (B_{(W')}(\tau), B_{(W'-1)}(\tau), \cdots, B_{(N+N'+1)}(\tau))$, and $q_2 \leftarrow (q_b + q_5)$.
- $q_C \leftarrow (C_{(W')}(\tau), C_{(W'-1)}(\tau), \cdots, C_{(N+N'+1)}(\tau))$, and $q_3 \leftarrow (q_c + q_5)$.

2. Querying π_R. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{4+6\rho}$ and gets back $\pi_R(q_1), \pi_R(q_2), \cdots, \pi_R(q_{4+6\rho})$.
3. Linearity tests. Check whether following holds: $\pi_R(q_7) = \pi_R(q_6) + \pi_R(q_5)$, $\pi_R(q_{10}) = \pi_R(q_9) + \pi_R(q_8)$ and likewise for all other $\rho - 1$ iterations. If not, reject.
4. Divisibility test. $\mathcal{V}$ takes: $A_\tau = (\pi_R(q_1) - \pi_R(q_5) + \sum_{w=1}^{N+N'} z_w \cdot A_w(\tau) + A_0(\tau))$, $B_\tau = (\pi_R(q_2) - \pi_R(q_5) + \sum_{w=1}^{N+N'} z_w \cdot B_w(\tau) + B_0(\tau))$, $C_\tau = (\pi_R(q_3) - \pi_R(q_5) + \sum_{w=1}^{N+N'} z_w \cdot C_w(\tau) + C_0(\tau))$, and $\mathcal{V}$ checks whether the following equation holds: $D(\tau) \cdot (\pi_H(q_4) - \pi_H(q_8)) = A_\tau \cdot B_\tau - C_\tau$. If not, reject.

–If $\mathcal{V}$ makes it here, accept.

Figure 3.1 Zaatar's linear PCP based on QAPs

$K_H = (h_0, h_1, \cdots, h_{|C_Z|})$ and $q_H = (1, \tau, \tau^2, \cdots, \tau^{|C_Z|})$. Thus, $H(\tau)$ can also be evaluated through one PCP query to the oracle $\pi_H(\cdot) = \langle K_H, \cdot \rangle$. If Z consists of the input X with width $|X| = N$, output Y with width $|Y| = N'$ and intermediate results R with $|R| = W' - (N + N')$, then in order to guarantee that Y is the correct output when the input is X, the verifier needs to compute a part of $\langle Z, q \rangle$, and also a part of $\langle Z, q' \rangle$ and $\langle Z, q'' \rangle$, by himself. Consequently, $\mathcal{V}$ only queries the linear function oracle $\pi_R(\cdot) = \langle R, \cdot \rangle$, instead of $\pi_Z(\cdot)$. The detailed design (for one execution) of Zaatar is given in Figure 3.1[1].

[1]Note that the commitment/decommitment part is omitted for simplicity in Figure 3.1. Zaatar inherits the single-commit-multi-decommit protocol from Ginger [9].

CHAPTER 4. DELEGATION OF COMPUTATION WITH VERIFICATION OUTSOURCING: CURIOUS VERIFIERS

4.1 Problem Statement

4.1.1 System Model

In the context of cloud computing, we propose a computation architecture involving three different parties: the client $\mathcal{C}$, who is computationally weak, has computation tasks to be delegated to the cloud; the cloud server $\mathcal{P}$, who is computationally powerful, provides computing services to the client; the verifier $\mathcal{V}$, who is not required to be computationally powerful, provides verification services, helping $\mathcal{C}$ to check the results computed by $\mathcal{P}$. $\mathcal{P}$ also plays the role of the prover that attempts to convince $\mathcal{C}$ (through $\mathcal{V}$) that the result is correct.

The computation tasks are formalized into the *arithmetic circuit satisfiability problem* – i.e., the *Circuit-SAT* problem over an arithmetic circuit. This problem is NP-complete, hence any other NP problem can be deterministically and efficiently reduced to it. The reason we choose this arithmetic circuit version instead of the original Boolean Circuit-SAT is that most real-world computation tasks can be easily mapped to arithmetic circuits. Let x be a single-output, n-gate arithmetic circuit (n includes the input gates and the output gate). In the language of computational complexity, we can consider x as an instance of the *arithmetic circuit satisfiability problem*. Formally, $x \in L$ where L is the language of the arithmetic circuit satisfiability problem. If x is satisfiable for a given output value $E \in \mathbb{F}$, then there exists an input y of length $|y| = m$ to the circuit

x, which results in the circuit outputting E. This translates into a *correct assignment* z of the set of the outputs of all the gates in x. The assignment z can be in fact the concatenation of the input y with all the intermediate results inside the circuit, and has length $|z| = n$.

$\mathcal{C}$ is providing $\mathcal{P}$ with an output $E \in \mathbb{F}$, and expects $\mathcal{P}$ to return the input y which makes the circuit output E.

4.1.2 Threat Model

The threats faced by a client in our outsourced verification scenario come from malicious behaviors of both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. We assume $\mathcal{P}$ and $\mathcal{V}$ do not collude. This assumption is commonly used in multi-prover scenarios [18] [19] [20] [21]. Similar to previous proof systems, $\mathcal{P}$ can provide wrong responses to any queries, trying to cheat $\mathcal{C}$. In this chapter, we only address the problem caused by an "honest-but-curious" $\mathcal{V}$ – one that is interested in learning the computation task and/or the result, but performs the protocol faithfully. Different attack models such as dishonest $\mathcal{V}$ will be addressed in future work.

We need to point out that in all of our schemes we omit the authentication part, since authentication is a mature technology and it is not within the scope of our work. In our context, we assume that all parties are appropriately authenticated.

4.1.3 Design Goals

First of all, our proposed scheme should provide defense against the curious verifier $\mathcal{V}$ under the aforementioned model. To enable correct and efficient outsourcing of verification, the proposed scheme should satisfy the following requirements:

- *Correctness:* If y is the correct result, $\mathcal{P}$ can always construct a proof and convince the client $\mathcal{C}$ and the verifier $\mathcal{V}$ of the correctness of y.

- *Soundness:* If y is not the correct result, then for any proofs provided by a malicious $\mathcal{P}^*$, the probability that $\mathcal{V}$ wrongly accepts is negligibly small.

- *Efficiency:* The overall workload for the client $\mathcal{C}$ should be less – in an *amortized* sense (we will detail it later in Section 4.4) – than performing the verification himself. The workload for the verifier $\mathcal{V}$ should be comparable to that for verification in current two-parties designs [7, 8, 9]. Naturally, the workload for $\mathcal{V}$ should be far less than recomputing the result from scratch.

4.2 Basic Scheme: Verification Without Circuit Information

We present a basic version of our protocol in this section. We then bootstrap the process to develop the full solution in next section. In this basic scheme, the client $\mathcal{C}$ delegates the verification task to the verifier $\mathcal{V}$ and $\mathcal{V}$ can verify the proof without knowing the computational task, i.e., the underlying arithmetic circuit x. Our basic scheme is designed by joining together two protocols: a novel linear PCP and a new commitment protocol. Recall that PCP systems assume the proof is computed by $\mathcal{P}$, and fixed before the interaction with the $\mathcal{V}$ begins. The same assumption cannot be made in Cloud Computing. For efficiency reasons, it is also not feasible to require $\mathcal{P}$ to send the entire PCP proof to $\mathcal{V}$. It is the commitment protocol that guarantees $\mathcal{P}$ commits to the proof before starting the PCP protocol with $\mathcal{V}$.

4.2.1 A Building Block: A New Commitment Protocol

In the context of Circuit-SAT problem over an arithmetic circuit, we propose the following new commitment protocol for linear PCP system. It is a two-party protocol between the prover $\mathcal{P}$ and another party denoted by $\mathcal{C}/\mathcal{V}$ (as in "client/verifier"). We do not differentiate between the client $\mathcal{C}$ and the verifier $\mathcal{V}$ in this subsection. This separation will be done in next subsection. Recall that the Circuit-SAT problem is to

Table 4.1 Comparison of Commitment Protocols

Ishai et al. [7]	GINGER [9]	Our Basic Commitment Scheme
Commitment Phase	**Commitment Phase**	**Commitment Phase**
Prover's Input: a vector $d \in \mathbb{F}^{n^2+n}$, a linear function $\pi : \mathbb{F}^{n^2+n} \rightarrow \mathbb{F}$ where $\pi(q) = \langle q, d\rangle$.	**Prover's Input:** $z \in \mathbb{F}^n$, a linear function $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$ where $\pi(\cdot) = \langle z \Vert z \otimes z, \cdot\rangle$.	**Prover's Input:** a vector $z \in \mathbb{F}^n$, a linear function $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$ where $\pi(\cdot) = \langle z \Vert z \otimes z, \cdot\rangle$, n is the length of a correct assignment z.
Verifier's Input: arity n^2+n, security parameter k for the homomorphic encryption.	**Verifier's Input:** arity n, security parameter k of the encryption.	**Verifier's Input:** arity n, security parameter k of the encryption, the circuit x, the circuit's input $y = (y_1, \cdots, y_m)$ and output E.
Step 1: $\mathcal{V}$ generates the key pair $(pk, sk) \leftarrow \mathtt{Gen}(1^k)$ and $r = (r_1, \cdots, r_{n^2+n}) \in_R \mathbb{F}^{n^2+n}$, $r_i \in \mathbb{F}$, $i = 1, \cdots, n^2+n$. $\mathcal{V}$ encrypts each entry of r and sends $\mathtt{Enc}(pk, r_1), \cdots, \mathtt{Enc}(pk, r_{n^2+n})$ to $\mathcal{P}$.	**Step 1:** $\mathcal{V}$ generates the key pair $(pk, sk) \leftarrow \mathtt{Gen}(1^k)$ and $r = (r_1, \cdots, r_{n^2+n}) \in_R \mathbb{F}^{n^2+n}$. $r_i \in \mathbb{F}$, $i = 1, \cdots, n^2+n$. $\mathcal{V}$ encrypts each entry of r and sends $\mathtt{Enc}(pk, r_1), \cdots, \mathtt{Enc}(pk, r_{n^2+n})$ to $\mathcal{P}$.	**Step 1:** $\mathcal{C}/\mathcal{V}$ randomly picks $r_0 \in \mathbb{F}^n$ and $R_0 \in \mathbb{F}^{n^2}$ and constructs the corresponding commitments (s_0, S_0) according to Ishai et al.'s commitment protocol [7].
Step 2: $\mathcal{P}$ makes use of the homomorphism of $\mathtt{Enc}$ and gets $e = \mathtt{Enc}(pk, \langle r, d\rangle)$. $\mathcal{P}$ sends e to $\mathcal{V}$.	**Step 2:** Using the homomorphism, $\mathcal{P}$ gets: $e = \mathtt{Enc}(pk, \langle r, z\rangle)$ and sends it to $\mathcal{V}$.	**Step 2:** $\mathcal{C}/\mathcal{V}$ randomly generates an n-dimension weight vector $w_0 = (w_{01}, w_{02}, \ldots, w_{0n}) \in \mathbb{F}^n$, where each entry corresponds to a constraint f_u $(u = 1, \cdots, n)$ of the arithmetic circuit x. $\mathcal{C}/\mathcal{V}$ multiplies each constraint f_u of the circuit by w_{0u}, $u = 1, \cdots, n$ and constructs their summation as $\sum\limits_{u=1}^{n} w_{0u} f_u = \sum\limits_{u=1}^{m} w_{0u} y_u + w_{0n} E$. The summation of all these weighted constraints can be rewritten as $\langle R_1, z \otimes z\rangle + \langle r_1, z\rangle = c_0$, where $c_0 = \sum\limits_{u=1}^{m} y_u w_{0u} + w_{0n} E$.
Step 3: $\mathcal{V}$ decrypts e and gets $s = \langle r, d\rangle = \mathtt{Dec}(sk, e)$. (s, r) will be kept for decommitment.	**Step 3:** $\mathcal{V}$ receives e. He gets $s = \langle r, z\rangle = \mathtt{Dec}(sk, e)$. (s, r) will be kept for decommitment.	**Step 3:** Using R_1 and r_1, $\mathcal{C}/\mathcal{V}$ constructs the corresponding commitments (s_1, S_1) according to Ishai's commitment protocol [7].
Decommitment Phase	**Decommitment Phase**	**Decommitment Phase**
Prover's Input: d, π	**Prover's Input:** z, π, n	**Prover's Input:** x, z including y and E, π, n.
Verifier's Input: arity n^2+n, a PCP query q, decommitment information (r, s).	**Verifier's Input:** arity n, μ PCP queries $q_1, \cdots, q_\mu$, decommitment information (r, s).	**Verifier's Input:** n, μ PCP queries $q_1, \cdots, q_\mu$, decommitment information $(w_0, r_0, R_0, r_1, R_1, s_0, S_0, s_1, S_1)$.
Step 4: $\mathcal{V}$ picks at random a secret $\alpha \in_R \mathbb{F}$.	**Step 4:** $\mathcal{V}$ picks μ secrets $\alpha_1, \cdots, \alpha_\mu \in \mathbb{F}$	**Step 4:** $\mathcal{C}/\mathcal{V}$ generates randomly an n-dimension weight vector $w_1 = (w_{11}, \ldots, w_{1n}) \in \mathbb{F}^n$ and a secret $\alpha_1 \in \mathbb{F}$.
Step 5: $\mathcal{V}$ sends $q, r + \alpha q$ to the prover.	**Step 5:** $\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and $t = r + \sum\limits_{i=1}^{\mu} \alpha_i q_i$.	**Step 5:** $\mathcal{C}/\mathcal{V}$ queries $\mathcal{P}$ with vector w_1 and $w_2 = w_0 + \alpha_1 w_1$.
Step 6: $\mathcal{P}$ responds with 2 values that are in $\mathbb{F}$: (a, b) where a is supposed to be $\pi(q)$ and b is supposed to be $\pi(r + \alpha q)$.	**Step 6:** $\mathcal{P}$ returns $\mu + 1$ values: $(a_1, \cdots, a_\mu, b)$ where $a_i = \pi(q_i)$ for $i = 1, \cdots, \mu$ and $b = \pi(t)$	**Step 6:** From w_1, $\mathcal{P}$ constructs the weighted summation of all constraints just like what $\mathcal{C}/\mathcal{V}$ does in Step 2 and gets $\langle Q_0, z \otimes z\rangle + \langle q_0, z\rangle = \sum_{u=1}^{m} y_u w_{1u} + w_{1n} E$. From it, $\mathcal{P}$ learns Q_0 and q_0 and returns: $A_1 = \langle Q_0, z \otimes z\rangle$ and $a_1 = \langle q_0, z\rangle$. Similarly, from w_2, $\mathcal{P}$ constructs the weighted summation $\langle T_1, z \otimes z\rangle + \langle t_1, z\rangle = \sum_{u=1}^{m} y_u w_{2u} + w_{2n} E$ and learns T_1 and t_1. $\mathcal{P}$ returns: $B_1 = \langle T_1, z \otimes z\rangle$ and $b_1 = \langle t_1, z\rangle$.

Table 4.1 (Continued)

Comparison of Commitment Protocols

Ishai et al. [7]	GINGER [9]	Our Basic Commitment Scheme
Step 7: $\mathcal{V}$ will determine whether $b = s + \alpha a$. If it holds, the $\mathcal{V}$ will accept and output a; otherwise it will reject and output $\perp$.	**Step 7:** $\mathcal{V}$ checks whether $b = s + \alpha_1 a_1 + \cdots \alpha_\mu a_\mu$ holds. If so, $\mathcal{V}$ outputs $a_1, \cdots, a_\mu$. Otherwise, he rejects and output $\perp$.	**Step 7:** $\mathcal{C}/\mathcal{V}$ checks whether $b_1 = s_1 + \alpha_1 a_1$ and $B_1 = S_1 + \alpha_1 A_1$. If both hold, $\mathcal{C}/\mathcal{V}$ goes on to Step 8. Otherwise, $\mathcal{C}/\mathcal{V}$ rejects the proof.
		Step 8: $\mathcal{C}/\mathcal{V}$ randomly generates $\alpha_0, \alpha_1, \cdots, \alpha_\mu$ from $\mathbb{F}$. $\mathcal{C}/\mathcal{V}$ constructs $t = (r_1 \vert\vert R_1) + \alpha_0(r_0 \vert\vert R_0) + \sum_{k=1}^{\mu} \alpha_k q_k$.
		Step 9: $\mathcal{C}/\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and t.
		Step 10: $\mathcal{P}$ returns $\mu + 1$ corresponding answers $(a_1, \cdots, a_\mu, b_2)$ where for $k = 1, 2, \cdots \mu$, $a_k = \langle q_k, z \vert\vert z \otimes z \rangle$ and $b_2 = \langle t, z \vert\vert z \otimes z \rangle$.
		Step 11: $\mathcal{C}/\mathcal{V}$ checks whether $b_2 = (s_1 + S_1 + \alpha_0(s_0 + S_0)) + \sum_{k=1}^{\mu} \alpha_k a_k$ holds. If so, $\mathcal{C}/\mathcal{V}$ accepts. Otherwise $\mathcal{C}/\mathcal{V}$ rejects.

find an input $y = (y_1, y_2, \cdots, y_m)$ which makes the circuit x output a given value E. The arithmetic circuit x consists of $n = |x|$ arithmetic gates. Each gate implies a constraint $f_u, 1 \leq u \leq n$ as follows:

- $f_u(z_u) = z_u$ for $1 \leq u \leq m$. These are the constraints for the input gates.
- $f_u(z_i, z_j, z_k) = 0$ for $m+1 \leq u \leq n-1$, where f_u is a linear or quadratic polynomial of z_i, z_j, z_k. Here z_i, z_j, z_k are the two inputs and one output of a certain gate of x.
- $f_n(z_n) = E$. This is the constraint for the output gate.

Our commitment protocol rearranges the argument system to put the circuit-dependent portions inside the commitment phase (the offline stage). This approach not only simplifies the client/verifier's operation on the verification side, but also provides circuit-secrecy against the verifiers while outsourcing the verification tasks.

Our commitment protocol is demonstrated in the third column of Table 4.1.3. This protocol eventually includes two decommitment processes, one is from Step 4 to Step 7, the other is from Step 8 to Step 11.

We will prove that after the commitment construction phase, all of $\mathcal{P}$'s answers to later queries that pass both the decommitment checks are guaranteed to be bound to one single function (from queries to answers) with high probability. That is, having committed, $\mathcal{P}$ is very likely incapable of cheating the verifiers with fake answers. Moreover, this function is guaranteed to be linear with high probability.

Theorem 1. (*Main Theorem*) *For our commitment protocol, the following holds. For any environment $\mathcal{E}$, for any query q in either of the decommitment phases, the corresponding answer accepted by $\mathcal{V}$ at the end of the protocol is guaranteed to be the function value $\tilde{\pi}(q)$ except with probability less than $\frac{1}{|\mathbb{F}|} + neg(n)$, where $\tilde{\pi}(q)$ is a fixed function, $neg(n)$ is a negligible function, and the probability is over all randomness of $\mathcal{P}^*$ and $\mathcal{V}$ in all phases.*

We prove this theorem in Section 4.5.

4.2.2 A Delegation-of-Verification Scheme with Partial Circuit Confidentiality

As in the context of cloud computing, $\mathcal{C}$ sends the circuit description x to $\mathcal{P}$. After finding out the solution y with his powerful computation ability, $\mathcal{P}$ returns y to $\mathcal{C}$. Before outsourcing the verification task, $\mathcal{C}$ constructs the commitment according to our basic commitment scheme. $\mathcal{C}$ plays the role of $\mathcal{C}/\mathcal{V}$ in that commitment construction protocol and gets: w_0, r_1, R_1, r_0, R_0, s_1, S_1, s_0 S_0. The decommitment phases are a little bit different from our basic commitment scheme. $\mathcal{C}$ generates a random value $\alpha_0 \in \mathbb{F}$ and computes $(r_1||R_1) + \alpha_0(r_0||R_0)$. Then, $\mathcal{C}$ outsources the verification task to a third party $\mathcal{V}$. $\mathcal{C}$ sends w_0, $(r_1||R_1)+\alpha_0(r_0||R_0)$, s_1, S_1, $s_1+S_1+\alpha_0(s_0+S_0)$, E, and y_i's $(i = 1, \cdots m)$ to $\mathcal{V}$. Later, $\mathcal{V}$ will perform the decommitment.

For the linearity test, the idea is to check whether $y(x_1), y(x_2), y(x_1+x_2)$ (the answers to random queries x_1, x_2 and corresponding $x_1 + x_2$) satisfy $y(x_1) + y(x_2) = y(x_1 + x_2)$. This is detailed in [8],we omit it here.

For the circuit test, $\mathcal{V}$ generates w_1 and w_2 as in Step 4 and Step 5 of our basic commitment scheme. As in Step 5 and Step 6, $\mathcal{V}$ queries $\mathcal{P}$ with w_1 and w_2, receives back A_1, a_1, B_1 and b_1. Then $\mathcal{V}$ checks the equations as in Step 7. $\mathcal{V}$ will also check the circuit correctness, i.e., whether

$$A_1 + a_1 = \sum_{u=1}^{m} y_u w_{1u} + w_{1n} E. \tag{4.2.1}$$

For the quadratic consistency tests, $\mathcal{V}$ first conducts the second decommitment according to Steps 8, 9, 10, 11. Here, $\mathcal{V}$ uses only three testing queries (that is, $\mu = 3$) and the decommit query t. $\mathcal{V}$ randomly generates queries q_2, q_3 both from $\mathbb{F}^n$. He randomly generates $\alpha_2, \alpha_3, \alpha_4$, all from $\mathbb{F}$. He constructs the following queries: $q_4 = q_2 \otimes q_3$, and $t = (r_1||R_1 + \alpha_0(r_0||R_0)) + \sum_{i=2}^{3}(q_i||0^{n^2}) + \alpha_4(0^n||q_4)$, where 0^u is the u-dimension zero vector. $\mathcal{V}$ queries $\mathcal{P}$ with (q_2, q_3, q_4, t). $\mathcal{P}$ returns (a_2, a_3, a_4, b_2) where $a_2 = \langle q_2, z\rangle$, $a_3 = \langle q_3, z\rangle$, $a_4 = \langle q_4, z \otimes z\rangle$, $b_2 = \langle t, z \otimes z\rangle$. At the second decommitment, $\mathcal{V}$ checks whether $b_2 = (s_1 + S_1 + \alpha_0(s_0 + S_0)) + \sum_{i=2}^{4} \alpha_i a_i$. For quadratic consistency, $\mathcal{V}$ checks whether $a_4 = a_2 a_3$.

If all the checks pass, $\mathcal{V}$ will instruct $\mathcal{C}$ to accept. Otherwise, $\mathcal{V}$ instructs $\mathcal{C}$ to reject.

4.2.3 Theoretical Analysis: Correctness and Soundness

It is easy to see that without knowing the circuit x, $\mathcal{V}$ conducts all the PCP checks (except the linearity tests, since the commitment has provided linearity tests already) for $\mathcal{C}$. The correctness and soundness of this scheme follows directly from the linear PCP scheme. However, it should be noted that $\mathcal{V}$ has access to the pair $((r_1||R_1) + \alpha_0(r_0||R_0), s_1 + S_1 + \alpha_0(s_0 + S_0))$, which leaks information about the circuit. Hence the *partial circuit confidentiality* is afforded by this scheme. Nevertheless, building upon this scheme, full circuit confidentiality is achieved by the full solution outlined in the next section. Our basic scheme is also a distinct improvement over existing argument systems. During the verification procedure, the verifier does not need to read the circuit. He can generate all the queries with the cost of merely generating random numbers. By comparison, to generate a query, current argument systems need to both generate random numbers, and to calculate weighted summations of all circuit's constraints.

4.3 The Full Solution to Delegating Verification to a Curious Verifier

In certain scenarios, both the computation circuit and input/output of this circuit are sensitive. A curious verifier may be interested in information regarding the computation task (the circuit) and/or the circuit's input and output. In this section, we use the previously described basic scheme to develop the full solution against the curious verifier. The full version of the protocol consists of four phases, detailed in the following subsections: outsourcing computation, constructing commitments, outsourcing verification, and making a decision.

4.3.1 Outsourcing Computation Phase

$\mathcal{C}$ possesses an additive homomorphic cryptosystem. He generates a key pair (SK, PK). $\mathcal{C}$ sends the arithmetic circuit description x along with the public key PK to $\mathcal{P}$. After finding out the solution y with his powerful computation ability, $\mathcal{P}$ returns y to $\mathcal{C}$. After this computation, $\mathcal{P}$ obtains a correct assignment z of all the input of each gate in x. z can be viewed as the values of both y (the input of x) and intermediate results. $\mathcal{P}$ possesses a corresponding linear function: $\pi : \mathbb{F}^{n^2+n} \mapsto \mathbb{F}$ (remember $n = |x| = |z|$) such that, $\pi(\cdot) = \langle z||z \otimes z, \cdot\rangle$.

4.3.2 Constructing Commitments

Before outsourcing the verification task to a third party, $\mathcal{C}$ constructs the commitment according to the protocol described in Table 4.1.3. At the end of the construction, $\mathcal{C}$ possesses: w_0, r_1, R_1, r_0, R_0, and s_1, S_1, s_0, S_0. $\mathcal{C}$ generates a random value $\alpha_0 \in \mathbb{F}$ and computes $(r_1||R_1) + \alpha_0(r_0||R_0)$. After constructing the commitment, $\mathcal{C}$ randomly picks $w_{11}, w_{12}, \cdots, w_{1m}$ and w_{1n}, all in $\mathbb{F}$. Let w_1' be $((w_{11}, w_{12}, \cdots, w_{1m})||0^{n-m-1}||w_{1n})$. With these numbers, $\mathcal{C}$ computes $c_0 = \sum_{u=1}^{m} w_{1u}y_u + w_{1n}E$, then randomly generates α_1 and sends α_1, s_1,S_1, c_0, $w_0 + \alpha_1 w'_1$, $(r_1||R_1) + \alpha_0(r_0||R_0)$, $\texttt{Enc}(PK, (s_1 + S_1 + \alpha_0(s_0 + S_0)))$, and the public key PK to $\mathcal{V}$. Meanwhile, $\mathcal{C}$ sends $w_{11}, \cdots, w_{1m}$, $w_{1n}$and PK to $\mathcal{P}$.

4.3.3 Outsourcing Verification Phase

In this phase, $\mathcal{V}$ will verify the result without knowing the circuit and any of the assignments z (including y) in a PCP fashion. Given that the commit/decommit protocol has inherently provided the linearity test, it is sufficient to conduct only circuit tests and quadratic consistency tests.

The first step is the circuit satisfiability test. As in Section 4.2, $\mathcal{V}$ generates randomly two weight vectors. However, the vectors are a little bit different here: he generates $(w_{1(m+1)}, w_{1(m+2)}, \cdots, w_{1(n-1)})$, all from $\mathbb{F}$. Let w''_1 be $(0^m||(w_{1(m+1)}, w_{1(m+2)}, \cdots, w_{1(n-1)})||0)$. He generates w_2 as $w_2 = (w_0 + \alpha_1 w'_1) + \alpha_1 w''_1$, and queries $\mathcal{P}$ with w''_1 and w_2. This time, he will receive back $\texttt{Enc}(PK, A_1)$, $\texttt{Enc}(PK, a_1)$, $\texttt{Enc}(PK, B_1)$, $\texttt{Enc}(PK, b_1)$. Using PK, $\mathcal{V}$ computes $\texttt{Enc}$ $(PK, (s_1 + \alpha_1 c_0))$ from s_1, α_1 and c_0. Using the additive homomorphism of underlying encryption, $\mathcal{V}$ can compute $\texttt{Enc}(PK, (b_1 - ((s_1 + \alpha_1 c_0) + \alpha_1 a_1)))$ from $\texttt{Enc}(PK, (s_1 + \alpha_1 c_0))$, $\texttt{Enc}(PK, a_1)$, and $\texttt{Enc}(PK, b_1)$. $\texttt{Enc}(PK, (b_1 - ((s_1 + \alpha_1 c_0) + \alpha_1 a_1)))$ is denoted by $\texttt{Enc}(PK, plain_1)$. Similarly, he gets $\texttt{Enc}(PK, B_1 - (S_1 + \alpha_1 A_1))$, denoted by $\texttt{Enc}(PK, plain_2)$. Using the additive homomorphism of underlying encryption, from $\texttt{Enc}(PK, A_1)$ and from $\texttt{Enc}(PK, a_1)$, $\mathcal{V}$ computes $\texttt{Enc}(PK, A_1 + a_1)$, denoted by $\texttt{Enc}(PK, plain_3)$.

