

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

<http://wrap.warwick.ac.uk/153856>

Copyright and reuse:

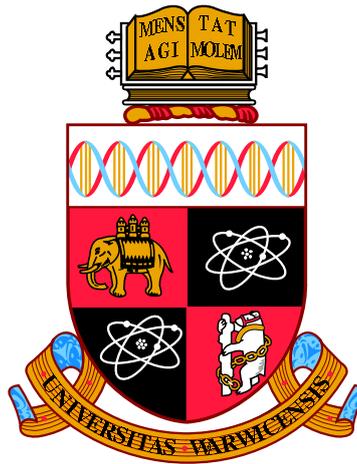
This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it.

Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk



Streaming Interactive Proofs

by

Christopher Hickey

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Department of Computer Science

Mar 2021

Contents

Acknowledgments	iv
Declarations	v
Abstract	vi
Chapter 1 Introduction	1
1.1 The Setting	1
1.2 Core Definitions and Intuition	2
1.3 Our Results and Roadmap	3
Chapter 2 Proof Systems	4
2.1 Building the Notion of an Interactive Proof System	4
2.1.1 Languages and Decision Problems	4
2.1.2 Introducing Proofs	5
2.1.3 The Power of Interaction?	5
2.1.4 Interactivity and Randomness: IP	6
2.2 IP = PSPACE: A Brief History of IPs	7
2.3 Interactive Proofs in the Streaming Setting	10
2.3.1 Languages, Functions and Notation	10
2.3.2 Space Bounds and Streams	10
2.3.3 Annotated Data Stream Model (Non-interactive SIPs)	12
2.4 Costs in Streaming Interactive Proofs	13
2.5 SIP Toolbox	14
2.5.1 Fingerprints	15
2.5.2 Low Degree Extensions	15
2.5.3 Sum-Check Protocol	16
2.6 SIPs and Complexity Classes	18
2.6.1 Annotations and Arthur-Merlin Games	18

2.6.2	Tricks from IPs for SIPs	19
2.6.3	What can be done with a streaming interactive proof?	20
Chapter 3 Non-Interactive Proofs for Data Analysis		22
3.1	Motivation	22
3.1.1	Chapter Outline	24
3.1.2	Related Work	24
3.2	Linear Algebraic Checks	25
3.2.1	Fingerprinting the Gramian Matrix	25
3.2.2	Matrix Multiplication	27
3.2.3	Eigenvalue Check	30
3.2.4	Matrix Inversion	36
3.2.5	Cholesky Decomposition	38
3.2.6	Symmetric Generalised Eigenvalues	38
3.3	Statistical Analysis	42
3.3.1	Ordinary Least Squares	42
3.3.2	Principal Component Analysis	43
3.3.3	Fisher Linear Discriminant Analysis	46
3.4	Practical Results	50
Chapter 4 Achieving Optimal Costs through Interactivity		52
4.1	Motivation	52
4.1.1	Quantifying Costs	53
4.1.2	Prior Work	54
4.1.3	Contributions and outline	55
4.2	How Much Interaction Do We Want?	56
4.3	Protocols for Linear Algebra Primitives	57
4.3.1	Inner Product	58
4.3.2	Matrix Multiplication	64
4.3.3	Vector-Matrix-Vector Multiplication	66
4.4	Practical Analysis	67
4.4.1	Setup	68
4.4.2	Matrix Multiplication Results	69
4.5	Concluding Remarks	71
Chapter 5 Space-Bounded Commitment Schemes and Spatial Zero Knowledge		72
5.1	Introduction	72
5.2	Commitment Schemes	73

5.2.1	Space-Bounded Commitment Schemes	75
5.2.2	Aside: Negligible functions in the Space-Bounded setting	76
5.3	Motivation: Spatial Zero Knowledge	77
5.3.1	Zero Knowledge	77
5.3.2	Spatial Zero Knowledge	80
5.3.3	Why statistically indistinguishable?	81
5.3.4	Achieving Spatial Zero Knowledge Through Linear Space-Bounded Commitments	81
5.4	Preliminaries for Spatial Zero Knowledge	82
5.4.1	Polynomial Evaluation Protocol (PEP) [Chakrabarti et al., 2013] . .	82
5.4.2	SIPs for INDEX	84
5.5	A Spatially Hiding Commitment Scheme	85
5.5.1	The Average Case Hardness of INDEX	85
5.5.2	The Commitment Scheme	87
5.5.3	Warm-up: Zero Knowledge with a $O(\sqrt{n})$ Verifier	89
5.5.4	Making the Scheme Linear	92
5.6	Efficient Spatial Zero Knowledge INDEX	96
5.7	Spatial Zero Knowledge Sum-Check	98
5.8	Future Work	102
Chapter 6 Conclusion		104
6.1	Discussion	104
6.2	Extensions and Future Work	105
6.3	Closing Remarks	105

Acknowledgments

Without a doubt, the foremost thanks has to be to my supervisor, Graham Cormode. Week after week he's always been there to answer my questions and give valuable advice, without which this thesis would have been impossible. A further thanks must go to Tom Gur as well, for eighteen months of weekly meetings to try and understand streaming zero knowledge. Naturally, I would also like to thank my advisors Dmitry Chistikov and Victor Sanchez for their guidance in my yearly reviews.