The second step is the quadratic consistency test. $\mathcal{V}$ randomly generates $\alpha_2, \alpha_3, \alpha_4$ all from $\mathbb{F}$. He randomly generates queries q_2, q_3 and constructs following queries: $q_4 = q_2 \otimes q_3$, $t = r_1||R_1 + \alpha_0(r_0||R_0) + \sum_{i=2}^{3} \alpha_i(q_i||0^{n^2}) + \alpha_4(0^n||q_4)$. $\mathcal{V}$ queries $\mathcal{P}$ with (q_2, q_3, q_4, t) and gets back $\texttt{Enc}(PK, a_2)$, $\texttt{Enc}(PK, a_3)$, $\texttt{Enc}(PK, a_4)$, and $\texttt{Enc}(PK, b_2)$ where $a_2 = \langle q_2,\ z\rangle$, $a_3 = \langle q_3, z\rangle$, $a_4 = \langle q_4, z \otimes z\rangle$, $b_2 = \langle t, z||z \otimes z\rangle$. We denote $\texttt{Enc}(PK, (b_2 - ((s_1 + S_1 + \alpha_0(s_0 + S_0)) + \sum_{i=2}^{4} \alpha_i a_i)))$ by $\texttt{Enc}(PK, plain_4)$, which can be computed from $\texttt{Enc}(PK, (s_1 + S_1 + \alpha_0(s_0 + S_0)))$, $\texttt{Enc}(PK, a_i)$'s $(i = 2, 3, 4)$ and $\texttt{Enc}(PK, b_2)$

using the homomorphism of the underlying cryptosystem. Then, $\mathcal{V}$ randomly generates four random numbers from $\mathbb{F}$: $\theta_1, \cdots, \theta_4$, and constructs $\texttt{Enc}(PK, \sum_{i=1}^{4} plain_i \cdot \theta_i)$ from $\texttt{Enc}(PK, plain_i)$'s using the homomorphism. $\mathcal{V}$ sends $\texttt{Enc}\ (PK, \sum_{i=1}^{4} plain_i \cdot \theta_i)$ to $\mathcal{C}$ with $\texttt{Enc}(PK, a_4)$, $\texttt{Enc}(PK, a_3)$, and $\texttt{Enc}(PK, a_2)$ to $\mathcal{C}$.

We adopt the a variant of the El Gamal cryptosystem [8] as our additive homomorphic encryption. The cryptosystem operates over a group G of prime order q. $g \in G$ is a random generator of G. The ciphertext of m is $\texttt{Enc}(pk, m) = (g^k, g^{m+ak})$ where $(pk, sk) = (g^a, a)$. It is easy to see this encryption is an additive homomorphic encryption. The ciphertext of the sum of two plaintexts could be constructed as: $\texttt{Enc}(pk, m_1 + m_2) = \texttt{Enc}(pk, m_1) \cdot \texttt{Enc}(pk, m_2) = (g^{k_1+k_2}, g^{m_1+m_2+a(k_1+k_2)})$. The third step is to perform linearity tests. Since the underlying cryptosystem is addition homomorphic, it is easy to check the linearity homomorphically.

4.3.4 Making A Decision

$\mathcal{C}$ first decrypts $\texttt{Enc}(PK, \sum_{i=1}^{4} plain_i \cdot \theta_i)$ and determines whether the plaintext is 0. If not, $\mathcal{C}$ will reject. Otherwise $\mathcal{C}$ decrypts $\texttt{Enc}(PK, a_2)$, $\texttt{Enc}(PK, a_3)$ and $\texttt{Enc}(PK, a_2)$. Then, $\mathcal{C}$ determines whether $a_2 a_3 = a_4$. If so, he will accept that y is the correct solution of his computational task.

4.3.5 Security Analysis

Theorem 2. *(Correctness) If the arithmetic circuit x is satisfiable, then a prover $\mathcal{P}$ with the knowledge of the correct input y is able to make the client $\mathcal{C}$ accept y by performing our protocol.*

Proof. It is easy for a prover $\mathcal{P}$ who has found out the correct result y of the computation task to find out the correct assignment $z = (z_1, z_2, \cdots, z_n)$ including the correct result y and all the intermediate results of the circuit. That is, z satisfies all the constraints $f_u, u = 1, \cdots, n$. If $\mathcal{P}$ responds with correct values as in the protocol, all corresponding test equations (unencrypted version) will hold. Given the encryption used in this section is additive homomorphic, the ciphertexts of all the corresponding linear combinations are ciphertexts of 0. The conclusion follows. □

Definition 3. *We say that a verification protocol for the arithmetic circuit satisfiability problem x wins* λ-confidentiality *if the following properties are satisfied. In the context of computational complexity, x can be represented as a binary string, and so can the correct assignment z. Let $P_x, P_z : \{0,1\}^* \to \{0,1\}$ be arbitrarily-defined predicates, extracting one bit of information about binary strings of arbitrary length. For every probabilistic polynomial-time algorithm A, for every possible transmitted messages $m = (m_1, \cdots, m_k)$ with $k = |m| = poly(\lambda)$, for every positive polynomial $p(\cdot)$, for every P_x, P_z, we have:*

$$Pr[A(1^\lambda, m, 1^{|z|}, commt) = P_x(x))] - \frac{1}{2} < \frac{1}{p(\lambda)} \tag{4.3.1}$$

$$Pr[A(1^\lambda, m, 1^{|z|}, commt) = P_z(z))] - \frac{1}{2} < \frac{1}{p(\lambda)} \tag{4.3.2}$$

where $commt$ is the commitment information provided by $\mathcal{C}$ before verification. (The probability is over z as well as over the internal coin tosses of either algorithms.)

Theorem 3. *(Confidentiality) If the underlying homomorphic encryption in our protocol has the security parameter λ, then our verification protocol for the arithmetic circuit satisfiability problem wins λ-confidentiality.*

Proof. Recall that the commitment is $commt = (\alpha_1, s_1, S_1, c_0, w_0 + \alpha_1 w'_1, (r_1||R_1) + \alpha_0(r_0||R_0), \texttt{Enc}(PK, (s_1 + S_1 + \alpha_0(s_0 + S_0)))$. Given that all transmitted messages $m = (m_1, m_2, \cdots, m_k)$ are encrypted in our protocol, for every probabilistic polynomial-time algorithm A there exists a probabilistic polynomial-time algorithm A^* such that

$$\begin{aligned} &Pr[A(1^\lambda, m, 1^{|z|}, commt) = x_i)] \\ &< Pr[A^*(1^\lambda, 1^{|z|}, H) = x_i] + \frac{1}{p(\lambda)} \end{aligned} \tag{4.3.3}$$

where $H = (\alpha_1, s_1, S_1, c_0, w_0 + \alpha_1 w'_1, (r_1||R_1) + \alpha_0(r_0||R_0))$. This follows directly from an appropriate formulation of semantic security ([57], Def. 5.2.1) of the underlying homomorphic encryption. Now since $(r_0||R_0)$ is uniformly random, $(r_1||R_1)+\alpha_0(r_0||R_0))$ is random and independent of $(r_1||R_1)$ by the crypto lemma. Therefore, s_1, S_1 is the response of the prover to a query (on z) which is random and independent of $(\alpha_1, c_0, w_0 + \alpha_1 w_1', (r_1||R_1) + \alpha_0(r_0||R_0))$. This implies that the entire H is independent of x and z, and hence for any algorithm A^*, $Pr[A^*(1^\lambda, 1^{|z|}, H) = x_i] \leq \frac{1}{2} + \frac{1}{p(\lambda)}$ The inequalities in (4.3.1) and (4.3.2) follow. □

Theorem 3 implies that for sufficiently large λ, the advantage of an adversary finds out the circuit and the results in the execution of the protocol is negligible.

Theorem 4. *(soundness) The client will accept a wrong answer with probability less than* $\leq \frac{4}{|\mathbb{F}|} - \frac{1}{|\mathbb{F}|^2} + (1-(3\delta - 6\delta^2))^k$ *where the tests guarantee the proof is* δ*-close to linear and* k *is the number of iterations of the linearity tests.*

Proof. For any given wrong proof $\tilde{\pi}$ which consists of $\tilde{\pi}^{(1)}$ and $\tilde{\pi}^{(2)}$, we define the following events:

$$E_0 = \{\tilde{\pi} \text{ are accepted}\}$$

$$E_1 = \{\tilde{\pi} \text{ is not committed}\}$$

$$E_2 = \{\tilde{\pi} \text{ is not linear}\}$$

$$E_3 = \{\tilde{\pi}^{(1)} \text{ and } \tilde{\pi}^{(2)} \text{ are not quadratic consistent}\}$$

$$E_4 = \{\tilde{\pi} \text{ is not circuit correct}\}$$

If an incorrect answer is accepted by the client, then $\tilde{\pi}$ passes four kinds of tests: decommitment (DT, denoted by $Test_1$), linearity test (LT, denoted by $Test_2$), quadratic consistency test (QT, denoted by $Test_3$), and the circuit correctness test (CT, denoted by $Test_4$). Thus,

$$\begin{aligned}
&Pr[E_0]\\
=&Pr[(E_1 \text{ OR } E_2 \text{ OR } E_3 \text{ OR } E_4) \text{ AND passing 4 tests}]\\
=&Pr[\bigcup_{i=1}^{4}(E_i \text{ AND passing 4 tests})]\\
\leq&Pr[\bigcup_{i=1}^{4}(E_i \text{ AND passing } Test_i)]\\
\leq&\sum_{i=1}^{4} Pr[(E_i \text{ AND passing } Test_i)]
\end{aligned}$$

Analysis of the commitment/decommitment protocol [9] shows that

$$Pr[\tilde{\pi} \text{ is not commited AND passing DT}] \leq \frac{1}{\mathbb{F}}. \tag{4.3.4}$$

The commitment/decommitment protocol also executes the linearity test [59]. It is shown ([59]) that

$$Pr[\tilde{\pi} \text{ is } \delta \text{ close to linear AND passing LT}] \leq (1-(3\delta-6\delta^2))^k. \tag{4.3.5}$$

where k is the number of iterations of the linearity tests.

Each quadratic consistence test trial will reject the wrong proof with probability at least $\frac{(|\mathbb{F}|-1)^2}{|\mathbb{F}|^2}$ [7]. Thus, each quadratic consistence test will wrongly accept with probability at most $1-\frac{(|\mathbb{F}|-1)^2}{|\mathbb{F}|^2}=\frac{2|\mathbb{F}|-1}{|\mathbb{F}|^2}$.

Each circuit correctness test trial will wrongly accept a proof with probability at most $\frac{1}{|\mathbb{F}|}$ [7]. Thus,

$$\begin{aligned}
Pr[E_0] &\leq \frac{1}{|\mathbb{F}|}+(1-(3\delta-6\delta^2))^k+(\frac{2|\mathbb{F}|-1}{|\mathbb{F}|^2})+(\frac{1}{|\mathbb{F}|})\\
&=\frac{4}{|\mathbb{F}|}-\frac{1}{|\mathbb{F}|^2}+(1-(3\delta-6\delta^2))^k
\end{aligned} \tag{4.3.6}$$

This holds for any incorrect $\tilde{\pi}$. Given that for a wrong answer y, any proof $\tilde{\pi}$ is incorrect, the client will accept this wrong answer with probability less than $\frac{4}{|\mathbb{F}|}-\frac{1}{|\mathbb{F}|^2}+(1-(3\delta-6\delta^2))^k$ where the tests guarantee the proof is δ-close to linear and k is the number of iterations of the linearity tests. □

4.4 Practical Use and Complexity Analysis

4.4.1 Amortized Query Costs

Our proposed schemes can amortize query costs. That is, our designs are able to use the same commitment/decommitment queries and PCP queries across many instances of the same circuit (with different input/output). The same commitment/decommitment queries and PCP queries make sure that it is still infeasible for $\mathcal{P}$ to know the secret commit query and provide uncommitted responses. (It is known that if we fix a given instance, the probability of wrongly accepting will not be influenced by other instances [8].)

The amortizing usage is as follows.

1. There is an off-line stage. In this stage, $\mathcal{C}$ reads the circuit and constructs the commitment queries and sends to $\mathcal{P}$. This is done only once.

2. $\mathcal{P}$ possesses β proofs (linear functions) $\tilde{\pi}_1, \cdots, \tilde{\pi}_\beta$, one for each instance. For commitment construction, $\mathcal{P}$ will return β tuples of commitments, each of which is as in the previous section.

3. For each instance, $\mathcal{P}$ computes the results.

4. In the verification phase, for the *same* query tuple $q_1, \cdots, q_4, t$ from $\mathcal{V}$, $\mathcal{P}$ will response β tuples of answers, each for one instance. For each instance, $\mathcal{V}$ verifies results as in the previous section. Totally, $\mathcal{V}$ runs β decommitments, β circuit tests, and β quadratic consistency tests, one for each instance.

We give a practical use example here. Suppose $\mathcal{C}$ has a large number of computational tasks. All these tasks can be reduced to an instance of the Circuit-SAT problem with the same circuit. Before using the Cloud server to do computing, $\mathcal{C}$ will generate the commitment queries: w_0, r_1, R_1, r_0, R_0. This is an off-line stage and it runs only once for

Table 4.2 Comparison of Complexity for Each Instance

		Computation	**Communication**
Our Basic Scheme	$\mathcal{C}$'s cost	0	$m + O(1)$
	$\mathcal{V}$'s cost	$m \cdot Mult + m \cdot Add + \frac{1}{\beta}[n^2 \cdot Mult + (4n) \cdot RNG]$	$\frac{1}{\beta}O(n^2)$
Our Full Solution	$\mathcal{C}$'s cost	$(m+1) \cdot Mult + m \cdot Add + Enc + 3Dec$	$m + O(1)$
	$\mathcal{V}$'s cost	$poly(\lambda) \cdot Oper + O(1) \cdot Mult + O(1) \cdot Add + \frac{1}{\beta}[n^2 \cdot Mult + (4n) \cdot RNG]$	$\frac{1}{\beta}O(n^2)$
Re-computing		$\geq [poly(n) \cdot Mult + poly(n) \cdot Add]$	0
NP-proof		$poly(n) \cdot Mult + poly(n) \cdot Add$	n
Linear PCP (Ishai et al.)		$poly(n) \cdot Mult + poly(n) \cdot Add$	$O(n^2)$
GINGER		$[(m+1) \cdot Mult + m \cdot Add] + \frac{1}{\beta}[poly(n) \cdot Mult + poly(n) \cdot Add + (n^2 + 2n) \cdot RNG]$	$\frac{1}{\beta}O((n+m)^2)$

all instances. The on-line stage is as in Section 4.3. First, $\mathcal{C}$ gives the computing tasks to $\mathcal{P}$. Secondly, $\mathcal{P}$ computes the tasks and gives back the results and the commitments to $\mathcal{C}$. After choosing a verifier $\mathcal{V}$, $\mathcal{C}$ outsources the verification to $\mathcal{V}$. $\mathcal{V}$ advises $\mathcal{C}$ to accept or reject.

4.4.2 Complexity Analysis

Our design meets the efficiency goal outlined in Section 4.1.3. As in [58], we are ignoring the time of the offline stage, since the cost of generating the commit/decommit queries can be amortized over many instances. We compare the computational cost and the communication cost of $\mathcal{C}$ between our protocol and other related work in Table 4.2. In this table, $Mult$ and Add are the cost of multiplication and addition in $\mathbb{F}$. RNG is the cost of generating a random number in $\mathbb{F}$. $Oper$ is the cost of the additive homomorphic operation. In our design, the computational and communication complexity of $\mathcal{C}$ for each instance is $O(m)$, (m is the length of results y) and is not dependent on the circuit size n. This is much more efficient than all current two-party verification schemes. We observe that $\mathcal{V}$ is also very efficient. *Even the cost of both $\mathcal{C}$ and $\mathcal{V}$ combined is less than the verifier's cost in the state-of-the-art argument systems that rely only on standard cryptographic assumptions.*

4.5 Mathematical Proof of the Commitment

We prove the main theorem through several lemmas and other theorems. Our idea is to prove that, if combining our commitment construction protocol with either of our decommitment phases, the answers of the prover is guaranteed to be bound to a function. Then, we prove the two functions are actually the same function.

For combining our commitment construction protocol with either of our decommitment phases, we provide a Lemma stating that efficiently solving a certain cryptographic problem is infeasible. Then, we show that if the prover's answers are not bound, we can construct an efficient algorithm to solve that hard cryptographic problem. The binding property follows.

Specifically, we first prove that first decommitment protocol checks will guarantee the prover's output is bound to a function $\tilde{\sigma}$ with high probability (Theorem 6 and Corollary 7). Otherwise an efficient algorithm can solve an infeasible cryptographic problem that contradicts Lemma 5. Then, we prove the second decommitment protocol checks will guarantee the prover's output is bound to a function $\tilde{\rho}$ with high probability (Theorem 9 and Corollary 10). Otherwise an efficient algorithm will contradict Lemma 8. Lastly, we prove if we combine the first and the second decommitments, the prover's output is guaranteed to be bound to a single function $\tilde{\pi}$ with high probability (Theorem 13). Otherwise we can construct an efficient algorithm that contradicts Lemma 12. That is, with high probability $\tilde{\sigma}(q) = \tilde{\rho}(q) \triangleq \tilde{\pi}(q)$.

Lemma 5. *Let X be the tuple*

$$(pk, \texttt{Enc}(pk, R_1 || r_1), \Phi, w_1, w_0 + \alpha_1 w_1, w_0 + \alpha_1' w_1).$$

For every efficient algorithm $\mathcal{A}$, we have

$$Pr[\mathcal{A}(X) = \alpha_1] \leq \frac{1}{|\mathbb{F}|} + neg(n) \tag{4.5.1}$$

where $R_1 \in \mathbb{F}^{n^2}$, $r_1, w_0, w_1 \in \mathbb{F}^n$, $\alpha_1, \alpha_1' \in \mathbb{F}$, and Φ is the constraint function that R_1, r_1, w_0 satisfy, and neg() is a negligible function of its input. The probability is over all w_0 as well as the randomness of functions `Enc` *and* `Gen` *of the underlying cryptosystem.*

Proof. From the semantic security ([57], Def. 5.2.2), for every efficient $\mathcal{A}$, there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that

$$\begin{aligned} &Pr[\mathcal{A}(X) = \alpha_1] \\ \le &Pr[\mathcal{A}'(\Phi, w_1, w_0 + \alpha_1 w_1, w_0 + \alpha_1' w_1) = \alpha_1] + neg(n) \end{aligned} \tag{4.5.2}$$

Given that the constraint function Φ is independent on the tuple $w_1, w_0 + \alpha_1 w_1, w_0 + \alpha_1' w_1$, the knowledge of Φ will not help any function $\mathcal{A}'$ to figure out α_1. Thus, for every probabilistic polynomial-time algorithm $\mathcal{A}'$ we have

$$\begin{aligned} &Pr[\mathcal{A}'(\Phi, w_1, w_0 + \alpha_1 w_1, w_0 + \alpha_1' w_1) = \alpha_1] \\ = &Pr[\mathcal{A}'(w_1, w_0 + \alpha_1 w_1, w_0 + \alpha_1' w_1) = \alpha_1] \\ \le &1/|\mathbb{F}| \end{aligned} \tag{4.5.3}$$

The conclusion follows from 4.5.2 and 4.5.3.

□

Theorem 6. *Our commitment construction protocol combined with the first decommitment protocol is a commitment with linear decommitment.*

Proof. The correctness is straightforward. Now we prove the binding property.

For any environment $\mathcal{E}$ and efficient malicious prover $\mathcal{P}^*$, define the following experiment. $\mathcal{P}^*$ plays the role of $\mathcal{P}$. Now, invoke the decommitment phase with $\mathcal{P}^*$ and $\mathcal{V}$ twice, using the same commitment phase and w_1, but different α_1. We say $\mathcal{P}^*$ wins if $\mathcal{V}$ accepts two distinct values a_1, a_1', or accepts two distinct values A_1, A_1'. Assume $\mathcal{P}^*$ wins with non-negligible probability. Then, there is an environment $\mathcal{E}$ and an efficient malicious prover $\mathcal{P}^*$ with a weight vector w_1 such that for random pair α_1, α_1', $\mathcal{P}^*$ returns

(a_1, a'_1) and (A_1, A'_1) to $\mathcal{V}$ and makes him accept at least one of the two pairs with non-negligible probability. We show that this fact implies that we can construct an algorithm $\mathcal{A}$ that will contradict Lemma 5. Let w_2 be the value of $w_0 + \alpha_1 w_1$ and w'_2 be the value of $w_0 + \alpha'_1 w_1$. When $(pk, \texttt{Enc}(pk, w_0), w_2, w'_2)$ is input, $\mathcal{A}$ executes following operations, corresponding to the steps of the protocol.

1. $\mathcal{A}$ gives $\mathcal{P}^*$ $(pk, \texttt{Enc}(pk, w_0))$.
2. $\mathcal{P}^*$ gives back two values (e_1, E_1) which $\mathcal{A}$ ignores.
3. $\mathcal{A}$ gives $\mathcal{P}^*$ (w_1, w_2) and receives back (a_1, b_1) and (A_1, B_1).
4. $\mathcal{A}$ gives $\mathcal{P}^*$ (w_1, w'_2) and receives back (a'_1, b'_1) and (A'_1, B'_1).

From $w_2 = w_0 + \alpha_1 w_1$ and $w'_2 = w_0 + \alpha'_1 w_1$, $\mathcal{A}$ can figure out $(\alpha_1 - \alpha'_1) w_1 = w_2 - w'_2$. Given that there must be a index j such that $w_{1j} \neq 0$ (otherwise $w_1 = 0$ and the protocol need not to be invoked), $\mathcal{A}$ can have

$$\alpha_1 - \alpha'_1 = (w_{2j} - w'_{2j})/w_j \tag{4.5.4}$$

Then, from $b_1 = s_1 + \alpha_1 a_1$, $b'_1 = s_1 + \alpha_1 a'_1$, $B_1 = S_1 + \alpha_1 A_1$, and $B'_1 = S_1 + \alpha_1 A'_1$, $\mathcal{A}$ can have

$$b_1 - b'_1 = \alpha_1 a_1 - \alpha'_1 a'_1 \tag{4.5.5}$$

$$B_1 - B'_1 = \alpha_1 A_1 - \alpha'_1 A'_1 \tag{4.5.6}$$

From the assumption above, at least one of the following holds with non-negligible probability: $a_1 \neq a'_1$ and $A_1 \neq A'_1$. Without loss of generality, let us say $a_1 \neq a'_1$ holds with non-negligible probability. Then, $\mathcal{A}$ can compute α_1 from (4.5.4) and (4.5.5). this contradicts Lemma 5. □

Corollary 7. *For our commitment construction protocol combined with the first decommitment, the following holds. For any environment $\mathcal{E}$, any accepted output of $\mathcal{V}$ at the* *end of the decommitment phase is guaranteed to be a function* $\tilde{\sigma}(q) = \texttt{Ext}(v_{\mathcal{P}^*}, v_{\mathcal{V}}, q)$ *except with negligible probability in* n*, where* $v_{\mathcal{P}^*}, v_{\mathcal{V}}$ *are the views of* $\mathcal{P}^*$ *and* $\mathcal{V}$ *in the commitment phase,* q *is the query in decommitment phase generated by* $\mathcal{E}$*,* `Ext` *is a (possibly inefficient) extractor which given the views of* $\mathcal{P}^*$ *and* $\mathcal{V}$ *in the commitment phase "extracts" a function to which* $\mathcal{P}^*$ *is committed, and the probability is over all randomness of* $\mathcal{P}^*$ *and* $\mathcal{V}$ *in both phases.*

Proof. Given that our commitment construction protocol combined with the first decommitment protocol is a commitment with linear decommitment (Theorem 6), the Corollary follows directly from Lemma 3.2 of [7]. □

Lemma 8. *Let* X *be the tuple*

$$\{pk, \texttt{Enc}(pk, R_1||r_1), \texttt{Enc}(pk, R_0||r_0), (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha q, (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha' q\}$$

For every $q \in \mathbb{F}^{n+n^2}$ *and every efficient algorithm* $\mathcal{A}$*, we have*

$$Pr[\mathcal{A}(X) = \alpha] \leq \frac{1}{|\mathbb{F}|} + neg(n) \tag{4.5.7}$$

where $R_1, R_0 \in \mathbb{F}^{n^2}$*,* $r_1, r_0 \in \mathbb{F}^n$*,* $\alpha, \alpha' \in \mathbb{F}$*. The probability is over all* $R_1||r_1$*,* $R_0||r_0$*,* $\alpha, \alpha', \alpha_0$*, as well as the randomness of functions* `Enc` *and* `Gen` *of the underlying cryptosystem. Note:* $R_0||r_0$ *is uniformly distributed in* $\mathbb{F}^{n+n^2}$*; however,* $R_1||r_1$ *is a* punctured *vector in* $\mathbb{F}^{n+n^2}$*. Namely,* $R_1||r_1$ *have many fixed components with the value* 0*.*

Proof. From the semantic security ([57], Def. 5.2.2), for every efficient $\mathcal{A}$, there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that

$$\begin{aligned} & Pr[\mathcal{A}(X) = \alpha] \\ \leq & Pr[\mathcal{A}'((R_1||r_1) + \alpha_0(R_0||r_0) + \alpha q, (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha' q) = \alpha] \\ & + neg(n) \end{aligned} \tag{4.5.8}$$

For every probabilistic polynomial-time algorithm $\mathcal{A}'$ we have

$$Pr[\mathcal{A}'((R_1||r_1) + \alpha_0(R_0||r_0) + \alpha q, (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha' q) = \alpha] \leq 1/|\mathbb{F}| \tag{4.5.9}$$

The conclusion follows from 4.5.8 and 4.5.9. □

Theorem 9. *Our commitment construction protocol combined with the second decommitment is a commitment to a function with multiple decommitments (CFMD) with* $\epsilon_B = 1/|\mathbb{F}| + neg(n)$.

Proof. The correctness is straightforward. Now, we prove the binding property. We will show that if $\mathcal{P}^*$ can systematically cheat, then there exists an efficient algorithm that takes

$$\{pk, \texttt{Enc}(pk, R_1||r_1), \texttt{Enc}(pk, R_0||r_0),$$
$$(R_1||r_1) + \alpha_0(R_0||r_0) + \alpha q, (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha' q\}$$

as input and outputs α with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. It contradicts Lemma 8.