Outside of the academic help, the many hours of work on the PhD were only really manageable with the many other hours of not-work. A thousand thanks to Dan, Ollie, Nandini, James, Kayleigh, Toby, Charlie, Charles and all the friends who aided me through the countless hours of messing around on and around campus. Beyond this, I'd like to thank the above, and all those involved in the infinite shenanigans of the Game of Thrones Society, and each member's willingness to go along with whatever nonsense was unfolding at each event. It goes without saying that my time at the helm of the society was undoubtedly the highlight of my Warwick experience.

Beyond the bubble of Warwick, I'd like to thank my friends in Manchester, who I could always count on for fun and games on my occasional forages up north. I'd especially like to do an additional thanks to Dan, Louis, Nathan, Timmy and Poppy, the housemates who kept me somewhat sane during my write-up in the midst of the ongoing Covid-19 lockdown.

Finally, of course, I would like to express my gratitude to my family for their continued support and advice through my shifting ambitions leading to this point. Individual thanks goes to my dad, for always trying to understand what I am getting up to in my research; and my mum, for her updates on Gordon the wood pigeon.

Declarations

This thesis is presented in accordance with the regulations for the degree of Doctor of Philosophy. It has been composed by the author and has not been submitted in any previous application for any degree. The work described in this thesis has been undertaken by the author except where otherwise stated.

Much of the work presented in this thesis has been published in peer reviewed conferences, workshops and journals. Publication highlights include the following:

Chapter 3 is formed entirely from the work of Graham Cormode and Chris Hickey: “Cheap Checking for Cloud Computing: Statistical Analysis via Annotated Data Streams.” from the 2018 International Conference on Artificial Intelligence and Statistics (AISTATS).

Chapter 4 is formed entirely from the work of Graham Cormode and Chris Hickey: “Efficient Interactive Proofs for Linear Algebra.” from the 30th International Symposium on Algorithms and Computation (ISAAC 2019).

Chapter 5 is formed entirely from the as yet unpublished work of Graham Cormode, Chris Hickey and Tom Gur between 2019 and 2020.

Abstract

An interactive proof is a conversation between a powerful machine, the ‘prover’, and a ‘verifier’ with low resources. The aim of the conversation is for the prover to convince the verifier about the output of a function (that is computationally or space intensive to evaluate) over some shared data set. The concept of streaming interactive proofs (SIPs) considers a verifier with very small space, who streams the shared data, and then engages in an interactive protocol with the prover.

Our work begins by looking to improve protocols for the practical verification of outsourced data analysis. We explore non-interactive and multi-round protocols for vector and matrix multiplications and analyse the real-world practicality of these approaches. We demonstrate how these protocols can be used in data analysis, considering the numerical concerns when rounding is required. We investigate the costs for the verifier, while trying to keep the overheads for the prover at a minimum, and discuss bottlenecks.

Finally, we introduce the entirely new concept of Streaming Zero Knowledge for interactive proofs, which is the adaptation of regular SIPs where now the verifier learns *no additional information* about the data set besides the truth of the statement the prover is trying to prove. We show several examples and build up a powerful multipurpose protocol in order to showcase the strengths of this new model.

Chapter 1

Introduction

1.1 The Setting

The work in this thesis is designed to further improve our understanding of the powers of streaming interactive proofs (SIPs) [Cormode et al., 2011]. SIPs are a theoretical notion inspired by the real world setting of cloud computing. We consider a situation in which a client (hereafter referred to as the *verifier*), with very few computational resources but large amounts of data, can use a powerful second party to perform analysis on the data whilst maintaining confidence the analysis is performed correctly. The crux of the setting is that it's much easier to *check* work than it is to *do* work.

Our task is to create protocols that let the verifier outsource work to a powerful helper, such as Amazon Web Services or Microsoft Azure, whilst still having the verifier do a small amount of work in order to check the outsourced work is done correctly. These online cloud services are prone to failures, and relinquish any guarantee of flawless and correct computation in their terms and conditions [AWS, 2020]. The setting of checking cloud computing covers not only the idea of a weak verifier checking work sent to a super computer, but also covers distributed computation, where a comparatively weak server checks the work of a huge network of computers. An example of this would be SETI@home, which used redundancy, i.e., many computers repeating the work of others, to find errors [SETI@home, 1999], but could achieve better guarantees through SIPs.

Our work aims to provide a way for weak verifiers to perform basic analysis on large data streams, initially investigating the setting where the verifier receives a message from powerful second party (hereafter called the *prover*), but cannot communicate further. We then examine the consequences of adding interactivity where the prover and verifier have a short conversation in order to allow the verifier to be convinced using less total communication. Our final technical chapter studies the concept of zero knowledge [Goldwasser et al.,

1985], where the prover attempts to convince the verifier of a fact regarding the data set, but with the added task of making sure no extra information the verifier couldn't compute on its own is released in the messages. The setting of this final exploration is less immediate, it represents a scenario whereby a verifier with little space sees a dataset, and then wants to perform computation on it, but with the data being secure now to further leakage to the verifier, beyond what was already known.

1.2 Core Definitions and Intuition

We will now further detail the concepts within interactive proofs. To begin with our verifier is a probabilistic polynomial-time turing machine (PPT), whilst our prover is unbounded in both space and time. The first question to ask should be 'what does it mean to prove something?'. We want our verifier to be convinced that a solution to a problem is correct, but how should we define correct?

This is where the concept of *completeness* and *soundness* come in. Say our verifier is attempting to compute $f(x)$, where f is a complex function to compute with an input x . Our prover is trying to prove that $y = f(x)$. If it is indeed true that $y = f(x)$ we want our verifier to accept the proof; this is the concept of completeness. If it's false, so $y \neq f(x)$, we want our verifier to reject the proof, even in the presence of a malicious prover, and this is the concept of soundness. If an algorithm for proving a particular statement is both sound and complete, then it is an interactive proof system. However, this definition is incredibly strict, and for the majority of problems, it's near impossible to achieve both completeness, soundness, and efficiency (sublinear proof size and speed). In our work, and much of the prior work, we allow slight leniency to vastly increase the power of the class. We say that an interactive proof system is still considered sound if a verifier rejects false claims more than $\frac{2}{3}$ of the time.

A vital part of interactive proofs is of course the idea of *interactivity*. The main power of the verifier in an interactive proof system versus a non-interactive proof system is that now the verifier can quiz the prover on messages the prover sends. We will see plenty of examples where the fact the verifier has private randomness and can challenge the prover to provide consistent responses allows for short interactive proofs with incredibly high soundness.

Now we can begin to answer the question of what exactly it is interactive proofs can do. It follows immediately with even perfect soundness that any problem in NP has an interactive proof system. The class NP covers all decision problems where, when the answer is 'yes', there is a proof of this verifiable in polynomial time. For example, our helper can prove to a verifier that a graph has a 3-colouring by simply sending the 3-colouring. When

we remove the requirement of perfect soundness, the remarkable result is that we can solve any problem in PSPACE, the set of all problems that can be solved in polynomial space [Shamir, 1992; Lund et al., 1992].

1.3 Our Results and Roadmap

Our results can be split largely into three parts. Firstly, we investigate non-interactive proofs for data analysis (Chapter 3). We build on a prior annotated matrix multiplication protocol, and show the protocol is optimal up to constant factors. We use this to develop annotated streaming protocols for a variety of further problems involving approximations, such as finding eigenvalues and matrix inversion. We demonstrate practical data science primitives and show that these protocols allow the verifier to run significantly faster than solving the problem alone.

In the next chapter (Chapter 4), we delve into the practicality of streaming interactive proofs. We adapt several existing protocols, and develop some new protocols with *variable* interactivity, specifically allowing and discussing the costs incurred when the protocol designer can choose the number of rounds. We then run a series of experiments to test which properties of a SIP should be optimised in order to minimize the time the entire protocol takes.

In the final content chapter (Chapter 5), we discuss the potential for zero knowledge streaming interactive proofs, where the zero knowledge security comes from the verifier's space bound restricting it from learning additional information about a previously seen stream when interacting with the prover. We introduce the concept of spatial commitment schemes, and show how these allow us to create zero knowledge streaming interactive proofs for a large range of common problems.

Chapter 2

Proof Systems

In this chapter we make our previous notions rigorous, and discuss relevant work in the world of streaming interactive proofs, and the world of proof systems in general.

2.1 Building the Notion of an Interactive Proof System

A ‘interactive proof system’ is simply a conversation between two players, one player (the prover) is trying to convince the other (the verifier) that a particular statement is true. In order to build up our intuition, we will slowly work through the concepts involved in an interactive proof to lead to a strong definition allowing us to truly understand their power.

2.1.1 Languages and Decision Problems

First, the idea of proving a statement is true is equivalent to checking that, given some Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, an input $x \in \{0, 1\}^n$ satisfies $f(x) = 1$. This Boolean function f defines a *language* $\mathcal{L}_f = \{x \in \{0, 1\}^n : f(x) = 1\}$. If the verifier is *polynomial time*, it can evaluate Boolean functions where computing $f(x)$ for $x \in \{0, 1\}^n$ requires $O(n^c)$ steps, for some constant $c > 0$. We will write $\text{poly}(n)$ when referring to something that is $O(n^c)$ for some constant $c \in \mathbb{N}$ from here onwards.

A polynomial time verifier doesn’t *need* help in order to solve a language in the complexity class \mathbf{P} , which is the set of all languages that can be ‘decided’ in polynomial time (i.e. it can explicitly compute $f(x)$ to see if $x \in \mathcal{L}$). This complexity class includes a number of common problems such as maximum matchings, calculating the greatest common divisor, even determining if a number is prime [Agrawal et al., 2004]. However, many other useful problems are (likely) not in \mathbf{P} . This is why we introduce the prover, to help the verifier evaluate more complex functions.

It's worth noting that whilst originally interactive proofs were introduced to allow polynomial time verifier's to solve problems in larger complexity classes [Babai, 1985; Goldwasser et al., 1985], this thesis largely investigates the power of interaction in speeding up problems from \mathbf{P} , that would otherwise take a verifier a large amount of time or space. However, we will explore some of the historical basis for interactive proofs as it gives insight into the incredible power of proof systems.

This thesis will be working with *streaming* interactive proof systems (SIPs), where the verifier doesn't have the space to store x , and instead reads it in one pass. Once we have formally defined interactive proofs, we will move onto proofs with a space-bounded verifier.