Suppose the binding property does not hold. Then, there exists an environment $\mathcal{E}$ and an efficient malicious prover $\mathcal{P}^*$. $\mathcal{E}$ produces $(q_1, q_2, \cdots, q_\mu)$ and $(q'_1, q'_2, \cdots, q'_\mu)$ such that there exist indices i and j satisfying $q_i = q'_j \triangleq q$. $\mathcal{P}^*$ returns $a_1, a_2, \cdots, a_\mu$ and $a'_1, a'_2, \cdots, a'_\mu$ among which $a_i \neq a'_j$ to $\mathcal{V}$, and makes $\mathcal{V}$ accept them with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. The probability is over the randomness of the commit phase and of two runnings of the decommit phase.

We show that the existence of i, j implies that we can construct an algorithm $\mathcal{A}$ that will contradict Lemma 8. For any (α, α'), let v be the value of $(R_1||r_1) + \alpha q$ and v' be the value of $(R_1||r_1) + \alpha' q$. When

$$\{pk, \texttt{Enc}(pk, R_1||r_1), \texttt{Enc}(pk, R_0||r_0), v, v', R_0||r_0\}$$

is input, $\mathcal{A}$ executes following operations according to the protocol.

1. $\mathcal{A}$ gives $\mathcal{P}^*$: $(pk, \texttt{Enc}(pk, R_1||r_1), \texttt{Enc}(pk, R_0||r_0))$.

2. $\mathcal{P}^*$ gives back two values (e_1, e_2) which $\mathcal{A}$ ignores.

3. $\mathcal{A}$ randomly generates $(\alpha_1, \cdots \alpha_{i-1}, \alpha_{i+1}, \cdots, \alpha_\mu)$ and $(\alpha'_1, \cdots \alpha'_{j-1}, \alpha'_{j+1}, \cdots, \alpha'_\mu)$;

4. $\mathcal{A}$ constructs two queries $v + \sum_{k \in [\mu] \setminus i} \alpha_k q_k$ and $v' + \sum_{k \in [\mu] \setminus j} \alpha'_k q'_k$.

5. $\mathcal{A}$ gives the two queries constructed in the last step to $\mathcal{P}^*$ and receives back two values (b, b').

6. $\mathcal{A}$ gives $\mathcal{P}^*$ $(q_1, q_2, \cdots, q_\mu)$ and $(q'_1, q'_2, \cdots, q'_\mu)$. $\mathcal{A}$ receives back $a_1, a_2, \cdots, a_\mu$ and $a'_1, a'_2, \cdots, a'_\mu$.

From $v = (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha q$ and $v' = (R_1||r_1) + \alpha_0(R_0||r_0) + \alpha' q$, $\mathcal{A}$ can figure out $(\alpha - \alpha')q = v - v'$. Given that there must be an index k such that $q_k \neq 0$ (otherwise $q = (0, \cdots, 0)$ and the protocol need not to be invoked), $\mathcal{A}$ can have

$$\alpha - \alpha' = (v_k - v'_k)/q_k \tag{4.5.10}$$

Then, from $b = s_1 + \alpha_0 s_0 + \alpha a_i + \sum_{k \in [\mu] \setminus i} \alpha_k a_k$, $b' = s_1 + \alpha_0 s_0 + \alpha' a'_j + \sum_{k \in [\mu] \setminus j} \alpha'_k a'_k$, $\mathcal{A}$ can have

$$\begin{aligned} &\alpha a_i - \alpha' a'_j \\ =&(b - b') - (\sum_{k \in [\mu] \setminus i} \alpha_k a_k - \sum_{k \in [\mu] \setminus j} \alpha'_k a'_k) \end{aligned} \tag{4.5.11}$$

From the assumption above, $a_i \neq a'_j$ holds with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. Then, $\mathcal{A}$ can compute α from equations (4.5.10) and (4.5.11). It contradicts Lemma 8.

□

Corollary 10. *For our commitment construction protocol combined with the second decommitment, the following holds. For any environment $\mathcal{E}$, any accepted output of $\mathcal{V}$ at the end of the decommitment phase is guaranteed to be a function $\tilde{\rho}(q)$ except with negligible probability in n, where q is the query in decommit phase generated by $\mathcal{E}$, and the probability is over all randomness of $\mathcal{P}^*$ and $\mathcal{V}$ in both commit and decommit phases.*

Proof. This follows directly from Theorem 9 and Lemma B.2 of [8]. □

To prove our final Theorem, we will prepare two Lemmas: Lemma 11 and Lemma 12. Lemma 12 is a theoretical result based on Lemma 11. We show in the proof of our final Theorem that we will be able to construct a contradiction against Lemma 12 if $\mathcal{P}^*$ can systematically cheat.

Lemma 11. *Let X be the tuple*

$$\{pk, \mathtt{Enc}(pk, R_1||r_1), (R_1||r_1) + \alpha q, (R_1||r_1) + \alpha' q\}$$

where $R_1 \in \mathbb{F}^{n^2}$, $r_1 \in \mathbb{F}^n$, $\alpha, \alpha' \in \mathbb{F}$. However, $R_1||r_1$ is a punctured *vector in $\mathbb{F}^{n+n^2}$ as it is constructed in our protocol. Namely, $R_1||r_1$ have many fixed components with the value* 0*.*

Then, for every q that is constructed in the same way as $R_1||r_1$ (namely, q has the same fixed components 0 *as $R_1||r_1$), and every efficient algorithm $\mathcal{A}$, we have*

$$Pr[\mathcal{A}(X) = \alpha] \leq \frac{1}{|\mathbb{F}|} + neg(n), \tag{4.5.12}$$

The probability is over all $R_1||r_1$, α, α', as well as the randomness of functions `Enc` *and* `Gen` *of the underlying cryptosystem.*

Proof. From the semantic security ([57], Def. 5.2.2), for every efficient $\mathcal{A}$, there exists a probabilistic polynomial-time algorithm $\mathcal{A}'$ such that

$$\begin{aligned} &Pr[\mathcal{A}(X) = \alpha] \\ \leq &Pr[\mathcal{A}'((R_1||r_1) + \alpha q, (R_1||r_1) + \alpha' q) = \alpha] \\ +&neg(n) \end{aligned} \tag{4.5.13}$$

For every probabilistic polynomial-time algorithm $\mathcal{A}'$ we have

$$\begin{aligned} &Pr[\mathcal{A}'((R_1||r_1) + \alpha q, (R_1||r_1) + \alpha' q) = \alpha] \\ \leq &1/|\mathbb{F}| \end{aligned} \tag{4.5.14}$$

It follows that $Pr[\mathcal{A}(X) = \alpha] \leq \frac{1}{|\mathbb{F}|} + neg(n)$ □

Lemma 12. *Let X be the tuple $\{pk, \texttt{Enc}(pk, R_1||r_1), (R_1||r_1)+\alpha q, (R_1||r_1)+\alpha' q\}$. Then, for any (probably unknown) function $f(\cdot) : \mathbb{F}^{n+n^2} \mapsto \mathbb{F}$, any vector $R_0||r_0$ ($R_0||r_0 \neq R_1||r_1$), and s (the correponding value of $f(R_0||r_0)$), for every q that is constructed in the same way as $R_1||r_1$ (namely, q has the same fixed components 0 as $R_1||r_1$), and every efficient algorithm $\mathcal{A}$, we have:*

$$Pr[\mathcal{A}(X, R_0||r_0, s) = \alpha] \leq \frac{1}{|\mathbb{F}|} + neg(n), \qquad (4.5.15)$$

where $R_1, R_0 \in \mathbb{F}^{n^2}$, $r_1, r_0 \in \mathbb{F}^n$, $\alpha, \alpha' \in \mathbb{F}$. The probability is over all $R_1||r_1$, α, α', as well as the randomness of functions `Enc` *and* `Gen` *of the underlying cryptosystem. Note: $R_0||r_0$ is uniformly distributed in $\mathbb{F}^{n+n^2}$; however, $R_1||r_1$ is a* punctured *vector in $\mathbb{F}^{n+n^2}$. Namely, $R_1||r_1$ have many fixed components with the value 0.*

Proof. Since $R_1||r_1$, $R_0||r_0$, and the function $f(\cdot)$ are independent, the information of $R_0||r_0$ and s will not help any algorithm $\mathcal{A}$ to solve α. The conclusion follows directly from Lemma 11. □

Theorem 13. *For our commitment construction protocol combined with both the first and second decommitment, the following holds. For any environment $\mathcal{E}$, any accepted output of $\mathcal{V}$ at the end of the decommitment phase in each protocol, is guaranteed to be bound to the same function $\tilde{\pi}(q)$ except with probability less than $\frac{1}{|\mathbb{F}|} + neg(n)$, where q is the query in either of the decommitment phases generated by $\mathcal{E}$, and the probability is over all randomness of $\mathcal{P}^*$ and $\mathcal{V}$ in all phases. That is, for all q's, $\tilde{\sigma}(q) = \tilde{\rho}(q) \triangleq \tilde{\pi}(q)$ holds except with probability less than $\frac{1}{|\mathbb{F}|} + neg(n)$, where $\tilde{\sigma}(q)$ and $\tilde{\rho}(q)$ are defined in Corollary 7 and Corollary 10.*

Proof. We will show that if $\mathcal{P}^*$ can systematically output $\tilde{\sigma}(q) \neq \tilde{\rho}(q)$, then there exists an efficient algorithm that takes

$$\{pk, \texttt{Enc}(pk, R_1||r_1), (R_1||r_1) + \alpha q, (R_1||r_1) + \alpha' q, R_0||r_0, s\}$$

as input and outputs α with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. where $(R_1||r_1), \alpha, \alpha', q$ and $R_0||r_0, s$ are as in Lemma 12. That contradicts Lemma 12.

Suppose there is a malicious prover $\mathcal{P}^*$ which can systematically output $\tilde{\sigma}(q) \neq \tilde{\rho}(q)$ with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. Then, there exists an environment $\mathcal{E}$ that produces a function $f(\cdot)$ that $f(R_0||r_0) = s$, q_1' and $(q_1, q_2, \cdots, q_\mu)$ such that there exists a index i satisfying $q_1' = q_i \triangleq q$, and $\mathcal{P}^*$ returns a_1' (for q_1') and $a_1, a_2, \cdots, a_\mu$ (for $(q_1, q_2, \cdots, q_\mu)$), where $a_1' \neq a_i$, to $\mathcal{V}$ and makes $\mathcal{V}$ accept them with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. The probability is over the randomness of the commit phase and of two runnings of the decommit phase. Note: in our protocol, q_1' is not sent explicitly. Instead, q_1' was sent implicitly through the weight vector w_1. Thus, q_1' is a "punctured" vector in $\mathbb{F}^{n+n^2}$, with many fixed components of value 0. Correspondingly, q_i is also "punctured" vector in $\mathbb{F}^{n+n^2}$, with many fixed components of value 0.

We show that the existence of i implies that we can construct an algorithm $\mathcal{A}$ that will contradict Lemma 12. For random (α, α'), let v be the value of $(R_1||r_1) + \alpha q$ and v' be the value of $(R_1||r_1) + \alpha' q$. When $\{pk, \texttt{Enc}(pk, R_1||r_1), v, v', R_0||r_0, s\}$ is input, $\mathcal{A}$ executes following operations according to the protocol.

1. $\mathcal{A}$ encrypts $R_0||r_0$ to $\texttt{Enc}(pk, R_0||r_0)$ and gives $\mathcal{P}^*$: $(pk, \texttt{Enc}(pk, R_1||r_1), \texttt{Enc}(pk, R_0||r_0))$.

2. $\mathcal{P}^*$ gives back two values (e_1, e_2) which $\mathcal{A}$ ignores.

3. $\mathcal{A}$ randomly generates $(\alpha_0, \alpha_1, \cdots \alpha_{i-1}, \alpha_{i+1}, \cdots, \alpha_\mu)$ and α';

4. $\mathcal{A}$ constructs two queries $v + \alpha_0(R_0||r_0) + \sum_{k\in[\mu]\setminus i} \alpha_k q_k$ and v'.

5. $\mathcal{A}$ gives $\mathcal{P}^*$ $v + \alpha_0(R_0||r_0) + \sum_{k\in[\mu]\setminus i} \alpha_k q_k$ and v'. and receives back two values (b, b').

6. $\mathcal{A}$ gives $\mathcal{P}^*$ $(q_1, q_2, \cdots, q_\mu)$ and q_1'. $\mathcal{A}$ receives back $a_1, a_2, \cdots, a_\mu$ and a_1'.

From $v = (R_1||r_1) + \alpha q$ and $v' = (R_1||r_1) + \alpha' q$, $\mathcal{A}$ can figure out $(\alpha - \alpha')q = v - v'$. Given that there must be a index k such that $q_{(k)}$, q's kth entry is non-zero (otherwise

$q = (0, \cdots, 0)$ and the protocol need not to be invoked), $\mathcal{A}$ can have

$$\alpha - \alpha' = (v_{(k)} - v'_{(k)})/q_{(k)} \tag{4.5.16}$$

Then, from $b = s_1 + \alpha_0 s_0 + \alpha a_i + \sum_{k\in[\mu]\setminus i} \alpha_k a_k$, $b' = s_1 + \alpha' a'_1$, $\mathcal{A}$ can have

$$\alpha a_i - \alpha' a'_1 = (b - b') - (\alpha_0 s_0) - (\sum_{k\in[\mu]\setminus i} \alpha_k a_k) \tag{4.5.17}$$

From the assumption above, $a_i \neq a'_1$ holds with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. Then, $\mathcal{A}$ can compute α from equations (4.5.16) and (4.5.17) with probability more than $\frac{1}{|\mathbb{F}|} + neg(n)$. It contradicts Lemma 12.

□

The proof of the Main Theorem (Theorem 1) follows directly from Theorem 13.

4.6 Conclusions and Future Directions

In Cloud Computing, a client outsources computation to a more powerful server – the prover. To ensure the correctness of the results returned by the prover, the client has to perform a verification stage that is often tedious and expensive. In this work, we introduce the idea of delegation of verification in Cloud Computing. This natural approach relieves the client from performing the verification of the outsourced-computation results by outsourcing it to a third party – the verifier. We propose the first scheme that provides efficient outsourcing of the verification, while at the same time preserving the confidentiality of both the computational task and its result from untrusted verifiers. Given that the computational tasks are not limited to a specific computational problem, it appears at a first glance that the fully-homomorphic encryption is necessary for hiding the computational task. However, by means of combining a novel commitment protocol and the linear PCP system with only additive homomorphic encryption, our design enables a honest-but-curious third party to perform the bulk of the verification procedure, without gaining access to information about the original computational task or its result. We are currently investigating delegation of verification with a verifier who does not perform the protocol faithfully – a *curious and lazy verifier.*

CHAPTER 5. VERIFIABLE COMPUTATION WITH REDUCED INFORMATIONAL COSTS AND COMPUTATIONAL COSTS

5.1 Motivation

The state-of-the-art Zaatar [10] showed the connection between Linear PCP and QAP. However, unlike other QAP-based designs [13] [14], Zaatar relies only on standard cryptographic assumptions. It applies QAP into the framework of PCP and generates a novel verifiable computation scheme. Moreover, as an appealing verifiable computation scheme, Zaatar makes the prover more efficient than any other PCP-based designs. However, Zaatar does not bring major improvements to the verifier's computational cost. As in the recent PCP-based works [7] [60] [8] [9], once the prover has committed to the proof, the most computationally-intensive part for the verifier in Zaatar is the generation of queries. The high costs of the verifier are hence lowered by reusing some of the queries for multiple instances of the *same* problem – or *batching*. For computational tasks that can tolerate large batch sizes, the costs of verification in Zaatar can be driven down by amortizing. However, for tasks that require low investment on the verification and tolerate only small batch sizes, a new, more efficient protocol is needed.

Besides the computation and communication costs that have been concerned in existing research of verifiable computation, another type of verification cost has been generally ignored until now. We call this the *informational cost* – the cost associated with information required for verification on both sides. The verifier's information required for verification usually consists of verification keys and the full knowledge of the computation task. The prover's informational costs usually consists of the proof vector.

The informational cost generally has a strong impact on the adoption of the verification algorithms. On one hand, the size of the required information directly influences the memory cost for verification, and the speed of verification. For the memory cost, verifiers in existing research keep the required information in a large memory and frequently access it. For the speed, the length of the proof vector determines the cost of generating, and responding to, the queries while verifying. On the other hand, the informational cost implies the privacy/confidentiality issues. One obvious risk is that, storing this information itself introduces potential leakage of sensitive information about the computational task and its results. A more serious risk occurs in the context of the third party verification. (For example, disputes between the server and the client can be solved by an arbitrator who plays the role of the third-party verifier. Similar verifications may be required by government agencies, nonprofit organizations, and consumer safety organization, for the purpose of quality evaluation, project management, etc.) In such scenarios, information cost implies the computation task and its results being delivered to the third party. However, once delivered, the information is out of the control of the client and the client can never call them back. This security issue, in turn, may limit the outsourcing of verification.

As far as informational costs are concerned, all recent PCP-based works [7] [60] [8] [9], [10] require the verifier to have full knowledge of the computation circuit while performing the verification. Note: zero-knowledge property in proof systems refers to the requirement that nothing is known except the answer to the verifier. However, the computation are known by both parties.

5.1.1 Our Contributions

In this chapter, we introduce **RIVER**, a **R**educed-**i**nvestment **ver**ified computation protocol, whose improvement further enhances the practicality of argument systems in verifiable computation. Our contributions are summarized as follows:

- RIVER reduces the verifier's workload that needs to be amortized. Namely, the number of batched instances of the protocol, required for amortization, will largely decrease. Instead of batching over instances of the same circuit as in existing works, RIVER makes more parts of costs amortized over instances of *all different circuits of the same size*. We model the costs and compare the costs using a typical computation such as matrix multiplication, showing that RIVER is 28% better than state-of-the-art Zaatar at the verifier side.

- As a side effect, RIVER increases the prover's non-amortized cost and amortized cost. However, the introduced non-amortized part is negligible. Although RIVER introduces amortized cost to the prover side, this cost can be amortized over instances of *all different circuits of the same size.*

- RIVER reduces the informational cost of the verifier, by removing the requirement that the verifier has to access the circuit description during query generating. Thus, a third-party verifier can help generating the queries without knowing the computation task details.

- RIVER adopts one of our theoretical findings. We show that under certain assumptions, the Single-Commit-Multi-Decommit protocol provides the inherent linearity tests. Thus, a modified Single-Commit-Multi-Decommit protocol will make the linearity tests obsolete and reduce verification costs.

5.2 System Model

In the context of cloud computing, we propose a computation architecture involving two parties: the client $\mathcal{V}$, who is computationally weak, has computation tasks to be delegated to the cloud; the cloud server $\mathcal{P}$, who is computationally powerful, provides computing services to the client. The computation tasks are formalized into the *arithmetic circuit* – i.e., the computation task is performed over an arithmetic circuit. This is pretty natural, since arithmetic circuits can be easily mapped to real-world computation tasks[1]. Let Ψ be a $|\Psi|$-gate arithmetic circuit. The client $\mathcal{V}$ is providing the prover $\mathcal{P}$ with Ψ and input $X \in \mathbb{F}^n$, and expects $\mathcal{P}$ to return the correct output $Y \in \mathbb{F}^{n'}$. Then $\mathcal{P}$ tries to convince $\mathcal{V}$ that Y is correct. $\mathcal{P}$ will hold a proof Z, which is a *correct assignment* Z –the concatenation of the input X, output Y with all the intermediate results W inside the circuit, $(Z = X||Y||W)$ and has length $|Z| = m$, where W is the intermediate result vector $W \in \mathbb{F}^{m-n-n'}$ of the circuit Ψ.

5.3 A Technique: A Commitment Providing Inherent Linearity Tests

In the line of linear-PCP fashion verifiable computation designs, once the prover is committed to a proof, the verifier has to perform laborious linearity tests to ensure the proof is linear. In fact, the number of queries required to perform linearity tests dominate the number of overall queries of the protocol. Thus, the cost caused by linearity test is still one of the bottlenecks of current protocols. Up to now, in the context of Single-Commit-Multi-Decommit protocol (refer to Section 3.5), whether the linearity tests are necessary was still an open question. In this section, we propose our theoretical result, showing that under an assumption, the Single-Commit-Multi-Decommit protocol will

[1]Existing compilers can turn high-level programs into arithmetic circuits [9], [10], [14]. For simplicity, we omit these techniques.

provides inherent linearity tests. Thus, if linear PCP is combined with this commitment protocol, the linearity tests are obsolete. We will adopt the following theoretical results in our protocol design and thus achieve cost savings.

Theorem 14. *The Single-Commit-Multi-Decommit protocol ensures that, if the secret commit information is generated by the prover using an affine function (analytically defined), (or equivalent to the cases that is generated by the verifier himself), for all query tuples, unless the sender (or prover) replies to all queries with the same* linear function *and he knows the analytic description of this linear function, the prover will not pass the decommitment test except with probability* $\frac{1}{|\mathbb{F}|} + \epsilon_S$*, where the probability is over the randomness of the prover and the verifier in the decommitment phase.*

Proof. Let $\pi()$ denote the proof in the PCP sense. The prover knows that, when the verifier generates the commitment information $\pi(r)$, he uses the linear function $\texttt{F}_1(\cdot)$, such that $\pi(r) = \texttt{F}_1(r)$. We claim that in this scenario, the prover has to answer all queries with the same linear function $\texttt{F}_1$ – otherwise the probability that the prover passes the decommitment test is less than $\frac{1}{|\mathbb{F}|} + \epsilon_S$.

To prove this, we assume that there exists a PPT prover $\mathcal{P}^*$, and queries $q_1, q_2, \cdots, q_\mu$, such that once committed, with these queries, the probability that $\mathcal{P}^*$ answers the μ queries with a function $f(\cdot)$ – such that there exists at least one index k for which $f(q_k) \neq \texttt{F}_1(q_k)$ – and passes the decommitment test, is more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$, where the probability is over the randomness of the prover and the verifier in the decommitment phase.

We can now modify $\mathcal{P}^*$ and make it into an algorithm $\mathcal{P}^\dagger$ which can solve the problem stated in Lemma 15 with probability more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$:

1. $\mathcal{P}^\dagger$ has inputs: $\texttt{Enc}(\texttt{r}), r + \alpha q_k, q_k$
2. $\mathcal{P}^\dagger$ uses $\texttt{Enc}(\texttt{r})$ as inputs and runs $\mathcal{P}^*$'s commitment phase.
3. $\mathcal{P}^*$ outputs with $\texttt{Enc}(\texttt{F}_1(r)$ which $\mathcal{P}^\dagger$ will neglect.

4. $\mathcal{P}^\dagger$ produces a set of coefficients $\{\alpha_1, \cdots, \alpha_{k-1}, \alpha_{k+1}, \cdots, \alpha_{\mu-1}, \alpha_\mu\}$, and runs $\mathcal{P}^*$'s decommitment phase with following: $\{q_1, q_2, \ldots, q_{\mu-1}, q_\mu, (r + \alpha q_k) + \sum_{i=1}^{k-1} \alpha_i q_i + \sum_{i=k+1}^{\mu} \alpha_i q_i\}$..

5. $\mathcal{P}^*$ outputs $\{f(q_1), \ldots, f(q_\mu), \mathtt{F}_1(r) + \alpha f(q_k) + \sum_{i=1}^{k-1} \alpha_i f(q_i) + \sum_{i=k+1}^{\mu} \alpha_i f(q_i)\}$. This will pass the decommitment test with probability more than $\frac{1}{|\mathbb{F}|} + \epsilon_S$.

6. Having access to $\{\alpha_1, \cdots, \alpha_{k-1}, \alpha_{k+1}, \cdots, \alpha_{\mu-1}, \alpha_\mu\}$, and $f(q_1), \ldots, f(q_\mu)$, $\mathcal{P}^\dagger$ can now obtain an equation of the form $\mathtt{F}_1(r) + \alpha f(q_k) = b$ (where $b \in \mathbb{F}$ is easily calculated).

Recall that $\mathcal{P}^\dagger$ has knowledge of $r + \alpha q_k$, which yields a group of n linearly-independent linear equations in the form of $r + \alpha q_k = a$. Given that $\mathtt{F}_1(q_k) \neq f(q_k)$, the equation $\mathtt{F}_1(r) + \alpha f(q_k) = b$ is linearly independent of the former n equations $r + \alpha q_k = a$. Thus, $\mathcal{A}$ can solve for α from these $n + 1$ linearly independent equations. This will contradict Lemma 15:

Lemma 15. *(from [7]) For any probabilistic polynomial time algorithm $\mathcal{A}$, any $q \in \mathbb{F}^n$, and any uniformly-randomly picked $r \in \mathbb{F}^n$ we have $Pr[\mathcal{A}(\mathtt{Enc(r)}, r + \alpha q, q) = \alpha] \leq \frac{1}{|\mathbb{F}|} + \epsilon_S$, where ϵ_S is from the semantic security.*

□

5.4 A Reduced-Investment Verifiable Computation Protocol: RIVER

In this section, we introduce RIVER(*reduced-investment verifiable computation protocol*), an improvement of Zaatar [10], aimed at reducing the amortized cost of the verifier – or equivalently, the number of instances required before amortization can be considered complete. We accomplishes this by deferring some of the verifier's amortizable computation to the prover. In doing so, two other benefits are achieved as side effects. First, the overall cost for the verifier is decreased when compared to Zaatar (and implicitly also to Ginger). This is despite deferring some of the amortized computation to the prover (the deferred part is almost negligible when compared to the construction of the proof). Second, RIVER enables the verifier to generate queries independently – that is, the query generating stage does not require full knowledge of the circuit. We detail RIVER as follows.

5.4.1 PCP Querying

Our main observation is that the PCP query generation in Zaatar is somewhat redundant. RIVER removes the redundancy by employing three rounds of PCP querying and one round of decision making. The logical procedure is demonstrated in Figure 5.1.

5.4.2 PCP Querying of RIVER

Let $l = |C_R|$. We represent the QAP polynomials $A_i(t)$, $B_i(t)$, $C_i(t)$, with $i = 0, 1, \cdots, m$ explicitly as:

$$A_i(t) = a_l^{(i)} t^l + a_{l-1}^{(i)} t^{l-1} + \cdots + a_1^{(i)} t + a_0^{(i)} \tag{5.4.1}$$

$$B_i(t) = b_l^{(i)} t^l + b_{l-1}^{(i)} t^{l-1} + \cdots + b_1^{(i)} t + b_0^{(i)} \tag{5.4.2}$$

$$C_i(t) = c_l^{(i)} t^l + c_{l-1}^{(i)} t^{l-1} + \cdots + c_1^{(i)} t + c_0^{(i)} \tag{5.4.3}$$

Evaluation of any one of these polynomials at the point $t = \tau$ can be expressed as a linear function: $A_i(\tau) = \pi_A^{(i)}(q_H) = \langle K_A^{(i)}, q_H \rangle$, $B_i(\tau) = \pi_B^{(i)}(q_H) = \langle K_B^{(i)}, q_H \rangle$, $C_i(\tau) = \pi_C^{(i)}(q_H) = \langle K_C^{(i)}, q_H \rangle$, where $q_H = (1, \tau, \tau^2, \cdots, \tau^l)$ and

$$K_A^{(i)} = (a_l^{(i)}, a_{l-1}^{(i)}, \cdots, a_1^{(i)}, a_0^{(i)}) \tag{5.4.4}$$

$$K_B^{(i)} = (b_l^{(i)}, b_{l-1}^{(i)}, \cdots, b_1^{(i)}, b_0^{(i)}) \tag{5.4.5}$$

$$K_C^{(i)} = (c_l^{(i)}, c_{l-1}^{(i)}, \cdots, c_1^{(i)}, c_0^{(i)}). \tag{5.4.6}$$

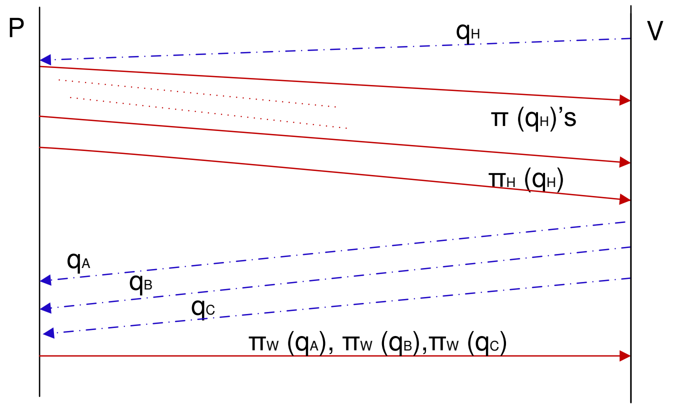


Figure 5.1 PCP querying

We can simply express the PCP queries of Zaatar as

$$q_A = (\pi_A^m(q_H), \pi_A^{m-1}(q_H), \ldots, \pi_A^{n+n'+1}(q_H)) \tag{5.4.7}$$

$$q_B = (\pi_B^m(q_H), \pi_B^{m-1}(q_H), \ldots, \pi_B^{n+n'+1}(q_H)) \tag{5.4.8}$$

$$q_C = (\pi_C^m(q_H), \pi_C^{m-1}(q_H), \ldots, \pi_C^{n+n'+1}(q_H)). \tag{5.4.9}$$

In RIVER, the verifier constructs q_A, q_B, q_C by querying linear functions π_A^i, π_B^i, π_C^i $(i = 0, \cdots, m)$ by a single query q_H.