2.1.2 Introducing Proofs

The complexity class \mathbf{NP} captures the notion of *checking* $x \in \mathcal{L}_f$. It is the set of all languages that can be *verified* in polynomial time. Imagine we want to check $x \in \mathcal{L}_f$, but we can't compute $f(x)$ in polynomial time. If the language \mathcal{L}_f , with corresponding $f : \{0, 1\}^n \rightarrow \{0, 1\}$, is in \mathbf{NP} that means there is a $g : \{0, 1\}^n \times \{0, 1\}^{\text{poly}(n)} \rightarrow \{0, 1\}$ and a 'proof', $w \in \{0, 1\}^{\text{poly}(n)}$ (also known as a 'witness' or 'certificate'), such that

$$g(x, w) = 1$$

and g can be evaluated in polynomial time. In simpler terms, if we want to check $x \in \mathcal{L}$, we can get a more powerful computer to send us w , and we can then use w to confirm $x \in \mathcal{L}$ in polynomial time.

This is our first example of a proof system. However, it's not currently very well defined. What if the prover sends some arbitrary w such that $g(x, w) \neq f(x)$? What if the prover can't find an appropriate w ? We need to capture the real necessities of a proof. Recall that we want a proof to be **complete** and **sound**. Loosely speaking, completeness tells us that a honest prover will indeed be able to send a w such that $g(x, w) = 1$ if $x \in \mathcal{L}$. Soundness tells us that a dishonest prover can *not* find a w such that $g(x, w) = 1$ if $x \notin \mathcal{L}$. If we require that the function g satisfies these conditions, we have achieved a proof system, and can show that the language is in the complexity class \mathbf{NP} .

2.1.3 The Power of Interaction?

Within the notion of a proof system outlined in the previous section, the prover's role is to find a suitable witness, so the verifier can check it without the need for interaction with the prover. \mathbf{NP} covers a lot of problems, however, the next task is to determine whether we can achieve a larger class of problems if we let the verifier interrogate the prover. This was the

question asked in the mid-eighties [Babai, 1985; Goldwasser et al., 1985], and led us in to the world of interactive proof systems.

Unfortunately, it's not as simple as just requiring interaction. In the deterministic setting, we remain in **NP**. The prover will always know what the verifier will ask, given the input, and this will in turn mean that all the interaction can be collapsed into a single message. We need one more powerful tool - randomness.

2.1.4 Interactivity and Randomness: **IP**

So far our verifier has simply been a machine that can run in polynomial time. We can vastly increase the power of the verifier here by giving it access to randomness. This means now the message the verifier sends no longer depends simply on x and the prover's message, but also on r_1 , some random choices of the verifier *unknown to the prover*. These random choices stop the prover from knowing what it's second message will be, and means that the interactivity can't be collapsed to a single message. This is the notion of interactive proofs introduced in the eighties by Goldwasser et al. [1985], and will be the main overarching model we use. The complexity class of problems that can be solved with an interactive proof of k rounds is called **IP(k)**.

The original model had the verifier generating private randomness, however it turns out that the verifier's random choices can be public and are still effectively as powerful, just requiring an additional 2 rounds of interaction (Goldwasser and Sipser [1986]). Although it's worth noting that this statement doesn't hold when we have streaming interactive proofs [Chakrabarti et al., 2013].

It is important to note now we've added randomness, we're no longer in the case where the verifier *always* accepts a proof; we need to prepare for the possibility the verifier could, by chance, reject a correct proof (the *completeness* error), or accept an incorrect proof (the *soundness* error). Previously, when everything was deterministic, we required these errors to be zero, but part of the power of randomness is allowing this probability of failure. We can now define **IP(k)**, and the following is the textbook definition of a k -round interactive proof system [Arora and Barak, 2009].

Definition 1. A language \mathcal{L} belongs to **IP(k)** if there is a k -round interactive proof system to determine if $x \in \mathcal{L}_f$ in the following sense:

We have a probabilistic polynomial time Verifier, V , and a Prover, P , and both are given the input $x \in \{0, 1\}^n$. The verifier and prover will interact by sending a series of k messages back and forth, producing a transcript $t = \text{Transcript}(V, r, P, x)$, where r is the private randomness the verifier uses.

After the interaction, the verifier will use t to determine whether it should accept

or reject the claim $x \in \mathcal{L}$. Let $\text{Out}(V, r, P, x)$ be the output (accept, reject) of the verifier using randomness r , interacting with P on input x .

We say the interactive proof system can determine if $x \in \mathcal{L}$ if the following two properties hold:

Completeness There is a prover, P such that for each $x \in \mathcal{L}$

$$\mathbb{P}(\text{Out}(V, r, P, x) = \text{accept}) \geq \frac{2}{3}$$

Soundness For every $x \notin \mathcal{L}$, and every prover P'

$$\mathbb{P}(\text{Out}(V, r, P', x) = \text{accept}) \leq \frac{1}{3}$$

The complexity class **IP** is the set of all languages with interactive proof systems.

This is the broadest definition of **IP**. An interesting observation by Furer et al. [1989] is that we can have perfect completeness, $\mathbb{P}(\text{Out}(V, r, P, x) = \text{accept}) = 1$, without sacrificing the power of interactive proofs, although if we shifted to perfect soundness, $\mathbb{P}(\text{Out}(V, r, P, x) = \text{accept}) = 0$, we restrict ourselves back down to **NP**. An important intuition to understand here is that an interactive proof system is *sound* if the verifier will almost never accept a proof for an $x \notin \mathcal{L}$. The interactive proof is *complete* if the verifier will almost always accept a correct proof that $x \in \mathcal{L}$, and indeed, as mentioned, we can upgrade this to always accepting without consequence.

2.2 IP = PSPACE: A Brief History of IPs

Before we delve into the modern era of streaming interactive proofs, we survey the work and major results that have come out of the previous 35 years of IPs. As mentioned, it began in MIT in the early 80's with the paper 'The knowledge complexity of Interactive Proof Systems' by Goldwasser et al. [1985]. The motivation of the paper was two fold, primarily focusing on cryptographic applications, and secondly, on clarifying the idea of what proving something really means. The former covered the introduction of zero knowledge proofs, which we will be exploring in Chapter 5. The latter was the introduction of interactive proofs. Their idea, as discussed in Definition 1, was that a theorem-proving procedure should satisfy three conditions:

1. It's possible to prove a true theorem.
2. It's impossible to prove a false theorem.

3. Communicating the proof should be efficient.

These ideas correspond to completeness, soundness and the verifier running in polynomial time, as discussed.

Independent of Goldwasser, Micali and Rackoff, although published at the same conference, the paper ‘Trading Group Theory for Randomness’ came from Babai [1985]. This paper introduced Arthur-Merlin games. The motivation behind this paper was somewhat different to GMR, Babai had been trying to prove a problem in group theory [Babai and Szemerédi, 1984], specifically attempting to show the problem was in **NP**. He had succeeded in his 1984 paper, conditional on an unproven group theory conjecture. Babai introduces the classes **AM**[**k**], to describe Arthur Merlin games with k rounds of interaction. In the language of GMR and IPs, **AM**[**k**] is the class of interactive proofs with k rounds where Arthur (the verifier) speaks first and has public randomness, i.e., the verifier’s randomness is known to all parties. The name Arthur-Merlin stems from the idea of the all-powerful, although not necessarily trustworthy, wizard Merlin attempting to convince the impatient King Arthur [Arturus, 1136]. Babai showed that the group theory problem he was working on had a constant round IP, and furthermore proved that for constant k , **AM**[k] = **AM**[2] [Babai and Moran, 1988].

Historically, at this point there were two types of interactive proofs, IPs from GMR and Arthur-Merlin games from Babai. The class of problems solvable with interactive proofs, **IP**, is the private randomness analogue of **AM**[poly], which are interactive proofs with a polynomial number of rounds and public randomness. As we mentioned earlier, it was shown shortly after the creation of these two classes that private randomness IPs can be adapted to public randomness by adding two rounds, hence **IP**=**AM**[poly] [Goldwasser and Sipser, 1986].

This is when interactive proofs really began to be interesting to computational complexity scientists. In the GMR paper, the authors provide an IP for the quadratic residue problem, which is well known to be in **NP**. The problem in Babai [1985] was for a problem suspected of being in **NP**. The question of whether **IP** contained problems outside of **NP** became a big goal in the world of theoretical computer science.

In 1986, Goldreich, Micali and Wigderson showed the well known problem of graph non-isomorphism, which many people have failed to show is in **NP**, is in **IP** [Goldreich et al., 1986]. The general consensus after this was that (zero knowledge) **IP** was likely a slightly larger class than **NP**, but not much bigger. However in 1988 Fortnow and Sipser tried to see if **IP** contained **coNP**, the set of decision problems whose *no* instance is checkable in polynomial time [Fortnow and Sipser, 1988]. Unfortunately, they showed that in fact to make a statement such as **coNP** \subset **IP**, one would need a non-relativising technique, i.e. some new proof technique beyond ‘standard’ techniques using black boxes and reductions.

This was a setback for the study of **IP**, however towards the end of 1989 a flurry of work led to the result $P\#P \subset \mathbf{IP}$ [Lund et al., 1992]. $P\#P$ is a large class of problems that can be solved in polynomial time by an algorithm with access to an oracle that can solve a problem in $\#P$, which are the class of counting problems associated to problems in **NP**, such as ‘How many subsets of a list of integers add up to zero?’. It had been proved that the class $P\#P$ contained the polynomial hierarchy [Toda, 1991], so this was a major realisation that the class **IP** covered a huge number of problems.

The next step was to show that $\mathbf{IP}=\mathbf{PSPACE}$, and indeed this was achieved by Shamir [1992], using the techniques of Lund et al. [1992]. This result is a landmark in the world of complexity theory, leading to possibly the most famous result in complexity, the PCP Theorem.

The research following $\mathbf{IP}=\mathbf{PSPACE}$ were shifting towards restricting the power of the verifier. One such direction was that of Babai et al. [1991]. They showed that using error-correcting code, i.e., giving the verifier query access to the error-corrected encoding of the input, all problems in **NP** could be checked by a verifier in polylogarithmic time.

A similar direction restricted the number of queries the verifier could make to the proof, as well as the randomness of the verifier. This concept is captured by a PCP, a probabilistically checkable proof. The prover in this setting sends a gigantic message $\pi \in \{0, 1\}^{\text{poly}(n)}$, the verifier has $r(n)$ random bits, and can form a satisfactory $\text{Out}(V, r, P, x)$ after reading at most $q(n)$ bits of π . The languages for which this works form the complexity class $\mathbf{PCP}[r(n), q(n)]$.