Similarly, we can express $H(t)$ and $D(t)$ as:

$$H(t) = h_l t^l + h_{l-1}t^{l-1} + \cdots + h_1 t + h_0 \tag{5.4.10}$$

$$D(t) = d_l t^l + d_{l-1}t^{l-1} + \cdots + d_1 t + d_0, \tag{5.4.11}$$

Table 5.1 The First Round of Our QAP-based Linear PCP

For every π in the set of $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots m)$ π_D, perform the following:

- Divisibility queries generation. $\mathcal{V}$ randomly selects $\tau \in_R \mathbb{F}$. $\mathcal{V}$ takes $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^l)$.
- Querying. $\mathcal{V}$ sends out q_H and gets back $\pi(q_H)$.

If all these proofs pass all linearity tests, $\mathcal{V}$ will have: $\pi_D(q_H)$ and

- $\pi_A^{(m)}(q_H), \pi_A^{(m-1)}(q_H), \cdots, \pi_A^{(0)}(q_H)$,
- $\pi_B^{(m)}(q_H), \pi_B^{(m-1)}(q_H), \cdots, \pi_B^{(0)}(q_H)$,
- $\pi_C^{(m)}(q_H), \pi_C^{(m-1)}(q_H), \cdots, \pi_C^{(0)}(q_H)$,

and define: $\pi_H(\cdot) = \langle K_H, \cdot \rangle$, where $K_H = (h_0, h_1, \cdots, h_l)$, and $\pi_D(\cdot) = \langle K_D, \cdot \rangle$, where $K_D = (d_0, d_1, \cdots, d_l)$. Zaatar points out that the evaluation of $H(\tau)$ can be viewed as querying an oracle $\pi_H(\cdot)$ with q_H. Here, we argue that the same holds for the evaluation of $D(\tau)$ – querying the oracle $\pi_D(\cdot)$ with q_H. The idea is detailed in Table 5.1. Note that by comparison, Zaatar requires the queries q_A, q_B, q_C, along with $D(\tau)$ to be entirely computed by $\mathcal{V}$. It should be mentioned that computing these queries by querying another set of proofs requires additional commitments and testing. However, the procedure can be simplified by removing all linearity tests for these $3m + 4 = 3(m + 1) + 1$ proofs. The reason this works is that, according to Theorem 14, our decommitment already provides an inherent linearity test. In the second round of our design, $\mathcal{V}$ issues queries q_H as in Table 5.2. In the third round, $\mathcal{V}$ issues queries q_A, q_B, q_C, q_D as in Table 5.3. After $\mathcal{V}$ collects all responses, he makes the decision as in Table 5.4.

5.4.3 Commit, Decommit and Consistency Verification of RIVER

To ensure the security of the protocol, $\mathcal{P}$ commits to all the linear functions mentioned above. Similarly to Zaatar, our design inherits the single-commit-multiple-decommit protocol from Ginger. For π_H and π_W, $\mathcal{V}$ and $\mathcal{P}$ run the IKO-style single-commit-multi-

Table 5.2 The Second Round of Our QAP-based Linear PCP

$\mathcal{V}$ queries π_H.

- Linearity queries generation. $\mathcal{V}$ selects $q_2, q_3 \in_R \mathbb{F}^l$. Take $q_4 \leftarrow q_3 + q_2$. Perform ρ_{lin} iterations in total.
- QAP queries generation. $\mathcal{V}$ takes $q_H \leftarrow (1, \tau, \tau^2, \cdots, \tau^l)$ and $q_1 \leftarrow (q_H + q_2)$.
- Querying π_H. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{1+3\rho}$ and gets back $\pi_H(q_1), \pi_H(q_2), \cdots, \pi_H(q_{1+3\rho})$.
- Linearity tests. Check whether following holds: $\pi_H(q_4) = \pi_H(q_3) + \pi_H(q_2)$ and likewise for all other $\rho - 1$ iterations. If not, reject.

At the end of this phase, if π_H passes all linearity tests, $\mathcal{V}$ will have: $\pi_H(q_H)$.

decommit protocol to generate the commitment. This part is omitted for simplicity.

For $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D, the case is a bit more complex. We note that in addition to the commitments and decommitments, $\mathcal{V}$ has to also verify the consistency of the polynomials' coefficient vectors corresponding to $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$, and π_D. Namely, $\mathcal{V}$ needs to make sure that $\mathcal{P}$ eventually uses $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$, and π_D to answer $\mathcal{V}$'s queries.

To accomplish this, we use the technique in Section 5.3 and come up with the commitment/decommitment protocol as follows: Before sending $\mathcal{P}$ his computation task, $\mathcal{V}$ secretly generates a random number r and computes by himself the values $A_i(r)$, $B_i(r)$, $C_i(r)$ $(i = 0, 1, \cdots m)$ and $D(r)$, each of which represents, respectively, the commitment for $\pi_A^{(i)}()$, $\pi_B^{(i)}()$, $\pi_C^{(i)}()$, for $i = 0, 1, \ldots, m$, and $\pi_D()$. The algorithm to compute these values is demonstrated in Section 5.5.1. These values are stored for future decommitment. This setup computation is done only once for different values of τ. In comparison with Zaatar, where the setup requires the verifier to evaluate the queries associated with different values of τ, a single r suffices for all τ's in RIVER, since the verifier outsources extra computation to the prover. As in Table 5.5, our commitment design guarantees the consistency of the polynomials' coefficient vectors with the linear functions to which

Table 5.3 The Second Round of Our QAP-based Linear PCP

$\mathcal{V}$ queries π_W. Remember $\pi_W(\cdot) = \langle W, \cdot \rangle$, where $W = (z_m, z_{m-1}, \cdots, z_{N+1})$

- Linearity queries generation. $\mathcal{V}$ select $q_4, q_5 \in_R \mathbb{F}^{m-N}$. Take $q_6 \leftarrow q_4 + q_5$. Perform ρ_{lin} iterations in total.
- QAP queries generation. $\mathcal{V}$ takes:
  - $q_A \leftarrow (\pi_A^{(m)}(q_H), \pi_A^{(m-1)}(q_H), \cdots, \pi_A^{(n+n'+1)}(q_H))$, and $q_1 \leftarrow (q_A + q_4)$.
  - $q_B \leftarrow (\pi_B^{(m)}(q_H), \pi_B^{(m-1)}(q_H), \cdots, \pi_B^{(n+n'+1)}(q_H))$, and $q_2 \leftarrow (q_B + q_4)$.
  - $q_C \leftarrow (\pi_C^{(m)}(q_H), \pi_C^{(m-1)}(q_H), \cdots, \pi_C^{(n+n'+1)}(q_H))$, and $q_3 \leftarrow (q_C + q_4)$.
- Querying π_W. $\mathcal{V}$ sends out $q_1, q_2, \cdots, q_{3+3\rho}$ and gets back $\pi_W(q_1), \pi_W(q_2), \cdots, \pi_W(q_{3+3\rho})$.
- Linearity tests. Check whether following holds: $\pi_W(q_6) = \pi(q_4) + \pi(q_5)$ and likewise for all other $\rho - 1$ iterations. If not, reject. Otherwise, accept and output $\pi_W(q_A) \leftarrow \pi_W(q_1) - \pi_W(q_4)$, $\pi_W(q_B) \leftarrow \pi_W(q_2) - \pi_W(q_4)$, $\pi_W(q_C) \leftarrow \pi_W(q_3) - \pi_W(q_4)$.

$\mathcal{P}$ commits.

Theorem 16. *Let π be any of the linear functions $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$ and π_D. By performing our protocol, the commitment to π is guaranteed to be bound to a linear function $\tilde{\pi}$, and the probability that $\pi \neq \tilde{\pi}$ is at most $1/|\mathbb{F}|$. The probability is over all the randomness of the prover.*

Proof. Given that our protocol performs the single-commit-multi-decommit protocol when querying π, the response to the query is guaranteed to be bound to a linear function $\tilde{\pi}$. This feature is provided by the underlying single-commit-multi-decommit protocol. If $\pi \neq \tilde{\pi}$ but $\tilde{\pi}$ still passes the decommitment, $\tilde{\pi}(r) = \pi(r)$ must hold. For all possible choices of $\tilde{\pi}$, only $1/|\mathbb{F}|$ of them can satisfy this equation. However, r is unknown by the prover. Thus, the probability that a dishonest prover chooses a $\tilde{\pi} \neq \pi$ that passes the decommitment is at most $1/|\mathbb{F}|$. $\square$

Table 5.4 The Decision Making Stage of Our QAP-based Linear PCP

Decision Making: (Note: $(z_1, z_2, \cdot, z_{n+n'}) = X||Y$.)

- $\mathcal{V}$ computes:
  - $p_A \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_A^{(i)}(q_H) + \pi_A^{(0)}(q_H)$
  - $p_B \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_B^{(i)}(q_H) + \pi_B^{(0)}(q_H)$
  - $p_C \leftarrow \sum_{i=1}^{(n+n')} z_i \cdot \pi_C^{(i)}(q_H) + \pi_C^{(0)}(q_H)$
- Divisibility Test. $\mathcal{V}$ checks whether the following holds: $\pi_D(q_H) \cdot \pi_H(q_H) = (\pi_Z(q_A) + p_A) \cdot (\pi_Z(q_B) + p_B) - (\pi_Z(q_C) + p_C)$.

Table 5.5 Decommit Design for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D

$\mathcal{P}$'s Input: linear functions π_D, $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, for $i = 1, \ldots, m$.

$\mathcal{V}$'s Input: $A_i(r)$, $B_i(r)$, $C_i(r)$, $i = 0, \cdots, m$ and $D(r)$, $t = (1, r, r^2, \cdots, r^l)$. $q_1, \cdots, q_\mu$

Commitment

The verifier generates the commitment information as in Section 5.5.1.

Decommitment

Step 1: $\mathcal{V}$ picks μ secrets $\alpha_1, \cdots, \alpha_\mu \in \mathbb{F}$

$\mathcal{V}$ queries $\mathcal{P}$ with $q_1, \cdots, q_\mu$ and $T = t + \alpha_1 q_1 + \cdots + \alpha_\mu q_\mu$.

Step 2: $\mathcal{P}$ returns $(\pi_A^{(i)}(q_1), \cdots, \pi_A^{(i)}(q_\mu), \pi_A^{(i)}(T))$, $(\pi_B^{(i)}(q_1), \cdots, \pi_B^{(i)}(q_\mu), \pi_B^{(i)}(T))$, $(\pi_C^{(i)}(q_1), \cdots, \pi_C^{(i)}(q_\mu), \pi_C^{(i)}(T))$, where $i = 0, \cdots, m$ and $(\pi_D(q_1), \cdots, \pi_D(q_\mu), \pi_D(T))$.

Step 3: $\mathcal{V}$ checks whether $\pi_A^{(i)}(T) = A_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_A^{(i)}(q_j)$ and whether $\pi_B^{(i)}(T) = B_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_B^{(i)}(q_j)$ and whether $\pi_C^{(i)}(T) = C_i(r) + \sum_{j=1}^{\mu} \alpha_j \pi_C^{(i)}(q_j)$, $i = 0, \cdots, m$ and $\pi_D(T) = D(r) + \sum_{j=1}^{\mu} \alpha_j \pi_D(q_j)$ hold.

If so, $\mathcal{V}$ accepts. Otherwise, he rejects and output $\perp$.

5.5 Performance Analysis

For the informational cost, it is straightforward to see that, once committed, all the queries in the verification are not depending on the circuit description. Namely, during the query generating of the verification stage, the verifier does not need to access the circuit information any more. Our design separates the verification workload that involves only non-sensitive information from the verification workload that involves sensitive information (e.g. the circuit information). In the scenarios with a third-party verifier, the verifier can undertake the workload involving only non-sensitive information (e.g. query generating) without knowing the secrecy of the computation task.

In the following, we derive the computational cost of our RIVER design and compare it with previous work. In the process, we show that, similarly to Ginger and Zaatar, our protocol batches many instances for one *same* circuit to reduce the cost per instance. But RIVER can amortize more parts of amortized cost over *all different circuits of the same size.*

5.5.1 The Verifier

This section performs an analysis of the verifier's cost. A comparison with the verifier's costs in two other the state-of-the-art designs is given in Table 5.6.

Setup

. The cost that RIVER incurs upon the commitment is $(|W_R|+|C_R|)\cdot(e+c)/\beta$. This is because RIVER needs two commitment query constructions. One is for π_H, and incurs a cost of $|C_R|\cdot(e+c)/\beta$, while the other is for π_W, and incurs a cost of $|W_R|\cdot(e+c)/\beta$. This total cost is the same as that of Zaatar.

It is apparent that RIVER introduces additional workload to the setup stage. $\mathcal{V}$ has to evaluate $A_i(r)$, $B_i(r)$, $C_i(r)$ and $D(r)$. However, we have discovered that a large part of the computation cost is independent of the underlying circuits. Rather, the computation only depends on the size of the circuit. This implies that this part of the computation can be amortized over many different circuits, which only share the same size, rather than over many different instances of the same circuit. To see this, first notice that the target polynomial $D(t)=\prod_{k=1}^{|C_R|}(t-\sigma_k)$ does not depend on the circuit details, but rather $D(t)$ is determined by the circuit size. Hence, we can compute $D(r)$ once for all circuits of the same size, where r is the secret as in Section 5.4. If given in the form of generalized Newton's interpolation formula ([61], 4.6.4), $D(r)$ can be evaluated in time $|C_R|\cdot f$.

Second, we express $A_i(t)$, $B_i(t)$, $C_i(t)$ in the form of Lagrange Polynomial interpolation: $A_i(t) = \sum_{j=1}^{|C_R|} a_{ij} \cdot l_j(t)$, $B_i(t) = \sum_{j=1}^{|C_R|} b_{ij} \cdot l_j(t)$, $C_i(t) = \sum_{j=1}^{|C_R|} c_{ij} \cdot l_j(t)$, where $l_j(t) = \prod_{1 \le k \le |C_R|, k \ne j} \frac{(t-\sigma_k)}{(\sigma_j - \sigma_k)}$ are Lagrange basis polynomials. We can represent these Lagrange basis polynomials as follows: $l_j(t) = \frac{D(t)}{(t-\sigma_j)\cdot\frac{1}{v_j}}$, where $v_j = 1/\prod_{0 \le k \le |C|, k \ne j}(\sigma_j - \sigma_k)$. If we choose these σ_k $(k = 1, \cdots, |C_R|)$ to follow an arithmetic progression [10], $l_j(r)$ $(j = 1, \cdots, |C_R|)$ can be evaluated in total time of $(f_{div} + 4f)|C_R|$. (Computing $1/v_{j+1}$ from $1/v_j$ requires only two operations and computing $1/v_0$ uses $|C_R|$ multiplication. Recall that $D(r)$ is computed already. Finally, to get each $l_j(r)$, a multiplication and one division are needed.) Given that both the evaluation of $D(r)$ and $l_j(r)$ are independent of the underlying circuit, we can amortize the cost of the evaluation into all circuits of the same size.

The remaining work is to evaluate $A_i(r)$, $B_i(r)$, $C_i(r)$ from the Lagrange polynomials $l_j(r)$ $(j = 1, \cdots, |C_R|)$. But this is reduced to merely several additions of $l_j(r)$ polynomials – note that the coefficients a_{ij}, b_{ij}, c_{ij} are all either 0 or 1. The number of wires in the circuit that can contribute to the multiplication gates is at most $2|C_R|$. The total number of additions to evaluate $A_i(r)$ and $B_i(r)$ is at most the number of wires in the circuit that can contribute to the multiplication gates. Then, the total number of additions to evaluate $A_i(r)$ and $B_i(r)$ is at most $2|C_R|$. The total number of additions to evaluate $C_i(r)$ is $(|W_R| + |Y|)$, since it takes $(|W_R| + |C_R|) \cdot (e + c)$ to generate the commitment queries (where, the whole cost of setup is at most $(|W_R|+|C_R|)\cdot(e+c)/\beta + (f_{div} + 5f) \cdot |C_R|/\beta + (2|C_R| + |W_R| + |Y|) \cdot g/\beta)$, where g is the cost of addition over a finite field. Since g is small, we omit addition cost in the tables of cost, as Zaatar [10] does.

Compared to Zaatar, RIVER introduces an extra cost of $(f_{div}+5f)\cdot|C_R|/\beta+(2|C_R|+|W_R| + |Y|) \cdot g/\beta$ to the total cost of setup. However, notice that this part of the computation can be amortized over many different circuits, which only share the same size, rather than over many different instances of the same circuit. Thus, RIVER actually introduces a negligible cost in the setup phase.

Linearity Query Generation

The cost of generating the linearity queries for π_H is $\rho \cdot \rho_{lin} \cdot 2 \cdot |C_R| \cdot c/(\beta \cdot \gamma)$. Another group of linearity queries are for the proof π_W. The cost of generating these linearity queries is $\rho \cdot \rho_{lin} \cdot 2 \cdot |W_R| \cdot c/(\beta \cdot \gamma)$. Thus, the total cost of generating linearity queries amounts to $\rho \cdot \rho_{lin} \cdot 2 \cdot (|C_R| + |W_R|) \cdot c/(\beta \cdot \gamma)$.

Divisibility Query Generation, Decommitment Query Generation and Decommitment Test

These are straight-forward, we omit these for simplicity.

Non-amortized Costs

From the construction above, we draw the following observations:

- For $i = 1, \cdots, n$, we have $C_i(t) = 0$ for any $t \in \mathbb{F}$, since the inputs of the circuit cannot be outputs of multiplication gates.
- For $i = n+1, \cdots, n+n'$, we have $A_i(t) = 0$ for any $t \in \mathbb{F}$, since the outputs of the circuit Ψ' cannot be inputs to multiplication gates.
- For $i = n+1, \cdots, n+n'$, we have $B_i(t) = 0$ for any $t \in \mathbb{F}$, since the outputs of the circuit Ψ' cannot be inputs to multiplication gates.

Thus, the verifier's cost in the decision making stage (computing p_A, p_B, p_C) is merely $\rho \cdot (2|X| + |Y|) \cdot f$.

Comparison with Zaatar

We list the amortized and non-amortized cost of both RIVER and Zaatar in Table 5.6. At this time, it is useful to take $W_R = W_Z$ and $C_R = C_Z$.

We can see that, both the amortized and non-amortized cost of RIVER are less than Zaatar. For the amortized part, which is known as the *investment*, even for cases when $\beta = 1$ and $\gamma = 1$, the cost of RIVER is less than Zaatar. (To have a clear picture, we look at a real example: computing xA where the input x is a $1 \times M$ vector and A is a fixed $M \times M$ matrix. This is widely used in all kinds of scientific computing such as communications, signal processing, and control systems, and is a basic operation of many computations. We use previously published models ([10]) and instantiate the costs as in Table 5.6. From the instance, for $M > 5000$, We see the amortized cost in RIVER is at least 28% less than that in Zaatar. For $M < 5000$, the improvement is even greater.) Since the same part of the amortized cost in RIVER and Zaatar is dominated by linearity test queries, if we apply the query compressing technique in Ginger ([37]), RIVER will have a more significant improvement compared to Zaatar.

5.5.2 The Prover

The method to construct the proof vector is the same as that in Zaatar. The cost is $T + 3f \cdot |C_R| \cdot log^2|C_R|$. We omit the details here. The remaining cost is from the fact that the prover needs to compute the coefficients of $A_i(t)$, $B_i(t)$, and $C_i(t)$, $(i = 0, 1, \cdots, m)$. However, this could be amortized. First, remember that each of $A_i(t)$, $B_i(t)$ and $C_i(t)$, $(i = 0, 1, \cdots, m)$ are sums of several Lagrange basis polynomials. The cost to get the coefficients of the Lagrange basis polynomials is independent of the underlying circuit and can be amortized into all circuits of the same size and is negligible. Second, similarly to Section 5.2, the number of additions of Lagrange basis polynomials is at most $2|\Psi| + |Y|$. Each Lagrange basis polynomials has degree at most $|C_R|$. Thus, for each instance, the cost of computing the coefficients is at most $(2|\Psi| + |Y|) \cdot |C_R| \cdot g/\beta$, where g is the cost of addition over the field $\mathbb{F}$. As in Zaatar [10], we omit the addition cost.

When the prover issues the PCP responses, he needs to respond to not only queries for π_W and π_H, but also queries for $\pi_A^{(i)}$, $\pi_B^{(i)}$, $\pi_C^{(i)}$, $(i = 0, 1, \cdots, m)$ and π_D. The cost for the former is $(h+1)\cdot(|W_R|+|C_R|)+\rho\cdot(3|W_R|+|C_R|)\cdot f+\rho_{lin}\cdot 3\cdot(|W_R|+|C_R|)\cdot\rho\cdot f$. Given that the length of the latter is $|C_R|$ and these responses do not depend on underlying circuit or the proof vector π_W, the cost to compute the responses for the latter can be amortized into all instances of the same circuit size. This cost is $[h+\rho\cdot(3m+4)\cdot f]\cdot|C_R|/(\beta\cdot\gamma)$.

The comparison in terms of the prover's cost is in Table 5.7. We also use the computation example in Section 5.5.1 to demonstrate the improvement. For any $M > 100$, RIVER's non-amortized cost of the prover is almost the same as that of Zaatar. We demonstrate the results using $M = 10000$. Although RIVER introduces amortized cost, it becomes negligible since it can be amortized into all instances of the same circuit size.

5.6 Conclusions

The state-of-the-art designs such as Pepper/Ginger/Zaatar combine a commitment protocol to a linear PCP, achieving breakthroughs in verifiable computation. However, the high computation, communication and storage costs still keep general verifiable computation away from practicality. In this chapter, we presented a new verifiable-computation protocol called RIVER. We show that RIVER reduces the amortized computational costs of the verifier. Namely, the number of batched instances of the protocol, required for amortization, will largely decrease. RIVER introduces a negligible increase in the prover's costs. Specifically, the increased amortized cost can be amortized over instances of different circuits of the same size. Thus, this part can be done only once, but used for all possible verifications.

In addition, for the first time in the context of verifiable computation, we address the problem of reducing the informational costs. RIVER removes the requirement that the verifier has to access the circuit description during query generating. Furthermore, this feature of RIVER can be viewed as a first step towards applying QAP-based arguments to the secure outsourcing of verification.