Studies of PCPs were accelerated by the results of Feige et al. [1991] that stated $\mathbf{NP} \subset \mathbf{PCP}[\log(n) \log(\log(n)), \log(n) \log(\log(n))]$. This was followed by the proof that $\mathbf{NP} \subset \mathbf{PCP}[\log(n), \log(n)]$ achieved by Arora and Safra [1998]. The final result, known as the PCP theorem was proved by Arora et al. [1998] and stated $\mathbf{NP}=\mathbf{PCP}[\log(n), O(1)]$. This result says that any problem in **NP** can be checked by a verifier with $\log(n)$ random bits, who only queries constantly many points of the prover’s message. This was massive news in the complexity world, and stands today as one of the greatest results in computer science, and even made it into the New York Times [1992]. For further overview of PCPs, see the textbook of Arora and Barak [2009].

Since these results, work in interactive proofs has continued, albeit with less focus on the theoretical side, and more on the practical side. A significant line of work with interactive proofs has been on Zero Knowledge proofs, which we will study in Chapter 5, although the vast amount of work in that area is outside the scope of this thesis. Another significant line of work is the study of interactive proofs for a space-bounded streaming verifier. This is where our work fits in. In the next few sections we cover the vital definitions, and then do a historical survey of SIPs.

2.3 Interactive Proofs in the Streaming Setting

We now turn to looking at what happens if the verifier can't actually store the input x . We consider verifiers with sublinear space, and $x \in \{0, 1\}^n$. This corresponds to the real-world notion of streaming, where x could be a massive data stream, and the verifier wishes to compute some function of x that is impossible to compute in the streaming model. For Chapters 3 and 4 we continue to use a probabilistic polynomial time verifier, but restrict the model further as now we're dealing with languages the verifier can't compute in sublinear space in a streaming manner as well as in polynomial time.

2.3.1 Languages, Functions and Notation

In the SIP protocols we'll be constructing, we will no longer be dealing with just binary inputs. We allow inputs $x \in \mathbb{F}_q^n$, vectors made up of n elements of the finite field of size q , for some prime q . When we originally discussed proving a statement, $x \in \{0, 1\}^n$, we defined it as checking that a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ satisfies $f(x) = 1$. This difference is more of a subtlety; we are now wishing to confirm $f(x) = y$ for $x \in \mathbb{F}_q^n$, $y \in \mathbb{F}_q$ and $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$. Furthermore, we will ultimately use functions $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$. We henceforth will refer to proof systems to verify $(x, y) \in \mathcal{L}_f = \{(x, y) : y = f(x)\}$. We will refer to verifying ' $x \in \mathcal{L}_f$ ' as being equivalent to verifying $(x, y) \in \mathcal{L}_f = \{(x, y) : y = f(x)\}$, as we can simply let the prover send $y = f(x)$ at the start of the protocol, bundled with whatever other messages it sends. The reason for introducing this notion now, as opposed to before defining **IP**, is that the literature often restricts attention to Boolean functions. Our work on general functions comes from our aim to achieve 'practical' interactive proofs for common real-world problems.

2.3.2 Space Bounds and Streams

We will look into two different types of streaming interactive proofs: one round proofs and multiple round proofs. Before delving into the differences, we will discuss what exactly we mean by a streaming interactive proof.

Definition 2. *The streaming model we use will be the one-pass turnstile streaming model. Our inputs will be $A \in \mathbb{F}_q^n$, and this will arrive as a series of updates $(\delta, i) \in \mathbb{F}_q \times [n]$. Each update (δ, i) tells us to update A_i to $A_i + \delta$. For the most part, we will effectively be considering the stream to be a string of (A_i, i) , where each update simply contains the i th element. However, all our protocols will work in the one-pass turnstile streaming setting. Note in our proofs, the verifier streams the entire input **before** interaction with the prover.*

We assume that the verifier has restricted space $O(s)$, with $s = \log^{(n)} \log(q)$ or $\sqrt{n} \log(q)$ for example, and wishes to verify $x \in \mathcal{L}_f$ for $x \in \mathbb{F}_q^n$. We want to be able to deal with functions $f : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ that can't be solved in the verifier's space, and we also want to be able to deal with functions where the verifier can't store $f(x)$.

We will introduce the methods that allow for this in Section 2.5, but to start with we deal with adapting Definition 1 to formally define SIPs. This definition comes predominantly from the work to establish 'annotated data streams' by Chakrabarti et al. [2009], with the k -round extension coming from the works of Cormode et al. [2011] and Chakrabarti et al. [2013], amongst others.

Definition 3. *A language \mathcal{L} belongs to **SIP(k)** if there is a k -round streaming interactive proof system to determine if $x \in \mathcal{L}$, defined as follows:*

We have a probabilistic polynomial time Verifier, V , with space $O(s)$ and a Prover, P , and both are given the input $x \in \mathbb{F}_q^n$, however the verifier only has streaming access to the input (See Definition 2). The verifier and prover will interact by sending a series of k messages back and forth, producing a transcript $t = \text{Transcript}(V, r, P, x)$, where r is the randomness the verifier uses.

After the interaction, the verifier will use t to determine whether it should accept or reject the claim $x \in \mathcal{L}$. Let $\text{Out}(V, r, P, x)$ be the output (accept, reject) of the verifier using randomness r , interacting with P on input x .