Table 5.6 Comparison of Cost for Verifier in Each Instance

	Ginger	Zaatar	RIVER
Setup: Commit	$\lvert W_G\rvert \cdot e/(\beta \cdot \gamma)$	$(\lvert W_Z\rvert + \lvert C_Z\rvert) \cdot e/(\beta \cdot \gamma)$	$(\lvert W_R\rvert + \lvert C_R\rvert) \cdot e/(\beta \cdot \gamma) + (f_{div} + 5f) \cdot \lvert C_R\rvert/(\beta \cdot \gamma)$
Linearity Query Generation	$\rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert C_G\rvert + \lvert C_G\rvert^2) \cdot c/(\beta \cdot \gamma)$	$\rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert W_Z\rvert + \lvert C_Z\rvert) \cdot c/(\beta \cdot \gamma)$	$\rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert W_R\rvert + \lvert C_R\rvert) \cdot c/(\beta \cdot \gamma)$
Other PCP Query Generation	$\rho \cdot (c \cdot \lvert C_G\rvert + f \cdot K)/\beta$	$\rho \cdot [c + (f_{div} + 5f) \cdot \lvert C_Z\rvert + f \cdot K + 3f \cdot K_2]/\beta$	$\rho \cdot \lvert C_R\rvert \cdot f/(\beta \cdot \gamma)$
Decommitment Query Generation	$\rho \cdot L \cdot f/\beta$	$\rho \cdot (\rho_{lin} \cdot 3 \cdot (\lvert W_Z\rvert + \lvert C_Z\rvert) + (3\lvert W_Z\rvert + \lvert C_Z\rvert)) \cdot f/\beta$	$\rho \cdot (\rho_{lin} \cdot 3 \cdot (\lvert W_R\rvert + \lvert C_R\rvert) + (3\lvert W_R\rvert + \lvert C_R\rvert)) \cdot f/\beta$
Decommitment Test	$d + \rho \cdot L \cdot f$	$d + \rho \cdot (\rho_{lin} \cdot 6 + 4) \cdot f$	$2d + \rho \cdot (\rho_{lin} \cdot 6 + 4) \cdot f + \rho \cdot (3m + 4) \cdot f/\beta$
Decision Making	$\rho \cdot (\lvert X\rvert + \lvert Y\rvert) \cdot f$	$\rho \cdot (3\lvert X\rvert + 3\lvert Y\rvert) \cdot f$	$\rho \cdot (2\lvert X\rvert + \lvert Y\rvert) \cdot f$
Total non-amortized cost	$d + \rho \cdot (L + \lvert X\rvert + \lvert Y\rvert) \cdot f$	$2d + \rho \cdot f \cdot (3\lvert X\rvert + 3\lvert Y\rvert + \rho_{lin} \cdot 6 + 4)$	$2d + \rho \cdot f \cdot (2\lvert X\rvert + \lvert Y\rvert + \rho_{lin} \cdot 6 + 4)$
Total amortized cost	$\lvert W_G\rvert \cdot e/(\beta \cdot \gamma) + \rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert C_G\rvert + \lvert C_G\rvert^2) \cdot c/(\beta \cdot \gamma) + \rho \cdot c \cdot \lvert C_G\rvert/\beta + \rho \cdot (L + K) \cdot f/\beta$	$(\lvert W_Z\rvert + \lvert C_Z\rvert) \cdot e/(\beta \cdot \gamma) + \rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert W_Z\rvert + \lvert C_Z\rvert) \cdot c/(\beta \cdot \gamma) + \rho \cdot [c + (f_{div}) \cdot \lvert C_Z\rvert]/\beta + (\rho_{lin} \cdot 3 \cdot (\lvert W_Z\rvert + \lvert C_Z\rvert) + (3\lvert W_Z\rvert + 6\lvert C_Z\rvert + K + 3K_2)) \cdot \rho \cdot f/\beta$	$(\lvert W_R\rvert + \lvert C_R\rvert) \cdot e/(\beta \cdot \gamma) + \rho \cdot \rho_{lin} \cdot 2 \cdot (\lvert W_R\rvert + \lvert C_R\rvert) \cdot c/(\beta \cdot \gamma) + (\rho_{lin} \cdot 3 \cdot (\lvert W_R\rvert + \lvert C_R\rvert) + (3\lvert W_R\rvert + \lvert C_R\rvert) + 3m + 4) \cdot \rho \cdot f/\beta + ((f_{div} + 5f) \cdot \lvert C_R\rvert + \rho \cdot \lvert C_R\rvert \cdot f)/(\beta \cdot \gamma)$

C_G: set of constraints in Ginger	$\lvert W_G\rvert$: number of variables in the constraints (excluding inputs and outputs) in Ginger
C_Z: set of constraints in Zaatar	$\lvert W_Z\rvert$: number of variables in the constraints (excluding inputs and outputs) in Zaatar
C_R: set of constraints in our design	$\lvert W_R\rvert$: number of variables in the constraints (excluding inputs and outputs) in our design
$\lvert X\rvert$: number of input	$\lvert Y\rvert$: number of output
g: cost of addition over $\mathbb{F}$	L: number of PCP queries in Ginger
β: number of batching	γ: number of circuits of the same size.
ρ: number of iteration of verification for one instance	ρ_{lin}: number of iterations of linearity tests in one iteration of verification.
f_{div}: cost of division over $\mathbb{F}$	f: cost of multiplication over $\mathbb{F}$
c: cost of pseudorandomly generating an element in $\mathbb{F}$	
d: cost of decryption over $\mathbb{F}$	e: cost of encryption over $\mathbb{F}$
K: number of additive terms in the constraints of Ginger	K_2: number of distinct additive degree-2 terms in the constraints of Ginger

Table 5.7 Comparison of Cost for Prover in Each Instance

	Ginger	Zaatar	RIVER
Construct proof	$T + f \cdot \lvert W_G \rvert^2$	$T+3f \cdot \lvert C_Z \rvert \cdot log^2 \lvert C_Z \rvert$	$T + 3f \cdot \lvert C_R \rvert \cdot log^2 \lvert C_R \rvert$
	$32s + 3.2 \times 10^9 s$	$32s + 3.2 \times 10^4 s$	$32s + 3.2 \times 10^4 s$
Issue PCP responses	$(h + (\rho \cdot L + 1) \cdot f) \cdot (\lvert C_G \rvert + \lvert C_G \rvert^2)$	$(h + (\rho \cdot L' + 1) \cdot f) \cdot (\lvert C_Z \rvert + \lvert W_Z \rvert)$	$(h+1) \cdot (\lvert W_R \rvert + \lvert C_R \rvert) + \rho \cdot (3 \lvert W_R \rvert + \lvert C_R \rvert) \cdot f + \rho_{lin} \cdot 3 \cdot (\lvert W_R \rvert + \lvert C_R \rvert) \cdot \rho \cdot f + [h + \rho \cdot (3m+4) \cdot f] \cdot \lvert C_R \rvert / (\beta \cdot \gamma)$
	$2.9 \times 10^{12} s$	$5.78 \times 10^4 s$	$5.79 \times 10^4 s + \frac{7.7 \times 10^{10}}{(\beta \cdot \gamma)} s$

T: cost of computing the task h: cost of ciphertext add plus multiply

$L = 3\rho_{lin} + 2$: number of (high order) PCP queries in Ginger

$L' = 6\rho_{lin} + 4$: number of PCP queries in Zaatar

CHAPTER 6. BLOCK PROGRAMS: IMPROVING EFFICIENCY OF VERIFIABLE COMPUTATION FOR CIRCUITS WITH REPEATED SUBSTRUCTURES

6.1 Problem Statement

If viewing cloud computing from the high-level perspective, two parties are involved: the client $\mathcal{V}$, who is computationally weak, has computation tasks to be delegated to the cloud; the cloud server $\mathcal{P}$, who is computationally powerful, provides computing services to the client. After receiving the computation task, $\mathcal{P}$ performs the computation and returns the results to $\mathcal{V}$. Later, $\mathcal{V}$ runs the verification protocol with $\mathcal{P}$ to ensure the correctness of the returned result. The computation task is a piece of code written in a high-level programming language. In the verification stage, this piece of code is transformed into an arithmetic circuit form. Existing compilers can turn high-level programs into (non-deterministic) circuits [9, 10, 14]. Since this phase is outside the scope of the current chapter, we directly assume the computation task is formalized into an arithmetic circuit in the verification stage. We assume this piece of code is a loop program and in the verification stage, this program is formalized into a *loop circuit* (we will further illustrate loop circuits in Section 6.3).

6.2 Our Theoretical Results: Block Programs (BPs)

Just as QSPs are a natural extension of span programs (SPs) [62], our new form of arithmetization which we call Block Programs (BPs), is a natural extension of Quadratic Arithmetic Programs (QAPs). We focus on a kind of general circuits, which may be an arithmetic circuit or a boolean circuit, that is composed of *identical* units. This elementary unit, which we call *block*, can be a simple NAND gate. But it is more common to be a more complex *sub-circuit*. This kind of sub-circuit can be multiple-input-multiple-output. The sub-circuits can be viewed as pseudo-gates, a generalized form of gates. Consider a scenario where the whole circuit is composed of this kind of sub-circuits just like in Figure 6.1, where all blocks are identical. We define BPs as follows.

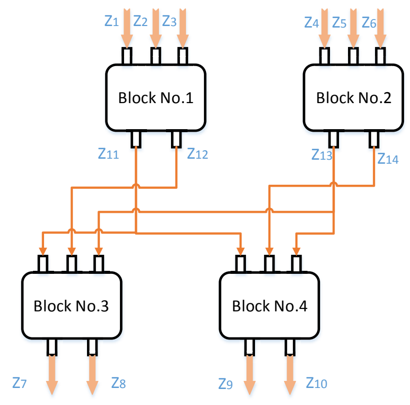
Definition 2. (Block Programs)(BPs, Generalization of Quadratic Arithmetic Programs (QAPs) [13])

We assume that a circuit computing a function $\Psi : \mathbb{F}^N \mapsto \mathbb{F}^{N'}$ *is composed of identical blocks, all of which compute the same function:* $\psi : \mathbb{F}^V \mapsto \mathbb{F}^M$*. (*$\psi$ *implies* M *functions* $\psi_j : \mathbb{F}^V \mapsto \mathbb{F}$ *where* $j = 1, 2, \cdots, M$*. Each of these* M *functions is associated to one of the outputs of* ψ*.)*

A BP $\mathcal{Q}$ *over field* $\mathbb{F}$ *contains* V *sets of* W *polynomials:* $\{A_{1w}(t)\}$*,* $\{A_{2w}(t)\}$*,* $\cdots$*,* $\{A_{Vw}(t)\}$*, and* M *sets of* W *polynomials :* $\{B_{1w}(t)\}$*,* $\{B_{2w}(t)\}$*,* $\cdots$*,* $\{B_{Mw}(t)\}$*, for* $w \in \{1, \cdots, W\}$*, and a target polynomial* $D(t)$*. We say* $\mathcal{Q}$ *computes* Ψ *using block* ψ *if the following holds:*

$(z_1, z_2, \cdots, z_{N+N'}) \in \mathbb{F}^{N+N'}$ *is a valid assignment of* Ψ*'s inputs and outputs, if and only if there exist coefficients* $z_{N+N'+1}, \cdots, z_W$ *such that* $D(t)$ *divides* $P_j(t)$*,* $(j = 1, \cdots, M)$*, where*

$$P_j(t) = \psi_j\left(\sum_{w=1}^{W} z_w \cdot A_{1w}(t), \cdots, \sum_{w=1}^{W} z_w \cdot A_{Vw}(t)\right) - [\sum_{w=1}^{W} z_w \cdot B_{jw}(t)]. \qquad (6.2.1)$$



	Meaning	Range
v	Index of Input of a Block	$1, \cdots, V$
j	Index of Output of a Block	$1, \cdots, M$
k	Index of Blocks	$1, \cdots, K$
w	Index of wires connecting blocks in the circuit	$1, \cdots, W$
w_1	w is represented as (w_1, w_2)	$1, \cdots, M+Q$
w_2	w is represented as (w_1, w_2)	$1, \cdots, K$
N	the number of inputs of the circuit	
N'	the number of outputs of the circuit	
i	Index of Direct Input of a Block	$1, \cdots, M$
u	Index of Extra Input of a Block	$1, \cdots, Q$

Figure 6.1 Computing through Blocks

Figure 6.2 Notations

In other words, for each j, there exists a polynomial $H_j(t)$ such that $D(t) \cdot H_j(t) = P_j(t)$, where $j = 1, 2, \cdots, M$.

After we define Block Programs, two natural questions are: (1) do such Block Programs exist? (2) if yes, how to construct Block Programs? Here, we will show the existence of Block Programs by constructing corresponding BPs by polynomial interpolation. In particular, given a circuit that computes Ψ and is composed of identical blocks denoted by $\psi : \mathbb{F}^V \mapsto \mathbb{F}^M$ as above, we firstly construct a group of interpolation polynomials in Lagrange form. Then, we prove that the constructed polynomials form the BP that computes Ψ.

Suppose the circuit is composed of K blocks. We pick an arbitrary value σ_k for each block. When we pick these K values from $\mathbb{F}$, we make sure all these values are different from each other. We define the target polynomial as follows: $D(t) = \prod_{k=1}^{K}(t - \sigma_k)$. Now we consider the set of all the inputs of the circuit, and all the outputs of each block. Firstly, we label each input of the whole circuit and each output from a block with an index w, where $w \in \{1, 2, \cdots, W\}$ and W is the total number of all *wires*, namely, all the inputs of the whole circuit and all the outputs of each block. We can easily deduce that $W = N + K \cdot M$. (Recall that N is the number of inputs to the whole circuit, K is the number of blocks, and M is the number of outputs from a block) Then the values of each input of the whole circuit and each output from a block can be denoted by z_w. Secondly, we assign $V + M$ interpolation polynomials in Lagrange form to each wire, indicating whether the corresponding value z_w is the v-th input, or the j-th output of each block, where $v = 1, 2, \cdots, V$ and $j = 1, 2, \cdots, M$. These polynomials indeed determine how these blocks are interconnected. Thus, the resulting set of polynomials is a complete description of the original circuit. Specifically, we let the polynomials $\{A_{vw}(t)\}$ (for $w = 1, 2, \cdots, W$) encode the v-th input into each block, where $v = 1, 2, \cdots, V$ and let $\{B_{jw}(t)\}$ (for $w = 1, 2, \cdots, W$) encode the j-th output from each block, where $j = 1, 2, \cdots, M$. In particular, we let

$$A_{vw}(\sigma_k) = \begin{cases} 1 & \text{if } z_w \text{ is the } v\text{-th input to the } k\text{-th block;} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2.2}$$

$$B_{jw}(\sigma_k) = \begin{cases} 1 & \text{if } z_w \text{ is the } j\text{-th output from the } k\text{-th block;} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2.3}$$

Based on the evaluations at the K values $\sigma_1, \cdots, \sigma_K$, it is straightforward to construct $\{A_{vw}(t)\}$ and $\{B_{jw}(t)\}$, for $w = 1, \cdots, W$, $v = 1, \cdots, V$, and $j = 1, \cdots, M$ using interpolation polynomials in Lagrange form. Let $\mathcal{K}$ be the set of indices: $\mathcal{K} = \{1, 2, \cdots, K\}$. Then,

$$A_{vw}(t) = \sum_{k \in \mathcal{K}} A_{vw}(\sigma_k) \cdot \frac{\prod_{k^* \in \mathcal{K}, k^* \neq k} (t - \sigma_{k^*})}{\prod_{k^* \in \mathcal{K}, k^* \neq k} (\sigma_k - \sigma_{k^*})}, \tag{6.2.4}$$

and

$$B_{jw}(t) = \sum_{k \in \mathcal{K}} B_{jw}(\sigma_k) \cdot \frac{\prod_{k^* \in \mathcal{K}, k^* \neq k} (t - \sigma_{k^*})}{\prod_{k^* \in \mathcal{K}, k^* \neq k} (\sigma_k - \sigma_{k^*})}. \tag{6.2.5}$$

We can show that $\{A_{vw}(t)\}$ and $\{B_{jw}(t)\}$, which we have constructed using interpolation polynomials in Lagrange form, constitute a BP that computes Ψ, where $w = 1, 2, \cdots, W$, $v = 1, 2, \cdots, V$, and $j = 1, 2, \cdots, M$.

Theorem 17. *For a circuit which computes a function $\Psi : \mathbb{F}^N \mapsto \mathbb{F}^{N'}$, if the circuit is composed of identical blocks denoted by $\psi : \mathbb{F}^V \mapsto \mathbb{F}^M$, then $\{A_{vw}(t)\}$ and $\{B_{jw}(t)\}$ constructed above constitute a BP that compute Ψ, where $w = 1, 2, \cdots, W$, $v = 1, 2, \cdots, V$, and $j = 1, 2, \cdots, M$.*

Proof. The rough idea of the proof is as follows. If we evaluate $\{A_{vw}(t)\}$ and $\{B_{jw}(t)\}$ in σ_k, for $w = 1, 2, \cdots, W$, $v = 1, 2, \cdots, V$, and $j = 1, 2, \cdots, M$, which we have constructed, we can observe that this makes the equation (6.2.1) become:

$$P_j(\sigma_k) = \psi_j(\text{the kth block's inputs}) - (\text{the kth block's output}). \tag{6.2.6}$$

This means that if the output is correct, then $P_j(\sigma_k) = 0$. Namely, σ_k is a root of $P_j(t)$, which in turn means that $(t - \sigma_k)$ divides $P_j(t)$, which sets up the divisibility by $D(t)$. The rigorous proof is in Section 6.5.1. □

Here, we give a concrete example to demonstrate how we build the BPs for the circuit in Figure 6.1.

Let us take the circuit in Figure 6.1 as a concrete example to demonstrate how we build the corresponding BPs. The corresponding BP is demonstrated in Table 6.1. The polynomials are determined by the evaluations at the four values $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ (this tuple is denoted by Σ in Table 6.1), which are associated with the four blocks. First, given that there are four blocks, we select four distinct values: $\sigma_1, \sigma_2, \sigma_3, \sigma_4 \in \mathbb{F}$, each associated with the corresponding block. The number of blocks also implies the degree of the BP: four. Given that each block has three inputs and two outputs, the BP has five sets of polynomials. We take the circuit's input/output wires and all the blocks' output wires into account. Given the number of these wires is fourteen, each set has fourteen polynomials. Second, we construct these polynomials based on each wires contributions to the blocks, according to the interconnection of the blocks in the circuit schematic. More specifically, for $v = 1, 2, 3$, $w = 1, 2, \cdots, 14$, and $k = 1, 2, 3, 4$, we let $A_{vw}(\sigma_k) = 1$ if z_w is the v-th input to the k-th block, and $A_{vw}(\sigma_k) = 0$ otherwise; we also let $B_{jw}(\sigma_k) = 1$ if z_w is the j-th output from the k-th block, and $B_{jw}(\sigma_k) = 0$ otherwise. For instance, since z_{11} is the first input to both Block No.3 and Block No.4, $A_{1,11}(\sigma_3) = 1$ and $A_{1,11}(\sigma_4) = 1$. since z_{13} is the first output from Block No.2, $B_{1,13}(\sigma_2) = 1$. Hence, we can simply generate a table of the polynomials' evaluations at $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ as in Table 6.1. Thirdly, we construct the polynomials using the interpolation polynomials in the Lagrange form. For example, the evaluation of $A_{1,1}(t)$ at the four values $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ are $(1, 0, 0, 0)$. Then, we have $A_{1,1}(t) = \frac{(t-\sigma_2)(t-\sigma_3)(t-\sigma_4)}{(\sigma_1-\sigma_2)(\sigma_1-\sigma_3)(\sigma_1-\sigma_4)}$. In essence, the polynomials in the BP are defined in terms of their evaluations at the values which we pre-select for each block.

Table 6.1 BP of the circuit

	Σ		Σ		Σ		Σ		Σ
$A_{1,1}(t)$	$(1,0,0,0)$	$A_{2,1}(t)$	$(0,0,0,0)$	$A_{3,1}(t)$	$(0,0,0,0)$	$B_{1,1}(t)$	$(0,0,0,0)$	$B_{2,1}(t)$	$(0,0,0,0)$
$A_{1,2}(t)$	$(0,0,0,0)$	$A_{2,2}(t)$	$(1,0,0,0)$	$A_{3,2}(t)$	$(0,0,0,0)$	$B_{1,2}(t)$	$(0,0,0,0)$	$B_{2,2}(t)$	$(0,0,0,0)$
$A_{1,3}(t)$	$(0,0,0,0)$	$A_{2,3}(t)$	$(0,0,0,0)$	$A_{3,3}(t)$	$(1,0,0,0)$	$B_{1,3}(t)$	$(0,0,0,0)$	$B_{2,3}(t)$	$(0,0,0,0)$
$A_{1,4}(t)$	$(0,1,0,0)$	$A_{2,4}(t)$	$(0,0,0,0)$	$A_{3,4}(t)$	$(0,0,0,0)$	$B_{1,4}(t)$	$(0,0,0,0)$	$B_{2,4}(t)$	$(0,0,0,0)$
$A_{1,5}(t)$	$(0,0,0,0)$	$A_{2,5}(t)$	$(0,1,0,0)$	$A_{3,5}(t)$	$(0,0,0,0)$	$B_{1,5}(t)$	$(0,0,0,0)$	$B_{2,5}(t)$	$(0,0,0,0)$
$A_{1,6}(t)$	$(0,0,0,0)$	$A_{2,6}(t)$	$(0,0,0,0)$	$A_{3,6}(t)$	$(0,1,0,0)$	$B_{1,6}(t)$	$(0,0,0,0)$	$B_{2,6}(t)$	$(0,0,0,0)$
$A_{1,7}(t)$	$(0,0,0,0)$	$A_{2,7}(t)$	$(0,0,0,0)$	$A_{3,7}(t)$	$(0,0,0,0)$	$B_{1,7}(t)$	$(0,0,1,0)$	$B_{2,7}(t)$	$(0,0,0,0)$
$A_{1,8}(t)$	$(0,0,0,0)$	$A_{2,8}(t)$	$(0,0,0,0)$	$A_{3,8}(t)$	$(0,0,0,0)$	$B_{1,8}(t)$	$(0,0,0,0)$	$B_{2,8}(t)$	$(0,0,1,0)$
$A_{1,9}(t)$	$(0,0,0,0)$	$A_{2,9}(t)$	$(0,0,0,0)$	$A_{3,9}(t)$	$(0,0,0,0)$	$B_{1,9}(t)$	$(0,0,0,1)$	$B_{2,9}(t)$	$(0,0,0,0)$
$A_{1,10}(t)$	$(0,0,0,0)$	$A_{2,10}(t)$	$(0,0,0,0)$	$A_{3,10}(t)$	$(0,0,0,0)$	$B_{1,10}(t)$	$(0,0,0,0)$	$B_{2,10}(t)$	$(0,0,0,1)$
$A_{1,11}(t)$	$(0,0,1,1)$	$A_{2,11}(t)$	$(0,0,0,0)$	$A_{3,11}(t)$	$(0,0,0,0)$	$B_{1,11}(t)$	$(1,0,0,0)$	$B_{2,11}(t)$	$(0,0,0,0)$
$A_{1,12}(t)$	$(0,0,0,0)$	$A_{2,12}(t)$	$(0,0,1,0)$	$A_{3,12}(t)$	$(0,0,0,0)$	$B_{1,12}(t)$	$(0,0,0,0)$	$B_{2,12}(t)$	$(1,0,0,0)$
$A_{1,13}(t)$	$(0,0,0,0)$	$A_{2,13}(t)$	$(0,0,0,0)$	$A_{3,13}(t)$	$(0,0,1,1)$	$B_{1,13}(t)$	$(0,1,0,0)$	$B_{2,13}(t)$	$(0,0,0,0)$
$A_{1,14}(t)$	$(0,0,0,0)$	$A_{2,14}(t)$	$(0,0,0,1)$	$A_{3,14}(t)$	$(0,0,0,0)$	$B_{1,14}(t)$	$(0,0,0,0)$	$B_{2,14}(t)$	$(0,1,0,0)$

6.3 Our Scheme: Interactive Verification for Loops

In this section, we use the previously described Block Programs to develop the full solution to the verification of a "loop" computation. Since the theoretical results in Section 6.2 apply to all general circuits that have repeated substructures, which are not limited to those circuits that a piece of "loop" code is mapped to, we are not directly ready for designing the verification protocol for a "loop" computation. Thus, in Section 6.3.1, we further refine the theoretical results we have developed in Section 6.2, to loop-specific results which are more appropriate for the blocks to which a piece of "loop" code is mapped. Then, in Section 6.3.2, we demonstrate our verification protocol for "loop" computations using Block Programs.

6.3.1 Theoretical Results for Loop Circuits

As stated in Section 6.1, the computation task is a piece of "loop" code written in a high-level programming language. In the verification stage, this piece of code is transformed into an arithmetic circuit form using existing compilers such as [9, 10, 14]. We call this arithmetic circuit which we map a piece of "loop" code to a *loop circuit*.

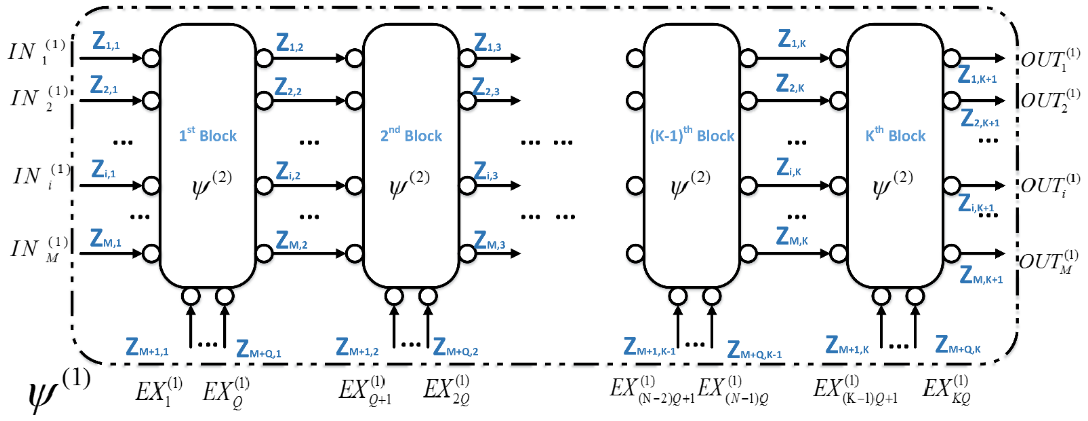


Figure 6.3 A Loop Circuit

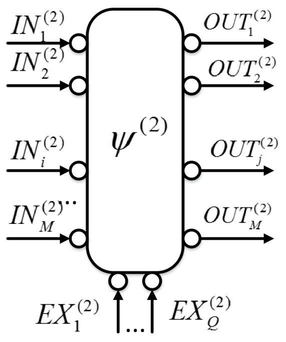


Figure 6.4 One Single Block

It is straightforward to abstract the structure of this circuit as follows. As shown in Figure 6.3, a loop circuit is a series of identical blocks, the output of each of which serves as the input for the next. Each block, which is an arithmetic circuit or boolean circuit, is actually an execution of the loop body of the original "loop" code. Meanwhile, since each execution of the loop body may have extra inputs from the outside of the whole piece of code, (e.g. in the big data processing scenarios) each block may have corresponding extra inputs from the outside of the circuit. The block is demonstrated in Figure 6.4, where $\mathtt{EX}_1^{(2)}, \mathtt{EX}_2^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}$ are the extra inputs.

In Section 6.2, we show that for a circuit that has repeated substructures there exists a BP that computes that circuit. We have already introduced the theoretical results that show how to construct the corresponding BP, and how to determine whether a given inputs/outputs tuple is valid for that circuit using BPs. Since a loop circuit has repeated substructures, we can easily apply our previous results to loops.

Now, we consider a loop circuit as shown in Figure 6.3. The circuit's functionality is to compute a function $\Psi : \mathbb{F}^{M+K\cdot Q} \mapsto \mathbb{F}^M$. The circuit is composed of identical blocks as shown in Figure 6.4, denoted by $\psi : \mathbb{F}^{M+Q} \mapsto \mathbb{F}^M$. The block ψ can also be formulated as M functions: $\psi_j : \mathbb{F}^{M+Q} \mapsto \mathbb{F}$, for $j = 1, 2, \cdots, M$. By Theorem 17, we can construct a BP $\mathcal{Q}$ that computes Ψ, where $\mathcal{Q}$ consists of a target polynomial $D(t)$, $M+Q$ sets of $K\cdot(M+Q)+M$ polynomials (See Figure 6.2 for notations): $\{A_{1,w}(t)\}$, $\{A_{2,w}(t)\}$, $\cdots$, $\{A_{M,w}(t)\}$, and M sets of $K\cdot(M+Q)+M$ polynomials: $\{B_{1,w}(t)\}$, $\{B_{2,w}(t)\}$, $\cdots$, $\{B_{M,w}(t)\}$, where w is the index which represents the labels of wires and $K\cdot(M+Q)+M$ is the number of the wires.

Since a "loop" circuit has a regular structure, we label the wires with a pair of indices and get explicit expressions of the corresponding Block Programs. Then we can simplify these Block Programs into succinct expressions denoted by $A_k(t), k = 1, 2, \cdots K$. Namely, we let w be (w_1, w_2) and denote each wire by $\mathtt{z}_{w_1,w_2}$ as shown in Figure 6.3. Correspondingly, each polynomial in $\mathcal{Q}$ is denoted by $\{A_{v,(w_1,w_2)}(t)\}$ or $\{B_{j,(w_1,w_2)}(t)\}$ where $v, w_1 \in \{1, 2, \cdots, M, M+1, \cdots, M+Q\}$, $j \in \{1, 2, \cdots, M\}$, and $w_2 \in \{1, 2, \cdots, K+1\}$. The explicit expressions of $\mathcal{Q}$ are as follows: Let $\mathcal{K}$ be the set of indices, $\mathcal{K} = \{1, 2, \cdots, K\}$. If $\sigma_1, \sigma_2, \cdots, \sigma_K$ are distinct values, each of which is picked from $\mathbb{F}$, then,

$$A_{v,(w_1,w_2)}(t) = \sum_{k\in\mathcal{K}} A_{v,(w_1,w_2)}(\sigma_k) \cdot \frac{\prod\limits_{k^*\in\mathcal{K},k^*\neq k}(t-\sigma_{k^*})}{\prod\limits_{k^*\in\mathcal{K},k^*\neq k}(\sigma_k-\sigma_{k^*})} \tag{6.3.1}$$

and

$$B_{j,(w_1,w_2)}(t) = \sum_{k\in\mathcal{K}} B_{j,(w_1,w_2)}(\sigma_k) \cdot \frac{\prod\limits_{k^*\in\mathcal{K},k^*\neq k}(t-\sigma_{k^*})}{\prod\limits_{k^*\in\mathcal{K},k^*\neq k}(\sigma_k-\sigma_{k^*})}, \tag{6.3.2}$$

where $A_{v,(w_1,w_2)}(\sigma_k)$ and $B_{j,(w_1,w_2)}(\sigma_k)$ are determined by the interconnection of the blocks in the circuit: $A_{v,(w_1,w_2)}(\sigma_k)$ is 1 if $\mathtt{z}_{w_1,w_2}$ is the v-th input of the k-th block and 0 otherwise; $B_{j,(w_1,w_2)}(\sigma_k)$ is 1 if $\mathtt{z}_{w_1,w_2}$ is the j-th output from the k-th block and 0 otherwise. From the circuit's structure, we can simplify above as follows: $A_{v,(w_1,w_2)}(\sigma_k)$ is 1 if $w_1 = v, w_2 = k$ and 0 otherwise; $B_{j,(w_1,w_2)}(\sigma_k)$ is 1 if $w_1 = j, w_2 = k+1$ and 0 otherwise.