We say the interactive proof system can determine if $x \in \mathcal{L}$ if the following two properties hold:

Completeness *There is a prover, P , such that for each $x \in \mathcal{L}$*

$$\mathbb{P}(\text{Out}(V, r, P, x) = \text{accept}) = 1$$

Soundness *For every $x \notin \mathcal{L}$, and every prover P'*

$$\mathbb{P}(\text{Out}(V, r, P', x) = \text{accept}) \leq \frac{1}{3}$$

*The complexity class **SIP** is the set of all languages with interactive proof systems with polylog verifier space and proof length.*

This is almost identical to the definition of interactive proof systems. The verifier's reduction to $O(s \log(q))$ space translates to the verifier keeping a *summary* of x , not x in its entirety. It is also important to note that the summaries the verifier keeps are *random* summaries, that is, the prover cannot know what is actually being kept. This is an interesting parallel to previously mentioned PCPs. The PCP theorem tells us that for SIPs with the

prover sending a single message (‘non-interactive streaming proofs’) where the verifier *can* store the input, the verifier with $O(\log(n))$ random bits and a constant number of queries to the proof will be able to verify any problem in **NP**. This constant number of queries can instead be thought of as a constant sized summary of the entire proof π , i.e., the verifier views the entire proof, and keeps some summary of size $O(1)$. This seems to be a more powerful notion, however Gertner et al. [2002] shows this is in fact equivalent to the PCP model. This idea of the prover sending a proof after the stream that the verifier can look at, but is potentially larger than the verifier’s space, is called the Annotated Data Stream Model.

2.3.3 Annotated Data Stream Model (Non-interactive SIPs)

In our protocols, we have a data stream \mathcal{S} , which is observed by two parties, a ‘prover’ (P) and a ‘verifier’ (V). The data stream will usually be arranged as a sequence of n tuples of elements, where each tuple typically defines an element of a larger structure, such as a matrix or vector. Abstractly, the verifier wishes to compute some function on \mathcal{S} , $f(\mathcal{S})$, with assistance from the prover. Typically, the prover will provide the value of $f(\mathcal{S})$, along with a proof of its correctness. This yields the *Annotated Data Stream* model, introduced by Chakrabarti et al. [2009]. The definition of this model follows from the definition of **SIP**(k) with $k = 1$, representing the one message (the *annotation*) from the prover after the stream \mathcal{S} . We formalize the model via the definition below.

Definition 4. *We have a prover P , and a verifier V , with the aim of cooperating to compute some function $f(\mathcal{S})$ of the stream \mathcal{S} . The prover provides a message $M_P(\mathcal{S})$ comprised of $\widehat{f(\mathcal{S})}$, the claimed value $f(\mathcal{S})$, and an annotation M^P which supports this claim according to some pre-agreed structure. Define the output of V , Out_V , that depends on the summary the verifier made of the stream $V(\mathcal{S})$, the proposed $\widehat{f(\mathcal{S})}$, V ’s randomly chosen bits \mathcal{R}_V , and the prover’s message.*

$$\text{Out}_V(V(\mathcal{S}), \widehat{f(\mathcal{S})}, \mathcal{R}_V, M^P(\mathcal{S})) = \begin{cases} \widehat{f(\mathcal{S})} & \text{If } V \text{ is convinced} \\ \perp & \text{Otherwise.} \end{cases}$$

A protocol is defined by the functions M^P and Out_V . We say that a protocol is **complete** if

$$\forall \mathcal{S} \quad \exists P : \mathbb{P}[\text{Out}_V(V(\mathcal{S}), \widehat{f(\mathcal{S})}, \mathcal{R}_V, M^P(\mathcal{S})) = f(\mathcal{S})] = 1 \quad (2.1)$$

The Verifier’s protocol is **sound** if

$$\forall \mathcal{S} \quad \forall P', V(\mathcal{S}') : \mathbb{P}[\text{Out}_V(V(\mathcal{S}'), \widehat{f(\mathcal{S}')}, \mathcal{R}_V, M^{P'}(\mathcal{S}')) \notin \{f(\mathcal{S}'), \perp\}] \leq \frac{1}{3}$$

As in previous IP definitions this says that we seek protocols so that an honest

prover (one who faithfully follows the protocol) can always persuade the verifier to accept the correct answer, while a dishonest prover cannot persuade the verifier to accept an incorrect result with more than probability $\frac{1}{3}$. Our protocols allow this probability to be reduced to an arbitrarily small value with minimal cost.

In order to show an annotated data streaming protocol works we need to show completeness and soundness. This is sufficient to check that our protocol will successfully do what we want: verify the computation with a high probability of detecting a malicious prover. For some problems trivial protocols will exist wherein the prover’s message is null, and the verifier evaluates the function in full. For these we seek protocols whose costs for the verifier are substantially lower than this. Ideally, the protocol should run in sublinear memory space for the verifier, without the need for intensive computation and the message should be as small as possible. Similarly, we seek protocols where the honest prover does not have to do substantially more work than simply computing $f(\mathcal{S})$. For our annotated data stream protocols in Chapter 3, we focus primarily on memory space for the verifier and the size of the proof, which we call the communication cost. We therefore primarily will refer to our annotated data stream protocols using the following notation.

Definition 5. *A (h,s) -protocol is a valid annotated data streaming protocol using a message of size $O(h)$ bits, and for a verifier with memory $O(s)$ bits.*

Of course, there are many other considerations, including runtime for the verifier during the streaming, as well as the checking of the prover message, and the overheads for the prover. These costs are our focus in Chapter 4.