Thus,

$$A_{v,(w_1,w_2)}(t) = \begin{cases} \frac{\prod\limits_{k\in\mathcal{K},k\neq w_2}(t-\sigma_k)}{\prod\limits_{k\in\mathcal{K},k\neq w_2}(\sigma_{w_2}-\sigma_k)} & \text{if } {w_1=v,\text{and} \atop 1\leq w_2\leq K,} \\ 0 \text{ otherwise} \end{cases}. \tag{6.3.3}$$

$$B_{j,(w_1,w_2)}(t) = \begin{cases} \frac{\prod\limits_{k\in\mathcal{K},k\neq w_2-1}(t-\sigma_k)}{\prod\limits_{k\in\mathcal{K},k\neq w_2-1}(\sigma_{(w_2-1)}-\sigma_k)} & \text{if } {w_1=j,\text{ and} \atop 2\leq w_2\leq K+1,} \\ 0 \text{ otherwise} \end{cases}. \tag{6.3.4}$$

If we define

$$A_k(t) = \frac{\prod\limits_{k'\in\mathcal{K},k'\neq k}(t-\sigma_{k'})}{\prod\limits_{k'\in\mathcal{K},k'\neq k}(\sigma_k-\sigma_{k'})} \tag{6.3.5}$$

then, the block program can be expressed by $A_k(t)$ where $k = 1, 2, \cdots, K$:

$$A_{v,(w_1,w_2)}(t) = \begin{cases} A_{w_2}(t) & \text{if } w_1 = v, 1 \leq w_2 \leq K \\ 0 & \text{otherwise.} \end{cases} \tag{6.3.6}$$

$$B_{j,(w_1,w_2)}(t) = \begin{cases} A_{w_2-1}(t) & \text{if } w_1 = j, 2 \leq w_2 \leq K+1 \\ 0 & \text{otherwise.} \end{cases} \tag{6.3.7}$$

If we express the corresponding Block Programs by these $A_k(t)$, instantiate the wire values, and plug them into Definition 2, we directly have the following result for loop circuits.

Corollary 18. *We consider a loop circuit which computes a function $\Psi : \mathbb{F}^{M+K\cdot Q} \mapsto \mathbb{F}^M$. As in Figure 6.3, this loop circuit is composed of successive blocks which are identical and denoted by $\psi : \mathbb{F}^{M+Q} \mapsto \mathbb{F}^M$. Naturally, the block can be formulated as M functions:*

$\psi_j : \mathbb{F}^{M+Q} \mapsto \mathbb{F}$, $j = 1, 2, \cdots, M$. Let $\mathcal{Q}$ be the block program that computes Ψ using block ψ. Then, for inputs $(\mathsf{z}_{1,1}, \mathsf{z}_{2,1}, \cdots, \mathsf{z}_{M,1}) \in \mathbb{F}^M$ and the extra inputs $\mathsf{z}_{M+1,1}$, $\mathsf{z}_{M+2,1}$, $\cdots$, $\mathsf{z}_{M+Q,1}$, $\cdots$, $\mathsf{z}_{M+1,K}$, $\mathsf{z}_{M+2,K}$, $\cdots$, $\mathsf{z}_{M+Q,K}$, all of which are in $\mathbb{F}$, the corresponding outputs of the circuit are $(\mathsf{z}_{1,K+1}, \mathsf{z}_{2,K+1}, \cdots, \mathsf{z}_{M,K+1}) \in \mathbb{F}^M$, iff there exist coefficients (they are actually the inner wire values) $(\mathsf{z}_{1,2}, \cdots, \mathsf{z}_{M,2}) \in \mathbb{F}^M$, $(\mathsf{z}_{1,3}, \cdots, \mathsf{z}_{M,3}) \in \mathbb{F}^M$, $\cdots$, $(\mathsf{z}_{1,K}, \cdots, \mathsf{z}_{M,K}) \in \mathbb{F}^M$ such that $D(t)$ divides $P_j(t)$ $(j = 1, 2, \cdots, M)$ where

$$P_j(t) = \psi_j\left(\sum_{k=1}^{K} \mathsf{z}_{1,k} \cdot A_k(t), \cdots, \sum_{k=1}^{K} \mathsf{z}_{M,k} \cdot A_k(t), \sum_{k=1}^{K} \mathsf{z}_{M+1,k} \cdot A_k(t), \cdots, \sum_{k=1}^{K} \mathsf{z}_{M+Q,k} \cdot A_k(t)\right) - \left(\sum_{k=1}^{K} \mathsf{z}_{j,k+1} \cdot A_k(t)\right). \tag{6.3.8}$$

In other words, there exists a polynomial $H_j(t)$ for each $P_j(t)$ $(j = 1, 2, \cdots, M)$, such that $D(t) \cdot H_j(t) = P_j(t)$.

Proof. Since the loop circuit is a circuit which is composed of identical blocks, by Definition 2, for the purported intermediate wire values $(\mathsf{z}_{1,2},\cdots,\mathsf{z}_{M,2})$, $(\mathsf{z}_{1,3},\cdots,\mathsf{z}_{M,3})$, $\cdots$, $(\mathsf{z}_{1,K},\cdots,\mathsf{z}_{M,K})$, it suffices to prove (6.3.8) is equivalent to (6.2.1) in the context of loop circuits. In the context of loop circuits, (6.2.1) is:

$$P_j(t) = \psi_j\left(\sum_{w_1=1}^{M+Q} \sum_{w_2=1}^{K+1} \mathsf{z}_{w_1,w_2} \cdot A_{1,(w_1,w_2)}(t), \cdots, \sum_{w_1=1}^{M+Q} \sum_{w_2=1}^{K+1} \mathsf{z}_{w_1,w_2} \cdot A_{M+Q,(w_1,w_2)}(t)\right) - \left(\sum_{w_1=1}^{M+Q} \sum_{w_2=1}^{K+1} \mathsf{z}_{w_1,w_2} \cdot B_{j,(w_1,w_2)}(t)\right). \tag{6.3.9}$$

If we instantiate $A_{v,(w_1,w_2)}(t)$, $B_{j,(w_1,w_2)}(t)$ using (6.3.6) and (6.3.7), we immediately get (6.3.8). The conclusion follows. □

6.3.2 Our Interactive Verification for Loops

Corollary 18 implies a way to verify a result computed by a "loop" circuit using BPs. To convince $\mathcal{V}$ that the result is correct, by the Corollary 18, it suffices to show the existence of those intermediate results and the polynomial $H_j(t)$ which satisfy the divisibility $D(t) \cdot H_j(t) = P_j(t)$.

The divisibility itself can be checked probabilistically: for polynomials $P_j(t)$, $H_j(t)$ and $D(t)$, $\mathcal{V}$ randomly picks $\tau \in \mathbb{F}$ and checks whether $D(\tau) \cdot H_j(\tau) = P_j(\tau)$. (We will show how to evaluate $D(\tau)$, $H_j(\tau)$ and $P_j(\tau)$ later.) If the result is correct, $\mathcal{P}$ must be able to find $H_j(t)$ such that $P_j(t) = H_j(t) \cdot D(t)$, then for any $\tau \in \mathbb{F}$, $D(\tau) \cdot H_j(\tau) = P_j(\tau)$. If the result is not correct, then for any $H_j(t)$, $D(\tau) \cdot H_j(\tau) \neq P_j(\tau)$ except with a small probability.

To check the existence of the intermediate results and the polynomial $H_j(t)$, one naive idea is to let $\mathcal{P}$ output the intermediate results and $H_j(t)$, and let $\mathcal{V}$ evaluate $H_j(\tau)$ and use the intermediate results to check the divisibility. Instead of this expensive approach, we use a "commit and query" method. Roughly speaking, we have $\mathcal{P}$ commit to the intermediate results and $H_j(t)$ first. Then, when $\mathcal{V}$ needs to use the intermediate results and $H_j(\tau)$, he will query $\mathcal{P}$ to get purported values and finally check the divisibility. (We will show the "commit and query" method in details later.)

Now we show how we evaluate $D(\tau)$, $H_j(\tau)$ and $P_j(\tau)$. For $D(\tau)$, $\mathcal{V}$ can construct $D(t)$ himself and evaluate $D(\tau)$ himself. In our protocol, neither $\mathcal{V}$ nor $\mathcal{P}$ will materialize $P_j(\tau)$. $P_j(t)$ involves three sets of polynomials: the polynomials that abstract the outputs of the blocks: $G_j(t) = \sum_{k=1}^{K} \mathtt{Z}_{j,k+1} \cdot A_k(t), j = 1, 2 \cdots M$; the polynomials that abstract the inputs of the blocks: $f_{\mathtt{IN}_i}(t) = \sum_{k=1}^{K} \mathtt{Z}_{i,k} \cdot A_k(t), i = 1, 2 \cdots M$; and the polynomials that abstract the extra inputs of the blocks: $f_{\mathtt{EX}_u}(t) = \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1) \cdot Q] + u} \cdot A_k(t), u = 1, 2, \cdots, Q$, where $\mathtt{EX}^{(1)}_{(k-1) \cdot Q + u} = \mathtt{Z}_{u+M,k}$. $\mathcal{V}$ can construct $f_{\mathtt{EX}_u}(t)$ and evaluate $f_{\mathtt{EX}_u}(\tau)$ himself. We observe that $G_j(t)$ and $f_{\mathtt{IN}_i}(t)$ are the only parts in the divisibility equation (6.3.8) that involve the intermediate results. When evaluating them, neither $\mathcal{V}$ nor $\mathcal{P}$ genuinely materializes the polynomial. We view $G_j(t)$, $f_{\mathtt{IN}_i}(t)$ and $H_j(t)$ to be linear functions, denoted by π_{G_j} $(j = 1, \cdots, M)$, $\pi_{\mathtt{IN}_i}$ $(i = 1, \cdots, M)$, π_{H_j} $(j = 1, \cdots, M)$, respectively. We have $\mathcal{P}$ commit to them first[1], then have $\mathcal{V}$ evaluate these polynomials through querying $\mathcal{P}$ on these committed functions, as in [8, 9, 10]. More specifically,

[1]Since the commit/decommit and corresponding linearity tests are mature techniques as in [8, 9, 10], we omit the details here

as a PCP style protocol, in our protocol, $\mathcal{P}$ holds a proof l and $\mathcal{V}$ holds a query q. When queried with q, $\mathcal{P}$ responds with the evaluation of the linear function $\pi(q) = \langle l, q\rangle$ (where $\langle \cdot, \cdot \rangle$ is the inner product of two vectors), which should be the evaluation of $l(t)$ at the point $t = \tau$. For example, if l represents the coefficient vector of the polynomial $l(t)$ and $q = (1, \tau, \tau^2, \cdots,)$, then $\pi(q)$ is $l(\tau)$. To evaluate $G_j(\tau)$, $\mathcal{V}$ queries $\mathcal{P}$ with: $q_G = (A_1(\tau), A_2(\tau)\cdots, A_{K-1}(\tau))$. To evaluate $f_{\mathtt{IN}_i}(\tau)$, $\mathcal{V}$ queries $\mathcal{P}$ with: $q_{IN} = (A_2(\tau), A_3(\tau)\cdots, A_K(\tau))$. To evaluate $H_j(\tau)$, $\mathcal{V}$ queries $\mathcal{P}$ with: $q_H = (1, \tau, \tau^2, \cdots)$. The length of q_H is equal to the highest degree of all $H_j(t)$ $(j = 1, \cdots M)$.

So far we have demonstrated all the techniques we will use. Now we show our interactive verification protocol as in Figure 6.5. Note that before verification, $\mathcal{V}$ sends the loop program and $(\mathtt{Z}_{1,1}, \mathtt{Z}_{2,1}, \cdots, \mathtt{Z}_{M,1})$ $(\mathtt{Z}_{M+1,1}, \cdots, \mathtt{Z}_{M+Q,1}) \cdots (\mathtt{Z}_{M+1,K}, \mathtt{Z}_{M+2,K}, \cdots, \mathtt{Z}_{M+Q,K})$ to $\mathcal{P}$; on these inputs $\mathcal{P}$ computes and returns the results: $(\mathtt{Z}_{1,K+1}, \cdots, \mathtt{Z}_{M,K+1})$. Our idea is to prove that the wire values between repeated substructures are calculated correctly, and then prove that the blocks themselves are computed correctly, rather than trying to verify the entire circuit in one round. Let the loop circuit contain K executions of the loop body. We view the whole circuit from a two-layer perspective. In the first layer's perspective, the circuit consists of one big block, which we call the first layer block and denote by a function $\psi^{(1)} : \mathbb{F}^{M+K\cdot Q} \mapsto \mathbb{F}^M$. In the second layer's perspective, the circuit consists of K smaller blocks, which we call the second layer block and denote by a function $\psi^{(2)} : \mathbb{F}^{M+K\cdot Q/K} \mapsto \mathbb{F}^M$. Each of the second layer block is one execution of the loop body. We reduce the task of verifying (for $j = 1, 2, \cdots, M$)

$$\mathtt{OUT}_j^{(1)} = \psi_j^{(1)}(\mathtt{IN}_1^{(1)}, \cdots, \mathtt{IN}_M^{(1)}, \mathtt{EX}_1^{(1)}, \cdots, \mathtt{EX}_{Q\cdot K}^{(1)}), \tag{6.3.10}$$

to the task of verifying (for $j = 1, 2, \cdots, M$)

$$\mathtt{OUT}_j^{(2)} = \psi_j^{(2)}(\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}, \mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}), \tag{6.3.11}$$

(where $\mathtt{IN}_1^{(1)}, \cdots, \mathtt{IN}_M^{(1)}$ and $\mathtt{EX}_1^{(1)}, \cdots, \mathtt{EX}_{Q\cdot K}^{(1)}$ are respectively the inputs and the extra inputs to the loop circuit, and $\mathtt{OUT}_1^{(1)}, \cdots, \mathtt{OUT}_M^{(1)}$ are the outputs from the loop circuit, as

Setup
Step 1: $\mathcal{V}$ randomly generates $\tau \in \mathbb{F}$ and $q_H = (1, \tau, \tau^2, \cdots, \tau^V)$, $q_G = (A_1(\tau), A_2(\tau) \cdots, A_{K-1}(\tau))$, $q_{IN} = (A_2(\tau), A_3(\tau) \cdots, A_K(\tau))$
Step 2: $\mathcal{V}$ generates the commitment query according to the commitment protocol as in Pepper [8], and sends it to $\mathcal{P}$;
Step 3: $\mathcal{V}$ generates $\sigma_1, \cdots, \sigma_K$ and constructs the BP for the circuit.

Verification
Step 4: Both $\mathcal{P}$ and $\mathcal{V}$ computes $\mathtt{IN}_i^{(1)}$, $\mathtt{EX}_{(k-1)\cdot Q+u}^{(1)}$, and $\mathtt{OUT}_j^{(1)}$: $\mathtt{IN}_i^{(1)} \leftarrow \mathtt{Z}_{i,1}$, $(i = 1, 2, \cdots, M)$; $\mathtt{EX}_{(k-1)\cdot Q+u}^{(1)} \leftarrow \mathtt{Z}_{u+M,k}$, where $u = 1, 2 \cdots Q$ and $k = 1, 2, \cdots, K$; $\mathtt{OUT}_j^{(1)} \leftarrow \mathtt{Z}_{j,K+1}$, $(j = 1, 2, \cdots, M)$
Step 5: $\mathcal{P}$ computes intermediate results $\mathtt{Z}_{w_1,w_2}$, linear functions π_{H_j}, π_{G_j}, $\pi_{\mathtt{IN}_i}$ for all i, j, replies the commitment information for all proofs.
Step 6: $\mathcal{V}$ sends $\mathcal{P}$ these queries: q_H, q_G, q_{IN},$\mathcal{P}$ replies: $\pi_{H_j}(q_H)$, $\pi_{G_j}(q_G)$, $\pi_{\mathtt{IN}_i}(q_{IN})$ $(i, j \in \{1, 2, \cdots, M\})$
Step 7: $\mathcal{V}$ checks the linearity of these proofs. If any fails, reject;
Step 8: $\mathcal{P}$ sends $\mathtt{OUT}_j^{(2)}$; $\mathcal{V}$ checks whether $\mathtt{OUT}_j^{(2)} = D(\tau) \cdot \pi_{H_j}(q_H) + \pi_{G_j}(q_G) + A_K(\tau) \cdot \mathtt{OUT}_j^{(1)}$, where $j = 1, \cdots, M$. If not, reject.
Step 9: $\mathcal{V}$ updates $\mathtt{IN}_i^{(2)}$, $i = 1, 2, \cdots, M$: $\mathtt{IN}_i^{(2)} = \pi_{\mathtt{IN}_i}(q_{IN}) + \mathtt{IN}_i^{(1)} \cdot A_1(\tau)$; and updates $\mathtt{EX}_u^{(2)}$ by computing: $\mathtt{EX}_u^{(2)} \leftarrow f_{\mathtt{EX}_u}(\tau)$ $(u = 1, 2, \cdots, Q)$.
Step 10: $\mathcal{P}$ convinces $\mathcal{V}$ that $\mathtt{OUT}_j^{(2)} = \psi_j^{(2)}(\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}, \mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)})$
Step 11: $\mathcal{V}$ performs the decommitment check for every proofs. If any fails, $\mathcal{V}$ rejects.
Step 12: If the running of our protocol goes here, accept.

Figure 6.5 Our Verification Protocol

in Figure 6.3; $\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}$ and $\mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}$ are respectively the inputs and extra inputs to one loop body, and $\mathtt{OUT}_1^{(2)}, \cdots, \mathtt{OUT}_M^{(2)}$ are the outputs from that loop body. See Figure 6.4). To verify (6.3.11), $\mathcal{V}$ can compute itself, or let $\mathcal{P}$ perform some other verification protocols like Pinocchio [14], Zaatar [10], etc. We omit the details here.

6.3.3 Security Analysis

Now, we provide the completeness and soundness of our design.

Theorem 19. *(Completeness) As in Figure 6.3, for the inputs* $(\mathtt{Z}_{1,1}, \mathtt{Z}_{2,1}, \cdots, \mathtt{Z}_{M,1})$, $(\mathtt{Z}_{M+1,1}, \mathtt{Z}_{M+2,1}, \cdots, \mathtt{Z}_{M+Q,1})$, $\cdots$, $(\mathtt{Z}_{M+1,K}, \mathtt{Z}_{M+2,K}, \cdots, \mathtt{Z}_{M+Q,K})$, *and* $(\mathcal{P}, \mathcal{V})$ *run the protocol in Figure 6.5. If the results* $(\mathtt{Z}_{1,K+1}, \mathtt{Z}_{2,K+1}, \cdots, \mathtt{Z}_{M,K+1})$ *are correct, then we have:* $Pr\{\mathcal{V}$ *accepts*$\} = 1$.

To prove the completeness(Theorem 19), the idea is to show that an honest prover is able to provide the correct proof associated with the correct results. This correct proof will pass all the checks with probability 1. This is straightforward and we omit the details here. The complete proof is provided in Section 6.5.3.

Theorem 20. *(Soundness) As in Figure 6.3, assume the inputs are* $(\mathtt{z}_{1,1}, \mathtt{z}_{2,1}, \cdots, \mathtt{z}_{M,1})$, $(\mathtt{z}_{M+1,1}, \mathtt{z}_{M+2,1}, \cdots, \mathtt{z}_{M+Q,1})$, $\cdots$, $(\mathtt{z}_{M+1,K}, \mathtt{z}_{M+2,K}, \cdots, \mathtt{z}_{M+Q,K})$*, and* $\mathcal{P}$ *and* $\mathcal{V}$ *proceed according to Figure 6.5.*

There exists a constant $\kappa < 1$ *such that if results* $(\mathtt{z}_{1,K+1}, \mathtt{z}_{2,K+1}, \cdots, \mathtt{z}_{M,K+1})$ *are not correct, then* $Pr\{\mathcal{V}$ *accepts*$\} < \kappa$ *holds for any purported proofs. The probability is over the randomness of both* $\mathcal{V}$ *and* $\mathcal{P}$ *in both phases in our protocol. This is equivalent to: if* $Pr\{\mathcal{V}$ *accepts*$\} > \kappa$*, the purported results are correct.*

The proof of the soundness is in Section 6.5.2.

6.3.4 Improving the Performance through Batching

One question left over is whether, if $\mathcal{V}$ queries different linear proofs, he should re-generate the queries for each proof. If so, the cost of issuing queries will be prohibitive. Our idea is to reuse τ and corresponding divisibility query for all linear proofs. We achieve this by making our protocol work in a batching style, which implies two levels: (1) same queries work over many computation instances, namely $\mathcal{V}$ verifies computation in batches; (2) for one computation instance, $\mathcal{V}$ uses the same τ and corresponding queries for all proofs in the same layer.

This batching technique was firstly proposed in [8], where $\mathcal{V}$ generates one single commitment query $\mathtt{Enc}(r)$(the query to get the commitment information) and one single set of PCP queries for β proofs. In the commitment phase, $\mathcal{P}$ replies to $\mathcal{V}$ with β pieces of commitment information $\mathtt{Enc}(s_1)$, $\mathtt{Enc}(s_2)$, $\cdots$, $\mathtt{Enc}(s_\beta)$, one for each of the β proofs. In the decommitment phase, $\mathcal{V}$ issues the decommitment query based on the commitment query $\mathtt{Enc}(r)$ and the set of PCP queries and receives β answers, one for each of the β proofs. $\mathcal{V}$ will perform the decommitment check separately. It is proved that this batching technique will not impair the soundness of the verification protocol ([8], Appendix C, Theorem C.1). This immediately implies the correctness and soundness of our first level "batching", as that in [10].

For the second level of "batching", we are targeting to use one single commitment query and one single set of PCP queries for all these proofs. Firstly, we notice that the bottom line of batching is that $\mathcal{P}$ is not allowed to output the answer for one proof after he learns whether $\mathcal{V}$ accepts or rejects another proof (the so-called Verifier Rejection Problem [58]) Thus, we move all the decommitment tests to the last step in our protocol, which is performed after all verification is done. Secondly, since our protocol is an interactive protocol, the soundness holds only under the restriction that $\mathcal{V}$'s τ and his verdict in each layer does not help $\mathcal{P}$ deviate from the protocol and generate proofs in the next layer to cheat $\mathcal{V}$. Thus, if we use Zaatar [10] to verify the loop body, we need to guarantee that the randomly generated number for the divisibility test in Zaatar should be different from our τ.

6.4 Performance Evaluation

In this section, we analyze the performance of our design and compare it with existing work. We firstly compare the complexity of the algorithms. Then, we compare the "experimental results" of our design with all existing work. However, we model, rather than measure, their performance. For our design, we built a model of our design's performance based on the latest performance results for [10], while for the others, we used previously published models [10, 14].

6.4.1 Performance Analysis and Comparison

We are targeting at reducing the amortized part of cost (for the verifier) and the proof generation cost (for the prover). Now, in the context of loop circuits, we compare the cost of our protocol with Zaatar and Pinocchio as in Figure 6.6. We use the published performance models of Zaatar [10] and Pinocchio [14]. For Pinocchio, we only list the most burdensome part in Figure 6.6; for Zaatar, given that Zaatar's circuit is formulated into constraints, we view the number of constraints in Zaatar as the number of multiplication gates (circuit size). Then, the number of variables in the constraints (excluding inputs and outputs) in Zaatar equals the circuit size minus the number of input/output wires.

Remember that our verification design has two layers, the first layer proves that the wire values between repeated substructures are calculated correctly, and then the second layer proves that the blocks themselves are computed correctly. At the second layer in our protocol, the result of a single execution of the loop body needs to be verified. We use existing methods as a building block to perform the verification. In Figure 6.6, the last terms in the column of our design's cost model (e.g., $C_{na}^{Z}(1)$, $\frac{1}{K}C_{a}^{Z}(K)$, etc.) represent the cost for the verification of the loop body in our protocol. In our protocol, if $\mathcal{V}$ adopts Zaatar or Pinocchio as the verification block for the loop body, this part of cost will usually be only $\frac{1}{K}$ of the total cost that $\mathcal{V}$ pays if $\mathcal{V}$ uses Zaatar or Pinocchio to verify the whole loop. In particular, the second layer verification(verification of the loop body) also contains amortized cost, which is linear in the size of the circuit associated with the loop body. For simplicity, We choose Zaatar to show our cost model.

Now we show why the amortized cost in our protocol is much less that its counterparts in all existing protocols: our design has the following properties: First, it has more flexibility in amortizing. Our protocol has a benefit that the first part of amortized cost, denoted by C_a^*, can be amortized over instances which have the same loop structure and the same degree of the polynomial that the loop body computes. (Notice: these instances can have different circuits for the loop body, as long as the circuit the loop bodies compute have the same degree.) This property renders our protocol superior to existing works, in which costs can only be amortized to instances that share the same circuit. Namely, even if our amortized cost is the same as its counterparts, our protocol can be amortized to more instances than other protocols. Second, it has smaller costs. Even if the amortized part for verifying the loop body, denoted by $C_a^{\dagger}$, counteracts a part of cost, the total

	Zaatar (Loop ver.)	Pinocchio (Loop ver.)	Our Algm[a]
$\mathcal{V}$'s Total non-amortized cost	$C^Z_{na}(K) = 2\texttt{Dec} + \rho \cdot (6M + 3Q \cdot K + 6\rho_{lin} + 4) \cdot \texttt{Mult}$	$C^P_{na} = 11\texttt{Map}$	$C_{na}(K) = 3M \cdot \texttt{Dec} + \rho \cdot (6M + Q \cdot K + 9M\rho_{lin}) \cdot \texttt{Mult} + C^Z_{na}(1)$
$\mathcal{V}$'s Total amortized cost	$C^Z_a(K) = (2K \cdot C) \cdot \texttt{Enc} + \rho \cdot \rho_{lin} \cdot 2 \cdot (2K \cdot C) \cdot \texttt{Rand} + \rho \cdot [\texttt{Rand} + (\texttt{Div}) \cdot K \cdot C] + (\rho_{lin} \cdot 3 \cdot (2K \cdot C) + (3K \cdot C + 6K \cdot C + K_1 + 3K_2)) \cdot \rho \cdot \texttt{Mult}$	$C^P_a(K) = [10(K \cdot C)] \cdot \texttt{Exp}$	$C_a(K) = (K \cdot D) \cdot [\texttt{Enc} + 2\rho \cdot \rho_{lin} \cdot (\texttt{Rand} + \texttt{Mult}) + 2\rho \cdot \texttt{Mult}] + \rho \cdot (\texttt{Div} + 3 \cdot \texttt{Mult}) \cdot K + \frac{1}{K} C^Z_a(K)$
Task Computing	`Comp`	`Comp`	`Comp`
$\mathcal{P}$: Construct proof vector	$C^Z_p(K) = 3\texttt{Mult} \cdot (K \cdot C) \cdot log^2(K \cdot C)$	$C^P_p(K) = 8\texttt{Muex} \cdot (K \cdot C)$	$C_p(K) = 3\texttt{Mult} \cdot (K \cdot D - K) \cdot log^2(K \cdot D - K) + C^Z_p(1)$
$\mathcal{P}$: Issue PCP responses	$C^Z_i(K) = (\texttt{Oper} + (\rho \cdot (6\rho_{lin} + 4) + 1) \cdot \texttt{Mult}) \cdot (2K \cdot C)$	N/A	$C_i(K) = [\texttt{Oper} + (\rho \cdot (3\rho_{lin} + 2) \cdot M + 1) \cdot \texttt{Mult}] \cdot (K \cdot D) + C^Z_i(1)$

`Map`: cost of bilinear map in $\mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$	`Exp`: cost of exponential operation in $\mathbb{G}$
`Div`: cost of division over $\mathbb{F}$	`Mult`: cost of multiplication over $\mathbb{F}$
`Rand`: cost of pseudorandomly generating an element in $\mathbb{F}$	Q: number of the extra input of the loop body
`Dec`: cost of decryption over $\mathbb{F}$	`Enc`: cost of encryption over $\mathbb{F}$
`Comp`: cost of evaluating the whole circuit	`Oper`: cost of ciphertext add plus multiply
`Muex`: cost of multiplication over exponent	
ρ: number of iteration of verification for one instance	ρ_{lin}: number of iterations of linearity tests in one iteration of verification.
K_1: number of additive terms in the constraints of Ginger, Zaatar's underlying protocol	K_2: number of distinct additive degree-2 terms in the constraints of Ginger, Zaatar's underlying protocol
K: number of executions of the loop body	M: the number of input (or output) of the loop body
D: degree of the polynomial that the loop body computes	C: size of the circuit of the loop body

[a]Note: the costs of our design are calculated in Section 6.6.1, 6.6.2, and 6.6.3.

Figure 6.6 Comparison of Costs in Each Instance

amortized cost in our protocol is still far less than that of Zaatar in the loop circuit. Notice the total amortized cost in our protocol is $[C^*_a + C^\dagger_a] \sim [O(K \cdot D) + O(C)]$. Zaatar has amortized cost which is linear in the size of the whole circuit, denoted by $C^Z_a \sim O(K \cdot C)$. Recall that K is the number of executions of the loop body and D is the degree of the polynomial that the loop body computes. (The loop body is equivalent to an arithmetic circuit and the arithmetic circuit computes a polynomial.) C is the size of the circuit that the loop body is equivalent to. We can prove that the amortized cost in our protocol is usually far less than the amortized cost in existing work.

Although C^*_a and C^Z_a seems similar, in general, they are largely different. We know arithmetic circuits are the standard model for computing polynomials. In other words, an output of an arithmetic circuit is a polynomial in the input variables. The arithmetic complexity is characterized by the size of the arithmetic circuit. From the theory of arithmetic circuit complexity, we know *most* polynomials have high arithmetic complexity ([63], Section 4). More specifically, a detailed analysis using a counting argument shows that most polynomials in n variables and degree d require circuits of size at least $\Omega(\sqrt{\binom{n+d}{d}})$ ([64], Section 3.1). Here *most* means that the number of polynomials that have small circuits (i.e. smaller than the lower bound above) is much smaller than the total number of polynomials. In most cases, to compute an n-variate polynomial of degree d requires a circuit of size at least $\Omega(\sqrt{\binom{n+d}{d}})$. In our context, $n = M + Q$ and $d = D$. Then, by this lower bound, C is at least $\Omega(\sqrt{\binom{M+Q+D}{D}})$ which is much more than D, since $\Omega\left(\sqrt{\binom{M+Q+D}{D}}\right) \sim \Omega\left(\sqrt{\frac{(M+Q+D)^{M+Q+D+\frac{1}{2}}}{(M+Q)^{M+Q+\frac{1}{2}} \cdot D^{D+\frac{1}{2}}}}\right)$ by Stirling's approximation. For Given M and Q, this is an exponential function of D.

Similarly, we improve the prover's cost of proof generation from quasilinear in the size of the loop body (like in Zaatar) to quasilinear in the degree of the loop body, with an additional cost for generating the proof of a single execution of the loop body. This is also a big improvement according to the analysis above.

6.4.2 An Example for Performance Comparison

To have a clear picture of these costs and how powerful our method is in verification of real-world iterative computation, we look at a real loop. We look at an example of successive matrix multiplication, which is widely used in communications, signal processing, and control systems. The code to compute: $xA_0 \cdots A_K$, where x is a $1 \times K$ vector and A_i, $(i = 0, \cdots K)$ are $M \times M$ matrices. It is easy to write this into a "for" loop. Let A be a 1000×1000 matrix and the number of executions be $K = 500$. Let the width of the inputs/outputs be $M = 1000$, the width of the extra inputs for each execution $Q = M^2$, and the circuit size of the loop body $C = M^2$. It is easy to see the polynomial that each loop body computes is quadratic, thus $D = 2$. We use published models ([10, 14]) and instantiate the costs as in Figure 6.7. We can see that, for the amortized cost and the

	Zaatar	Pinocchio	Our Algm[a]
$\mathcal{V}$'s Cost (Non-amortized)	$1.10h$	$9.9ms$	$1281s + 7.70s$
$\mathcal{V}$'s Cost (Amortized)	$96.76h$	$164.3h$	$0.29s + 696.67s$
Task Computing	$160s$	$160s$	$160s$
$\mathcal{P}$: Construct proof	$111.34h$	$445.56h$	$38.6ms + 381s$
$\mathcal{P}$: Issue responses	$124.38h$	N/A	$158.8s + 895.5s$

`Map`: $0.9ms$

`Exp`: $118.3\mu s$, fixed base, optimized for twist curve

`Div`: $3\mu s$, 220-bit

`Mult`: $320ns$, 220-bit

`Rand`: $260ns$, 220-bit

`Muex`: $401.0\mu s$, 254-bit, optimized for twist curve

`Oper`:$130\mu s$, 220-bit

ρ: 8

ρ_{lin}: 20

`Dec`: $170\mu s$, 220 bits

`Enc`: $88\mu s$, 220 bits

[a]Note: we list the cost for both the first layer and the second layer in our design, connected by "+".

Figure 6.7 An Loop Example and Its Cost

prover's cost, our protocol is far superior to the other two. This also implies that in our protocol smaller breakeven batching size suffices to guarantee that the average cost for verification per instance is less than re-computing.

6.4.3 Further Discussion

If we examine our protocol carefully, a natural question will arise: why cannot the block be of size $K_0 \cdot C$ such that $K_0 \neq 1$? Moreover, if we group K_0 successive executions of the loop body into large blocks ($K_0 < K$), then we can further group several sequential big blocks into a bigger block. The whole circuit can be viewed as a multiple-layer structure. More specifically, we can view the whole circuit as one big block, which we call the first layer block and denote by a function $\psi^{(1)} : \mathbb{F}^{M+K\cdot Q} \mapsto \mathbb{F}^M$. In the second layer's perspective, the circuit consists of K_0 smaller blocks, which we call the second layer block and denote by a function $\psi^{(2)} : \mathbb{F}^{M+K\cdot Q/K_0} \mapsto \mathbb{F}^M$. In each layer, several blocks constitute the circuit. The block in the l-th layer is denoted by a function $\psi^{(l)} : \mathbb{F}^{M+K\cdot Q/(K_0^{l-1})} \mapsto \mathbb{F}^M$. The l-th layer block is composed of K_0 lower layer blocks, i.e. the $l+1$ layer blocks. At the L-th layer, the last layer, each of the blocks is equivalent to one execution of the loop body. Note that the reduction of our verification can go further recursively in this multiple-layer view. According to our current protocol, the correctness of computing $\psi^{(1)}$ is reduced to the correctness of computing $\psi^{(2)}$. Similarly, the correctness of computing $\psi^{(2)}$ is reduced to the correctness of computing $\psi^{(3)}$. We can keep doing this until we meet the last layer. At a first glance, this recursive protocol seems promising. However, it is the first layer that dominates the cost. This implies that this method introduces high cost due to the recursion. More specifically, if we have multiple-layer blocks, then from $\psi^{(1)}(\cdot) = \psi^{(2)}(\psi^{(2)}(\cdots(\psi^{(2)}(\cdot))\cdots))$, we see the degree of the polynomial a block computes increases exponentially. Simple analysis will show that the cost of the first layer dominates and therefore the cost for this recursive method is proportional to $O(D^K)$, which is much more than our current protocol. Recall in our current protocol, the block is one execution of the loop body. Thus, the degree of the polynomial this block computes is D. This is also the reason why sequential circuits – where the output of each gate is an input to the next gate – is the worst-case scenario for our protocol. Such circuits are better tackled with Zaatar, Pinocchio, etc.

6.5 Mathematical Proofs

6.5.1 Complete Proof of Theorem 17

Proof. In Section 6.2, We have constructed a group of interpolation polynomials in Lagrange form. Now, we prove that the constructed polynomials are indeed the BP that computes Ψ.

(1) ($\Longrightarrow$)

Suppose $(z_1, z_2, \cdots, z_{N+N'}) \in \mathbb{F}^{N+N'}$ is a valid assignment of Ψ's inputs and outputs.

Then, there exist intermediate results $z_{N+N'+1}, \cdots, z_W$, which are all the outputs from each block except those which are also outputs of the whole circuit. Thus, according to (6.2.1), we can generate $P_j(t)$ for $j = 1, 2, \cdots, M$ using these z_w $(w = 1, 2, \cdots = W)$ and the BP we have constructed above.

Now we look at $P_j(\sigma_k)$ where $k = 1, 2, \cdots, K$.

$$\begin{aligned} P_j(\sigma_k) =& \psi_j \left(\sum_{w=1}^{W} z_w \cdot A_{1w}(\sigma_k), \cdots, \sum_{w=1}^{W} z_w \cdot A_{Vw}(\sigma_k) \right) \\ & - \left(\sum_{w=1}^{W} z_w \cdot B_{jw}(\sigma_k) \right). \end{aligned} \tag{6.5.1}$$

We have constructed $A_{vw}(t)$ and $B_{jw}(t)$ for $v = 1, 2, \cdots, V$, $j = 1, 2, \cdots, M$ and $k = 1, 2, \cdots, K$. Recall that in our construction, we let $A_{vw}(\sigma_k) = 1$ if z_w is the v-th input to the k-th block, and $A_{vw}(\sigma_k) = 0$ otherwise; we also let $B_{jw}(\sigma_k) = 1$ if z_w is the j-th output from the k-th block, and $B_{jw}(\sigma_k) = 0$ otherwise. Then, if $z_{w_i^\dagger}$ is the v-th input to the k-th block, and $z_{w_j^*}$ is the j-th output from the k-th block, we will have:

$$P_j(\sigma_k) = \psi_j \left(z_{w_1^\dagger}, z_{w_2^\dagger}, \cdots, z_{w_V^\dagger} \right) - \left(z_{w_j^*} \right). \tag{6.5.2}$$

(In the example of the circuit in Figure 6.1, if $j = 2$ and $k = 3$, then $z_{w_1^\dagger}$ is z_{11}, $z_{w_2^\dagger}$ is z_{12}, $z_{w_3^\dagger}$ is z_{13}, and $z_{w_2^*}$ is z_8.)

From the k-th block's functionality, we know that

$$\psi_j \left(z_{w_1^\dagger}, z_{w_2^\dagger}, \cdots, z_{w_V^\dagger} \right) - \left(z_{w_j^*} \right) = 0 \tag{6.5.3}$$

Thus, for all σ_k, where $k = 1, 2, \cdots, K$ (recall K is the number of blocks in the circuit), and for $j = 1, 2, \cdots, M$, we have $P_j(\sigma_k) = 0$. Namely, σ_k, where $k = 1, 2, \cdots, K$, are the roots of the polynomials $P_j(t)$, where $j = 1, 2, \cdots, M$. If we recall the expression of the target polynomial $D(t) = \prod_{k=1}^{K}(t - \sigma_k)$, it is straightforward that σ_k, where $k = 1, 2, \cdots, K$, are the roots of $D(t)$. Given that σ_k, where $k = 1, 2, \cdots, K$, are different from each other, we can conclude that $D(t)$ divides $P_j(t)$, for $j = 1, 2, \cdots, M$.

(2) ($\Longleftarrow$)

Suppose for $(z_1, z_2, \cdots, z_{N+N'}) \in \mathbb{F}^{N+N'}$, there exist coefficients $z_{N+N'+1}, \cdots, z_W$ such that $D(t)$ divides $P_j(t)$, $(j = 1, 2, \cdots, M)$, where $P_j(t)$ is defined as in (6.2.1).

Then, each of $D(t)$'roots is also a root of $P_j(t)$ for $j = 1, 2, \cdots, M$. Namely, σ_k, where $k = 1, 2, \cdots, K$, are also the roots of the polynomials $P_j(t)$, where $j = 1, 2, \cdots, M$. Thus, for $k = 1, 2, \cdots, K$ and $j = 1, 2, \cdots, M$ we have:

$$P_j(\sigma_k) = 0 \tag{6.5.4}$$

This can be represented as

$$\begin{aligned} 0 =& \psi_j \left(\sum_{w=1}^{W} z_w \cdot A_{1w}(\sigma_k), \cdots, \sum_{w=1}^{W} z_w \cdot A_{Vw}(\sigma_k) \right) \\ & - \left(\sum_{w=1}^{W} z_w \cdot B_{jw}(\sigma_k) \right) \end{aligned} \tag{6.5.5}$$

Now, we assign $(z_1, z_2, \cdots, z_{N+N'})$ to the input/output wires of the circuit. We also assign $z_{N+N'+1}, \cdots, z_W$ to the corresponding output wires of all the blocks.

Recall that in our construction, we let $A_{vw}(\sigma_k) = 1$ if z_w is the v-th input to the k-th block, and $A_{vw}(\sigma_k) = 0$ otherwise; we also let $B_{jw}(\sigma_k) = 1$ if z_w is the j-th output from the k-th block, and $B_{jw}(\sigma_k) = 0$ otherwise. Then, if $z_{w_v^\dagger}$ is the v-th input to the k-th block, and $z_{w_j^*}$ is the j-th output from the k-th block, we can simplify (6.5.5) as follows:

$$\psi_j \left(z_{w_1^\dagger}, z_{w_2^\dagger}, \cdots, z_{w_V^\dagger} \right) - \left(z_{w_j^*} \right) = 0 \tag{6.5.6}$$

This implies that $z_{w_i^\dagger}$ where $i = 1, 2, \cdots, M$ and $z_{w_j^*}$ where $j = 1, 2, \cdots, M$ is a valid assignment of the k-th block's inputs and outputs.

Similarly, for all $k = 1, 2, \cdots, K$, our assignment covers valid inputs and outputs of the corresponding block. Namely, $(z_1, z_2, \cdots, z_{N+N'})$ is a valid assignment of the circuit's inputs and outputs.

From (1) and (2), we know that the constructed polynomials are indeed the BPs that compute Ψ. □

6.5.2 Proof of Soundness

To better illustrate the logical flow of the soundness proof, we first provide the following simple experiment. Suppose $\mathcal{P}$ must choose between two boxes (the left and the right box), each containing a number of white balls and black balls. The right box stands for the correct proof, while the left box for the wrong proof. White balls represent tests that pass, black balls represent tests that fail. By the completeness argument, we know that all the balls in the right box are white. On the other hand, the left box contains both white and black balls. The prover $\mathcal{P}$ chooses one of the boxes, and places it in a dark room (the room stands for the prover's commitment). The verifier $\mathcal{V}$ has to test the box, to see whether the prover chose the wrong proof. To accomplish this, $\mathcal{V}$ enters the dark room and picks a ball from the box. Then $\mathcal{V}$ exits the room and looks at the ball. Our proof flows as follows. 1. We first show that if $\mathcal{V}$ were to randomly pick a ball from the left box, then $Pr\{\text{the ball is white}\} < \kappa_0^M$, where κ_0^M is some small positive constant (that is, the left box contains mostly black balls). 2. We then reason that, if for the box in the dark room we have $Pr\{\text{the ball is white}\} > \kappa_0^M$, then the box has to be the right box.

Our soundness proof is a bit less straightforward. To make it work, we have to first condition our probabilities on the event that all proofs provided by the prover are linear. In Lemma 22, we show that, if the verifier provides the wrong result (and consequently the wrong proof), then $Pr\{\mathcal{V} \text{ accepts } | \text{all purported proofs are linear } \} < \kappa_0^M$. Now looking back to Theorem 20, to prove the soundness it suffices to show the existence of κ. We start our proof by providing an explicit value κ^*. It now suffices to prove that, if the probability that the verifier accepts (not conditioned on anything) is greater than κ^*, then the purported results are correct. So we show that, if the probability that the verifier accepts is greater than κ^*, then we have $Pr\{\mathcal{V} \text{ accepts } | \text{all purported proofs are linear}\} > \kappa_0^M$, Finally, we bootstrap our argument as above, reasoning that unless the results of the computation are correct, a contradiction ensues with Lemma 22.

Let us proceed. For simplicity, we extract a part of purely mathematical transformation from our proof into Lemma 21.

Lemma 21. *In the context of loop circuits (as in Figure 6.3),*

$$D(\tau) \cdot H_j(\tau) = \psi_j^{(2)}\Bigg(\sum_{k=2}^{K} \mathtt{Z}_{1,k} \cdot A_k(\tau) + \mathtt{IN}_1^{(1)} \cdot A_1(\tau), \cdots,$$
$$\sum_{k=2}^{K} \mathtt{Z}_{M,k} \cdot A_k(\tau) + \mathtt{IN}_M^{(1)} \cdot A_1(\tau), \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+1} \cdot A_k(\tau),$$
$$\cdots, \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+Q} \cdot A_k(\tau)\Bigg) - \Bigg(\sum_{k=1}^{K-1} \mathtt{Z}_{j,k+1} \cdot A_k(t)$$
$$+ A_K(\tau) \cdot \mathtt{OUT}_j^{(1)}\Bigg) \tag{6.5.7}$$

is equivalent to $\mathtt{OUT}_j^{(2)} = \psi_j^{(2)}(\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}, \mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)})$*, for inputs* $\mathtt{IN}_1^{(1)}, \cdots, \mathtt{IN}_M^{(1)}$*, extra inputs* $\mathtt{EX}_1^{(1)}, \cdots, \mathtt{EX}_{Q\cdot K}^{(1)}$*, and outputs* $\mathtt{OUT}_1^{(1)}, \cdots, \mathtt{OUT}_M^{(1)}$*, and* $\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}$*,* $\mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}$*,* $\mathtt{OUT}_1^{(2)}, \cdots, \mathtt{OUT}_M^{(2)}$ *are defined in terms of* τ *as in Figure 6.5.*

A simple change of variable suffices to prove it.

Lemma 22. *Let* $\mathcal{V}$ *and* $\mathcal{P}$ *run our protocol as in Figure 6.5. If the results of the computation task are not correct, for the cases that all purported proofs* π_{G_j} $(j = 1, \cdots, M)$*,* $\pi_{\mathtt{IN}_i}$ $(i = 1, \cdots, M)$*,* π_{H_j} $(j = 1, \cdots, M)$ *are linear functions, we have:*

$$Pr\{\mathcal{V} \textit{ accepts } | \textit{all purported proofs are linear}\} < \kappa_0^M, \tag{6.5.8}$$

where $\kappa_0 = \frac{K \cdot D}{|\mathbb{F}|}$ *or* κ_{Zaatar}*,* D *is the degree of the polynomial that computes the loop body and* K *is the number of executions of the loop body. The probability is over the randomness of both* $\mathcal{V}$ *and* $\mathcal{P}$ *in both phases our protocol.*

Proof. Under the condition that all purported proofs π_{G_j} $(j = 1, \cdots, M)$, $\pi_{\mathtt{IN}_i}$ $(i = 1, \cdots, M)$, π_{H_j} $(j = 1, \cdots, M)$ are linear functions, all the linearity tests pass. Then, $\mathcal{V}$ accepts if

$$\mathtt{OUT}_j^{(2)} = \psi_j^{(2)}(\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}, \mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}). \tag{6.5.9}$$

This is checked by computing himself or using existing protocols, like Zaatar or Pinocchio. If he computes himself, by Lemma 21, the verifier accepts only when the following holds:

$$D(\tau)\cdot H_j(\tau) = \psi_j^{(2)}\Big(\sum_{k=2}^{K} \mathtt{Z}_{1,k}\cdot A_k(\tau) + \mathtt{IN}_1^{(1)}\cdot A_1(\tau), \cdots, \\ \sum_{k=2}^{K} \mathtt{Z}_{M,k}\cdot A_k(\tau) + \mathtt{IN}_M^{(1)}\cdot A_1(\tau), \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+1}\cdot A_k(\tau), \cdots, \\ \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+Q}\cdot A_k(\tau)\Big) - [\sum_{k=1}^{K-1} \mathtt{Z}_{j,k+1}\cdot A_k(\tau) + A_K(\tau)\cdot \mathtt{OUT}_j^{(1)}]. \tag{6.5.10}$$

This is a simple test of the following at the point $t=\tau$:

$$D(t)\cdot H_j(t) = \psi_j^{(2)}\Big(\sum_{k=2}^{K} \mathtt{Z}_{1,k}\cdot A_k(t) + \mathtt{IN}_1^{(1)}\cdot A_1(t), \cdots, \\ \sum_{k=2}^{K} \mathtt{Z}_{M,k}\cdot A_k(t) + \mathtt{IN}_M^{(1)}\cdot A_1(t), \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+1}\cdot A_k(t), \cdots, \\ \sum_{k=1}^{K} \mathtt{EX}^{(1)}_{[(k-1)\cdot Q]+Q}\cdot A_k(t)\Big) - [\sum_{k=1}^{K-1} \mathtt{Z}_{j,k+1}\cdot A_k(t) + A_K(t)\cdot \mathtt{OUT}_j^{(1)}]. \tag{6.5.11}$$

However, since the results are not correct, by Corollary 18, for any $j = 1, 2, \cdots, M$, there is no $H_j(t)$ for which (6.5.11) holds. Thus, $\mathcal{V}$ wrongly accepts only if τ is a root of (6.5.11). By Schwartz-Zippel lemma, the probability that $\mathcal{V}$ wrongly accepts is bounded above by $\frac{K\cdot D}{|\mathbb{F}|}$, where D is the degree of the polynomial that computes the loop body and K is the number of executions of the loop body. If $\mathcal{V}$ uses Zaatar to check (6.5.9), then the probability is bounded above by κ_{Zaatar} (refer to [10], Apdx. A.2). Notice the divisibility tests will run M times. Thus, if the results of the computation are not correct, then, $Pr\{\mathcal{V}$ accepts| all purported proofs are linear $\} < \kappa_0^M$, where $\kappa_0 = \frac{K\cdot D}{|F|}$ or κ_{Zaatar}. The probability is over the randomness of $\mathcal{V}$ and $\mathcal{P}$ in both phases our protocol. □

The proof of Theorem 20 is as follows (our articulation follows [10]):

Proof. We address the general cases where maybe not all purported proofs π_{G_j} ($j = 1, \cdots, M$), $\pi_{\mathtt{IN}_i}$ ($i = 1, \cdots, M$), π_{H_j} ($j = 1, \cdots, M$) are linear functions. We define $\kappa^* = max\{(1-3\delta+6\delta^2)^{\rho_{lin}}, 6M\delta + \kappa_0^M\}$ (where ρ_{lin} is the number of linearity tests, and

$0 < \delta < \delta^*$, δ^* is the lesser root of $6\delta^2 - 3\delta + 2/9 = 0$), and claim this κ^* is the κ we are looking for. Now, we prove our claim. It suffices to prove: if $Pr\{\mathcal{V}\text{ accepts}\} > \kappa^*$, then the purported results are correct. (This statement is equivalent to the requirement for κ.) The probability is over the randomness of both $\mathcal{V}$ and $\mathcal{P}$ in both phases our protocol.

$Pr\{\mathcal{V}\text{ accepts}\} > \kappa^*$ implies that both linearity tests and the divisibility tests pass with probability greater than κ^*. Then, we know the linearity tests pass with probability greater than $(1 - 3\delta + 6\delta^2)^{\rho_{lin}}$. If the linearity tests pass with probability greater than $(1 - 3\delta + 6\delta^2)^{\rho_{lin}}$, then the proof is δ-close to linear; this follows from results of Bellare et al. [59, 65]; see the analysis in the extended version of [8], Apdx. A.2. Suppose $Pr\{\mathcal{V}\text{ accepts}\} > \kappa^*$, then we also have $Pr\{\mathcal{V}\text{ accepts}\} > 6M\delta + \kappa_0^M$. If we exclude the cases that any of the queries in the divisibility tests "hit" the non-linear part, the remaining cases are those that all the queries in the divisibility tests "hit" the linear part of the purported proofs. Let E_H be the event that all the queries in the divisibility tests "hit" the linear part of the purported proofs. Since one query in the divisibility tests "hit" the non-linear part is δ, by union bound, $Pr\{\overline{E_H}\} \leq 6M\delta$. Thus, $Pr\{\mathcal{V}\text{ accepts}, \overline{E_H}\} < Pr\{\overline{E_H}\} < 6M\delta$. Since $Pr\{\mathcal{V}\text{ accepts}\} = Pr\{\mathcal{V}\text{ accepts}, E_H\} + Pr\{\mathcal{V}\text{ accepts}, \overline{E_H}\}$, and $Pr\{\mathcal{V}\text{ accepts}\} > 6M\delta + \kappa_0^M$, we can have $Pr\{\mathcal{V}\text{ accepts}, E_H\} > \kappa_0^M$.

Then, we have $Pr\{\mathcal{V}\text{ accepts}|E_H\} > Pr\{\mathcal{V}\text{ accepts}, E_H\} > \kappa_0^M$. Since the effect of testing the divisibility using all the queries that "hit" the linear part of the purported proofs is exactly the same as testing the divisibility under the condition that all purported proofs are linear, we will have :

$$Pr\{\mathcal{V}\text{ accepts}|\text{all purported proofs are linear}\} > \kappa_0^M \qquad (6.5.12)$$

(6.5.12) implies the purported results are correct. Otherwise, if the purported results are not correct, by Lemma 22, we will have $Pr\{\mathcal{V}\text{ accepts}|\text{all purported proofs are linear}\} < \kappa_0^M$, which contradicts (6.5.12). □

6.5.3 Proof of Completeness

The proof of Theorem 19 is as follows:

Proof. If $\mathcal{P}$ computes the result correctly, then he is able to construct the correct proofs: π_{G_j} $(j = 1, 2, \cdots, M)$, $\pi_{\mathtt{IN}_i}$ $(i = 1, 2, \cdots, M)$, π_{H_j} $(j = 1, 2, \cdots, M)$. These proofs will pass the linearity tests and the commitment tests.

Since the results are correct, by Corollary 18, there exist polynomials $H_j(t)$ for $j = 1, 2, \cdots, M$ such that

$$\begin{aligned}&D(t) \cdot H_j(t)\\ =&\psi_j^{(2)}\left(\sum_{k=2}^{K} \mathtt{Z}_{1,k} \cdot A_k(t) + \mathtt{IN}_1^{(1)} \cdot A_1(t), \cdots, \sum_{k=2}^{K} \mathtt{Z}_{M,k} \cdot A_k(t) + \mathtt{IN}_M^{(1)} \cdot A_1(t), \sum_{k=1}^{K} \mathtt{EX}_{[(k-1)\cdot Q]+1}^{(1)} \cdot A_k(t),\right.\\ &\left.\cdots, \sum_{k=1}^{K} \mathtt{EX}_{[(k-1)\cdot Q]+Q}^{(1)} \cdot A_k(t)\right) - \left(\sum_{k=1}^{K-1} \mathtt{Z}_{j,k+1} \cdot A_k(t) + A_K(t) \cdot \mathtt{OUT}_j^{(1)}\right). \qquad (6.5.13)\end{aligned}$$

where $\mathtt{IN}_1^{(1)}, \cdots, \mathtt{IN}_M^{(1)}$ and $\mathtt{EX}_1^{(1)}, \cdots, \mathtt{EX}_{Q\cdot K}^{(1)}$ are respectively the inputs are the extra inputs to the loop circuit, and $\mathtt{OUT}_1^{(1)}, \cdots, \mathtt{OUT}_M^{(1)}$ are the outputs from the loop circuit, as in Figure 6.5. (6.5.13) holds for any $\tau \in \mathbb{F}$, namely

$$\begin{aligned}&D(\tau) \cdot H_j(\tau)\\ =&\psi_j^{(2)}\left(\sum_{k=2}^{K} \mathtt{Z}_{1,k} \cdot A_k(\tau) + \mathtt{IN}_1^{(1)} \cdot A_1(\tau), \cdots, \sum_{k=2}^{K} \mathtt{Z}_{M,k} \cdot A_k(\tau) + \mathtt{IN}_M^{(1)} \cdot A_1(\tau), \sum_{k=1}^{K} \mathtt{EX}_{[(k-1)\cdot Q]+1}^{(1)} \cdot A_k(\tau),\right.\\ &\left.\cdots, \sum_{k=1}^{K} \mathtt{EX}_{[(k-1)\cdot Q]+Q}^{(1)} \cdot A_k(\tau)\right) - \left(\sum_{k=1}^{K-1} \mathtt{Z}_{j,k+1} \cdot A_k(\tau) + A_K(\tau) \cdot \mathtt{OUT}_j^{(1)}\right). \qquad (6.5.14)\end{aligned}$$

By Lemma 21, following holds:

$$\mathtt{OUT}_j^{(2)} = \psi_j^{(2)}(\mathtt{IN}_1^{(2)}, \cdots, \mathtt{IN}_M^{(2)}, \mathtt{EX}_1^{(2)}, \cdots, \mathtt{EX}_Q^{(2)}) \qquad (6.5.15)$$

Then, $\mathcal{P}$ can always convince $\mathcal{V}$ (6.3.11) holds. Hence, $Pr\{\mathcal{V} \text{ accepts}\} = 1$. □

6.6 Cost Analysis

6.6.1 Amortized Cost Calculation

The amortized part includes:

1. constructing the BPs based on the “loop” circuit;
2. generating the commitment queries;
3. generating the linearity test queries;
4. generating the divisibility test queries.
5. generating the decommitment queries;

We will analyze these investments one by one.

For the first part, once the BPs are constructed, they can work over all instances of the same “loop” circuit. Thus, it is a constant cost.

For the second part, $\mathcal{V}$ needs to generate the queries for the commitment phase of the commit/decommit protocol, and send these to $\mathcal{P}$. These queries are in the form of a vector $Enc(r)$ where r is $\mathcal{V}$'s secret. $\mathcal{V}$ only needs to generate one commitment query for all layers and all instances. This query has length equal to the length of the longest linear proof. Recall that the longest linear proof is $\pi_{H_j}(t)$ whose length is $K \cdot D - K$, where K is the number of executions of the loop body, D is the degree of the polynomial that the loop body computes. Thus, the cost of generating the commitment query is $[K \cdot D - K] \cdot \texttt{Enc}$ where $\texttt{Enc}$ is the cost of one encryption over the finite field $\mathbb{F}$. Similarly, if we generate the commitment query for π_G, and π_{IN}, we will introduce extra cost $K \cdot \texttt{Enc}$.

For the third part, $\mathcal{V}$ generates the queries for checking the linearity of each proofs. (Recall that in one linearity test, $\mathcal{V}$ sends out queries q_1, q_2 and $q_1 + q_2$ and expects $\mathcal{P}$ to return answers $\pi(q_1)$, $\pi(q_2)$ and $\pi(q_1 + q_2)$ such that $\pi(q_1) + \pi(q_2) = \pi(q_1 + q_2)$.)

Suppose for each proof, $\mathcal{V}$ needs to perform ρ_{lin} linearity tests per iteration of the verification. (One verification instance we run ρ iteration.) Then, $\mathcal{V}$ needs to randomly generate $2\rho_{lin}$ queries and perform ρ_{lin} times vector addition. As in Zaatar [10], we omit the addition part. We use only one set of queries to check the linearity of all proofs. The cost is $2\rho_{lin} \cdot \texttt{Rand} \cdot [K \cdot D - K]$ where $\texttt{Rand}$ is the cost of randomly generating a number over the finite field $\mathbb{F}$. Similarly, if we generate the linearity test queries for π_G, and π_{IN}, we introduce extra cost $2\rho_{lin} \cdot \texttt{Rand} \cdot K$.

For the fourth part, $\mathcal{V}$ needs to generate the queries q_H, q_G, and q_{IN}. In every layer's verification, $\mathcal{V}$ does not need to send all these queries to $\mathcal{P}$. Instead, he can just send τ to $\mathcal{P}$ and $\mathcal{P}$ calculates these queries himself. The reason why $\mathcal{V}$ needs to construct the queries q_H, q_G, q_{IN}, is that he needs to construct the decommitment queries based on these queries. We use the same q_H for all π_{H_j}, the same q_G for all π_{G_j} (where $j = 1, 2, \cdots, M$), the same q_{IN} for all $\pi_{\texttt{IN}_i}$ (where $i = 1, 2, \cdots, M$). The cost of generating q_H is $[K \cdot D - K] \cdot \texttt{Mult}$, where $\texttt{Mult}$ is the cost of multiplication over the field $\mathbb{F}$. To generate q_G, and q_{IN}, it suffices to generate $(A_1(\tau), A_2(\tau), \cdots, A_K(\tau))$. The cost of generating $(A_1(\tau), A_2(\tau), \cdots, A_K(\tau))$ is $(\texttt{Div} + 4\texttt{Mult}) \cdot K$, according to [10], where $\texttt{Div}$ is the cost of division over the field $\mathbb{F}$.

For the fifth part, in the decommitment phase of the commit/decommit protocol, $\mathcal{V}$ sends $\mathcal{P}$ an auxiliary query which is the weighted sum of the set of PCP queries (including the divisibility queries and the linearity test queries) and the commitment query. Remember that for one verification instance we run ρ iterations, thus, the cost is

$$\begin{aligned} \rho \cdot [2\rho_{lin}(K \cdot D - K) \cdot \texttt{Mult} + 2\rho_{lin} \cdot K \cdot \texttt{Mult} \\ +(K \cdot D - K) \cdot \texttt{Mult} + K \cdot \texttt{Mult}]. \end{aligned} \tag{6.6.1}$$

Now, we omit the cost of constructing BPs. Since for one verification instance we run ρ iterations, the total amortized cost is the sum of the cost from 2) to 5):

$$\begin{aligned} C_a =& [K \cdot D - K] \cdot \texttt{Enc} + K \cdot \texttt{Enc} \\ &+ \rho \cdot [2\rho_{lin} \cdot \texttt{Rand} \cdot (K \cdot D - K) + 2\rho_{lin} \cdot \texttt{Rand} \cdot K] \\ &+ \rho \cdot [(K \cdot D - K) \cdot \texttt{Mult} + (\texttt{Div} + 4\texttt{Mult}) \cdot K] \\ &+ \rho \cdot [2\rho_{lin} \cdot (K \cdot D - K) \cdot \texttt{Mult} + 2\rho_{lin} \cdot K \cdot \texttt{Mult} \\ &+ (K \cdot D - K) \cdot \texttt{Mult} + K \cdot \texttt{Mult}]. \end{aligned} \tag{6.6.2}$$

We simplify the above into:

$$\begin{aligned} C_a =& (K \cdot D) \cdot [\texttt{Enc} + 2\rho \cdot \rho_{lin} \cdot (\texttt{Rand} + \texttt{Mult}) + 2\rho \cdot \texttt{Mult}] \\ &+ \rho \cdot (\texttt{Div} + 3 \cdot \texttt{Mult}) \cdot K \end{aligned} \tag{6.6.3}$$

6.6.2 Non-amortized Cost

The non-amortized cost consists of two parts: one is the cost in each layer, including the check and the updating; the other is the decommitment test.

Firstly, to check

$$\texttt{OUT}_j^{(2)} = D(\tau) \cdot \pi_{H_j}(q_H) + \pi_{G_j}(q_G) + A_K(\tau) \cdot \texttt{OUT}_j^{(1)},$$

$\mathcal{V}$ needs to perform $2M$ multiplications. Secondly, to update $\texttt{IN}_i$, $\mathcal{V}$ needs to perform another M multiplications. Thirdly, $\mathcal{V}$ needs to generate $\texttt{EX}_u$'s. Since the extra inputs are changing for every computation instance, this part can not be amortized. This cost is $Q \cdot K \cdot \texttt{Mult}$. Lastly, in the final decommitment test, $\mathcal{V}$ needs to perform the decommitment test for $3M$ proofs: q_{H_j}, $q_{\texttt{IN}_i}$ and q_{G_j} where $i, j = 1, 2, \cdots, M$. For each decommitment test, $\mathcal{V}$ needs to perform one decryption, one multiplication for the divisibility test query, one multiplication for each of the linearity test queries. This implies a cost of $3M \cdot (\texttt{Dec} + \texttt{Mult}(1 + 3\rho_{lin}))$, where $\texttt{Dec}$ is the cost for one decryption operation over field $\mathbb{F}$. Since for one verification instance we run ρ iterations, the total non-amortized cost is

$$3M \cdot \texttt{Dec} + \rho(6M + Q \cdot K + 9M\rho_{lin}) \cdot \texttt{Mult}. \qquad (6.6.4)$$

6.6.3 Prover's Cost

The method to construct the proof vector is the same as that in Zaatar. However, in our design, the proof length is reduced from $K \cdot C$ to $K \cdot D$. This implies the corresponding costs in Figure 6.6.

6.7 Conclusions

This chapter addresses two fundamental problems in the verifiable computation: 1) whether and what computations can have lower amortized cost and proof generation cost; 2) how to handle loops concisely in verifiable computing. We give a first-step answer by showing that for computation with loops, we can use Block Programs, our new theoretical result, to reduce the verifier's amortized cost to the sum of two parts, one of which is merely verification of one execution of the loop body (which does not scale with the number of loop repetitions) and the other is linear in the *degree* of the loop body and the number of executions of the loop body. From the theory of arithmetic circuit complexity, the degree of most circuits will be far less than their size. Hence the verifier's amortized costs in our design is far less than the counterpart of existing algorithms, which are linear in the *size* of the whole circuit. We also improve the prover's cost of proof generation from quasilinear in the size of the loop body (like in Zaatar) to quasilinear in the degree of the loop body, attached with a cost of generating a proof for one single execution of the loop body, achieving great savings similarly.

For applications that require a large number of loop executions (which is common in not only compute-intensive computations but also data-intensive computations such as big data applications), and have loop bodies the degree of which is far less than their size (from the theory of arithmetic circuit complexity, this is nature: the degree of *most* circuits will be far less than their size), our approach is expected to perform better than existing verifiable computation protocols. However, for “deep” loop bodies, in which the output of each gate is an input to the next gate, standard algorithms like Zaatar and Pinocchio would probably do better.

CHAPTER 7. FUTURE WORK: DISTRIBUTED VERIFIABLE COMPUTATION WITH VERIFICATION OUTSOURCING

7.1 Motivation and Problem Statement

Cloud computing is bound to become the leading trend of modern computing. Its potential client base is extremely diverse, ranging from small businesses, trying to cut down on their computation and storage costs, to private users trying to run computation-intensive applications on their lightweight, hand-held devices, and to the military, trying to opportunistically employ both trusted and untrustworthy computational resources for quick information processing, leading to responsible decision making. In the *delegation of computation* (or *computation outsourcing*) paradigm, the client delegates a computational task to the server. The client provides the server with the input of the computational task. The server produces a result, and returns it to the client. Should the client require result assurance, he can start a standard verification protocol, where the server and the client assume the roles of the *prover* and the *verifier*, respectively. The problem of *delegation of computation* or *verifiable computation* has been intensely investigated, and almost-practical verification algorithms have been recently proposed not only in this dissertation but also in existing work. Moreover, the idea of *verification outsourcing*, first introduced in this dissertation, is bound to additionally decrease the computational costs of the client, by outsourcing verification to a third party. Of course, new soundness and confidentiality issues arise in this context.

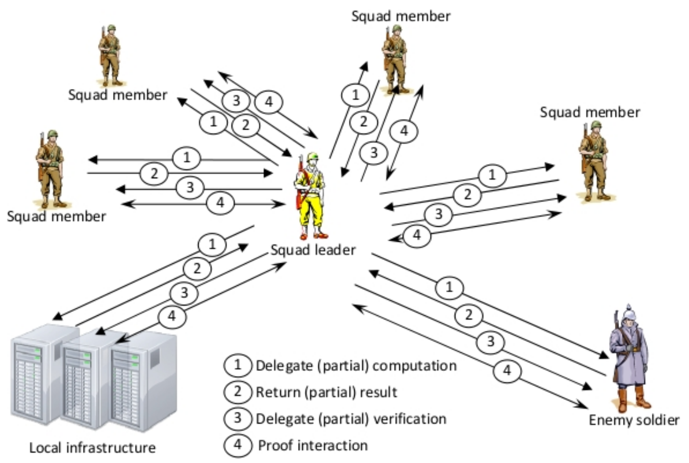


Figure 7.1 A basic scenario of distributed delegation of verification with distributed delegation of verification.

Yet another, more challenging application of cloud computing emerges in distributed environments, and is the focus of the current research project. While civilian applications abound, for demonstration purposes we like to refer to a more security-sensitive example, like the one presented by a combat environment.

Imagine a soldier squad, or a squadron of aircraft deployed in the field, acquiring, sharing and processing information to support decision making. Since information and (as a consequence) computational loads may easily become overwhelming (especially for a single one of the lightweight devices carried by infantry troops), the need for distributed delegation of computation becomes apparent. In this scenario, the squad (or squadron) leader plays the role of the *client*, while all other available computation resources play the role of the *server*. The *available computation resources* include, but are not limited to, the computing devices carried by the squad (or squadron) members. However, in addition to these, the local infrastructure may be used to help the computation. This approach immediately implies a need for confidentiality. Moreover, since all protocols are negotiated over a wireless medium, the confidentiality constraints on the delegation of computation would also protect against undetected intrusion by malicious devices.

A basic scenario is depicted in Figure 7.1, where squad members, the local infrastructure, and enemy soldiers are all part of two protocols: delegation of computation, and delegation of verification. The local infrastructure may prove to be a computational asset, but may not be trusted with sensitive information, while the enemy soldier will probably attempt to interfere with the correctness of the protocol, or intelligently influence the protocol in a way that leads to maximum information leakage. Consequently security mechanisms have to be implemented, to restrict the amount of information that leaks to the delegates about: (a) the details of the computational task, (b) the input of the computational task and (c) the result of the computational task. In addition, since verification is also outsourced in a distributed manner, an additional layer of security mechanisms should ensure that the verification protocol maintains its soundness, even in the presence of colluding cheating or lazy verifiers.

7.2 Methods and the Key Contribution

In the future work, we aim to combine the distributed delegation of computation with confidentiality constraints, and the distributed delegation of verification, such that the set of provers coincides with the set of verifiers. In fact this new framework implies only two types of actors: the client $\mathcal{C}$ and the multiple prover/verifiers $\mathcal{P}/\mathcal{V}$ (as in Figure 7.2). We aim at keeping the computation and/or the inputs/outputs (at least partially) confidential from the prover/verifiers. Intelligent mixing and scheduling of the delegation and verification tasks is required to maintain the soundness of the protocol. Intelligent mixing and scheduling would decrease the correlation between the computation and verification tasks at each prover/verifier. This would reduce the prover/verifier's ability to cheat during the computation and verification processes, while ensuring that the prover/verifier cannot recover too much information about the computation task, its inputs and its outputs. In essence, each computation/verification assignment is masked by other computation/verification assignments.

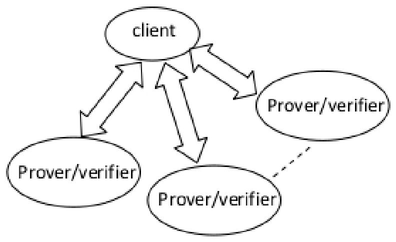


Figure 7.2 Delegation of computation with delegation of verification in a simple star topology.

The challenge is to ensure that such exposure, which is beyond the control of the client, does not compromise the confidentiality and soundness of the protocol. Information leakage may prove to be inevitable under particular network conditions. For these cases, we design a mechanism that can easily detect and isolate the point where such information leakage occurs. The following cases shall be considered under this topology: (1) the client has the full knowledge of the topology of the network of all verifiers/provers; (2) the client has partial or no knowledge of the topology.

The first case requires a deeper understanding of the network topology. We shall decompose the network according to graph theory and find out critical nodes along the verification/computation chain. Critical nodes are those that are exposed to the most information about the computational task and its inputs/outputs. Special algorithms will be required when interacting with the critical nodes. For the second case, a tradeoff between security constraints and efficiency will be provided.

To summarize, the novelty of our future work can be stated as follows:

1. We propose to augment the distributed-delegation-of-computation paradigm with distributed delegation of verification. While the confidentiality aspect of delegated verification can be solved by an extension of the results of verification outsourcing, we will introduce more efficient algorithms, that rely on intelligently mixing the computation results and verification tasks during the verification-delegation process.

2. We will demonstrate, for the first time, that in a distributed environment, the same computational resources used for performing the delegated computation can also be used for performing the verification, in a secure and sound manner. We will focus on a randomly-connected topology with multiple, collaborating delegation and aggregation nodes.

BIBLIOGRAPHY

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, Feb. 1989. 1, 9

[2] S. Arora and S. Safra, "Probabilistic checking of proofs; a new characterization of np," in *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, ser. SFCS '92, Washington, DC, USA, 1992, pp. 2–13. 1, 9, 12

[3] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, ser. STOC '91, New York, NY, USA, 1991, pp. 21–32. 1, 9, 12

[4] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 156–189, 1988. 1, 13

[5] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, ser. STOC '92, New York, NY, USA, pp. 723–732. 1, 9, 12

[6] ——, "Improved efficient arguments (preliminary version)," in *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '95. London, UK, UK: Springer-Verlag, 1995, pp. 311–324. 1, 9, 12

[7] Y. Ishai, E. Kushilevitz, and R. Ostrovsky, "Efficient arguments without short pcps," in *Proceedings of the Twenty-Second Annual IEEE Conference on Computa-*

tional Complexity, ser. CCC '07, Washington, DC, USA, 2007, pp. 278–291. vii, 1, 9, 14, 15, 16, 18, 25, 26, 27, 35, 41, 49, 50, 54

[8] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *NDSS*, 2012. 1, 9, 17, 18, 19, 25, 28, 32, 36, 43, 49, 50, 80, 82, 83, 95

[9] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *USENIX Security*, 2012. vii, 1, 4, 9, 18, 19, 21, 22, 25, 26, 27, 35, 49, 50, 52, 68, 74, 80

[10] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," ser. EuroSys '13, 2013. 1, 2, 9, 17, 49, 50, 52, 54, 62, 64, 68, 74, 80, 82, 83, 84, 87, 94, 98

[11] B. Braun, A. J. Feldman, Z. A. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *SOSP*, 2013. 1

[12] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish, "A hybrid architecture for interactive verifiable computation," in *the IEEE Symposium on Security and Privacy*, ser. IEEE S&P 13, 2013. 1, 4, 10

[13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Proceedings of the IACR Eurocrypt Conference*, ser. Eurocrypt 13, 2013. 1, 2, 10, 20, 21, 49, 69

[14] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: nearly practical verifiable computation," in *the IEEE Symposium on Security and Privacy*, ser. IEEE S&P 13, 2013. 1, 2, 9, 20, 21, 49, 52, 68, 74, 82, 84, 87

[15] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *Proceedings of*

the 33rd Annual International Cryptology Conference, ser. CRYPTO '13, 2013, pp. 90–108. 1, 7, 9, 10

[16] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 781–796. 4, 7, 10, 11

[17] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–178. 6, 8, 13

[18] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 445–454. 6, 24

[19] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, "Multi-prover interactive proofs: how to remove intractability assumptions," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ser. STOC '88. New York, NY, USA: ACM, 1988, pp. 113–131. 6, 9, 12, 24

[20] U. Feige and J. Kilian, "Making games short (extended abstract)," in *STOC*, 1997, pp. 506–516. 6, 24

[21] R. Canetti, B. Riva, and G. N. Rothblum, "Two 1-round protocols for delegation of computation," Cryptology ePrint Archive, Report 2011/518, 2011, http://eprint.iacr.org/. 6, 10, 24

[22] L. Babai, L. Fortnow, and C. Lund, "Nondeterministic exponential time has two-prover interactive protocols," in *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, ser. SFCS '90. Washington, DC, USA: IEEE Computer Society, 1990, pp. 16–25 vol.1. 12

[23] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, “Proof verification and the hardness of approximation problems,” *J. ACM*, vol. 45, no. 3, pp. 501–555, May 1998. 9, 12, 14

[24] I. Dinur, “The pcp theorem by gap amplification,” *J. ACM*, vol. 54, no. 3, June 2007. 12

[25] M. Blum and S. Kannan, “Designing programs that check their work,” *J. ACM*, vol. 42, no. 1, pp. 269–291, Jan. 1995. 9, 12

[26] S. Micali, “Computationally sound proofs,” *SIAM J. Comput.*, vol. 30, no. 4, pp. 1253–1298, Oct. 2000. 9, 12, 14

[27] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, “Short pcps verifiable in polylogarithmic time,” in *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, ser. CCC ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 120–134. 9, 12

[28] O. Meir, “Combinatorial pcps with efficient verifiers,” in *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 463–471. 9, 12

[29] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan, “Robust pcps of proximity, shorter pcps and applications to coding,” in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, ser. STOC ’04. New York, NY, USA: ACM, 2004, pp. 1–10. 9, 12

[30] E. Ben-Sasson and M. Sudan, “Short pcps with polylog query complexity,” *SIAM J. Comput.*, vol. 38, no. 2, pp. 551–607, May 2008. 9, 12

[31] A. Polishchuk and D. A. Spielman, "Nearly-linear size holographic proofs," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, ser. STOC '94. New York, NY, USA: ACM, 1994, pp. 194–203. 9, 12

[32] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra, "Pcp characterizations of np: Towards a polynomially-small error-probability," in *In Proc. 31st ACM Symp. on Theory of Computing*, 1999, pp. 29–40. 9, 12

[33] D. Moshkovitz and R. Raz, "Two-query pcp with subconstant error," *J. ACM*, vol. 57, no. 5, pp. 29:1–29:29, June 2008. 9, 12

[34] I. Dinur and P. Harsha, "Composition of low-error 2-query pcps using decodable pcps," in *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, 2009. 9, 12

[35] I. Dinur and O. Meir, "Derandomized parallel repetition of structured pcps," in *IEEE Conference on Computational Complexity*. IEEE Computer Society, 2010. 9, 12

[36] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, ser. STOC '82. New York, NY, USA: ACM, 1982, pp. 365–377. [Online]. Available: http://doi.acm.org/10.1145/800070.802212 13

[37] S. Setty, V. Vu, N. Panpalia, B. Braun, M. Ali, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality (extended version)," Cryptology ePrint Archive, Report 2012/598, 2012, http://eprint.iacr.org/. 17, 64

[38] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Proceedings of the 10th The-*

ory of Cryptography Conference on Theory of Cryptography, ser. TCC'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 315–333. 21

[39] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proceedings of the 31st annual conference on Advances in cryptology*, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 111–131. 9

[40] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, ser. EUROCRYPT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 149–168. 9

[41] F. Ergun and S. R. Kumar, "Approximate checking of polynomials and functional equations," in *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, ser. FOCS '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 592–. 9

[42] P. Golle and I. Mironov, "Uncheatable distributed computations," in *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, ser. CT-RSA 2001. London, UK, UK: Springer-Verlag, 2001, pp. 425–440. 9

[43] G. O. Karame, M. Strasser, and S. Čapkun, "Secure remote execution of sequential computations," in *Proceedings of the 11th international conference on Information and Communications Security*, ser. ICICS'09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 181–197. 9

[44] R. Sion, "Query execution assurance for outsourced databases," in *Proceedings of the 31st international conference on Very large data bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 601–612. 9

[45] B. Thompson, S. Haber, W. G. Horne, T. Sander, and D. Yao, “Privacy-preserving computation and verification of aggregate queries on outsourced databases,” in *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, ser. PETS ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 185–201. 9

[46] C. Wang, K. Ren, and J. Wang, “Secure and practical outsourcing of linear programming in cloud computing.” in *INFOCOM*. IEEE, 2011, pp. 820–828. 9

[47] C. Wang, K. Ren, J. Wang, and K. M. R. Urs, “Harnessing the cloud for securely solving large-scale systems of linear equations,” in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 549–558. 9

[48] M. J. Atallah and K. B. Frikken, “Securely outsourcing linear algebra computations,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS ’10. New York, NY, USA: ACM, 2010, pp. 48–59. 9

[49] M. Garofalakis, “Proof sketches: Verifiable in-network aggregation,” in *In IEEE Internation Conference on Data Engineering (ICDE)*, 2007. 9

[50] B. Przydatek, D. Song, and A. Perrig, “Sia: secure information aggregation in sensor networks,” in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys ’03. New York, NY, USA: ACM, 2003, pp. 255–265. 9

[51] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer, “Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract,” in *ITCS*, 2013, pp. 401–414. 10

[52] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, “Delegating computation: interactive proofs for muggles,” in *Proceedings of the 40th annual ACM symposium*

on Theory of computing, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 113–122. 10

[53] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: ACM, 2012, pp. 90–112. 10

[54] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *CRYPTO (2)*, 2013, pp. 71–89. 10

[55] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer, "On the concrete-efficiency threshold of probabilistically-checkable proofs." *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 19, p. 45, 2012. 10

[56] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them: from theoretical possibility to near-practicality," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 20, p. 165, 2013. 11

[57] O. Goldreich, *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004, p. 381. 33, 39, 41, 44

[58] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *Proceedings of the 30th annual conference on Advances in cryptology*, ser. CRYPTO'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 465–482. 37, 84

[59] M. Bellare, M. Kiwi, and M. Sudan, "Linearity testing in characteristic two," *IEEE Transactions on Information Theory*, 1996. 35, 95

[60] S. Setty, A. J. Blumberg, and M. Walfish, "Toward practical and unconditional verification of remote computations," in *Proceedings of the 13th USENIX conference*

on Hot topics in operating systems, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011, pp. 29–29. 49, 50

[61] D. E. Knuth, *Seminumerical Algorithms, the art of computer programming*, 3rd ed. Addison-Wesley, 2007. 61

[62] M. Karchmer and A. Wigderson, "On span programs," in *In Proc. of the 8th IEEE Structure in Complexity Theory*. IEEE Computer Society Press, 1993, pp. 102–111. 69

[63] X. Chen, N. Kayal, and A. Wigderson, *Partial Derivatives in Arithmetic Complexity and Beyond (Foundations and Trends(R) in Theoretical Computer Science)*. Hanover, MA, USA: Now Publishers Inc., 2011. 87

[64] A. Shpilka and A. Yehudayoff, "Arithmetic circuits: A survey of recent results and open questions," *Found. Trends Theor. Comput. Sci.*, vol. 5, pp. 207–388, Mar. 2010. 87

[65] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, "Efficient probabilistically checkable proofs and applications to approximations," in *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '93. New York, NY, USA: ACM, 1993, pp. 294–304. 95