2.4 Costs in Streaming Interactive Proofs

The costs we consider in subsequent chapters are communication and space costs, which echo this topic’s origin in communication complexity. Prior work has developed the idea of using interactive proofs to independently verify outsourced computations without duplicating the effort. Recent work has sought to argue that interactive proofs can indeed be practically used for verification. Modern research takes two main approaches, from highly general methods with currently far-from-practical costs, to tackling specific fundamental problems where the overhead of verification is negligible.

Chapters 3 and 4 both consider the ‘negligible overhead’ end of the spectrum and study primitive computations within linear algebra – a core set of tools with applications across engineering, data analysis and machine learning.

With streaming interactive proofs, we will consider the following costs.

Space Costs

Verifier Memory The verifier’s working memory.

Communication The total communication between the two parties.

Interactivity The number of back and forth messages between the prover and verifier.

Computation Costs

Verifier Streaming Cost – The additional work the verifier does over the input.

Verifier Checking Computation – The verifier’s work for the interactive stage.

Prover Overhead – The additional work of the prover beyond solving the problem.

Our concern in Chapter 4 will be explore how useful interactivity is for SIPs.

Previous work in this field has often considered increasing interactivity in order to reduce the two main concerns, verifier memory and communication. In Chapter 4 we consider the natural metric of *time taken*, where we consider bandwidth, latency, and computation time to create an aggregate measure. We examine this metric to determine the ideal level of interactivity in the practical setting.

2.5 SIP Toolbox

We will now outline some of the most useful techniques in streaming interactive proofs. These are the techniques that will be fundamental to all our results, and allow us to solve a series of quintessential problems in interactive proofs, such as identity testing, polynomial evaluation, and implicit function summation. To begin with, we will detail how we consider finite fields.

In line with prior work, all our protocols rely on computations performed over finite fields. For ease of implementation, we use prime fields. Given a prime q , the finite (prime) field \mathbb{F}_q is the set $\{0 \dots q - 1\}$ with addition and multiplication modulo q . Hence, storing field values requires $O(\log q)$ bits. We make use of the fact that in many cases arithmetic in the field and arithmetic over the integers can be directly compatible. However, as we consider more complicated computations, we encounter situations where we seek solutions over the reals, which do not correspond to solutions in the field. To avoid this, we will use scaling and rounding techniques to approximate using field values.

Specifically, one could consider the input to be fixed precision (ρ) rational numbers which can be represented as members of the set $\mathcal{F}_{\rho, M} = \{x \in \mathbb{R} \cap [-M, M] : b^\rho x \in \mathbb{Z}\}$, with respect to a base, b . We then choose the field size q as a function of ρ and M , in order to allow us to maintain the exact correspondence between the field and the fixed precision

rational. We map $y \in \mathcal{F}_{\rho, M}$ to $y' \in \mathbb{F}_q$, where $y' = xb^{\rho} \pmod q$, choosing q to be a prime bigger than $(2M + 1)b^{\rho}$.

2.5.1 Fingerprints

Fingerprints can be thought of as hash functions for large vectors and matrices with additional useful algebraic properties. For $A \in \mathbb{F}_q^{n \times n}$ and $x \in \mathbb{F}_q$, define the matrix fingerprint as

$$F_x(A) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_{ij} x^{in+j}.$$

Similarly, for $u \in \mathbb{F}_q^n$ we have the vector fingerprint

$$F_x^{\text{vec}}(u) = \sum_{i=0}^{n-1} u_i x^i.$$

Lemma 1 tells us probability of two different vectors or matrices having the same fingerprint (over the random choice of x) can be made arbitrarily small by increasing the field size, as an almost immediate consequence of the Schwartz-Zippel lemma [Schwartz, 1980], with the fingerprint application built from Rabin [1981].

Lemma 1 (Rabin [1981]). *Given $A, B \in \mathbb{F}_q^{n \times n}$ and $x \in_R \mathbb{F}_q$, we have*

$$\mathbb{P}_x[F_x(A) = F_x(B) | A \neq B] \leq \frac{n^2}{q}.$$

A similar result holds for F_x^{vec} . In our model, fingerprints can be constructed in constant space (measured in field elements), and with computation linear to the input size.

2.5.2 Low Degree Extensions

Low degree extensions (LDEs) have been used extensively in interactive proofs. LDEs have been used in conjunction with sum-check (Section 2.5.3) in a variety of contexts [Goldwasser et al., 2008; Cormode et al., 2011, 2012]. Formally, for a set of data S an LDE is a low degree polynomial that goes through each data point. Typically, we think of S as being laid out as a vector or d -dimensional tensor indexed over integer coordinates. This polynomial can then be evaluated at a random point r with the property that, like fingerprinting, two different data sets are unlikely to evaluate to the same value at r (inversely proportional to the field size).

Given input as a vector $u \in \mathbb{F}_q^n$, we consider two new parameters, l and d with $n \leq l^d$, and re-index u over $[l]^d$. The d -dimensional LDE of u satisfies $\tilde{f}_u(k_0, \dots, k_{d-1}) =$

u_k for $k \in [n]$ where $k_0 \dots k_{d-1}$ is the base l representation of k . For a random point $r = (r_0, \dots, r_{d-1}) \in \mathbb{F}_q^d$, we have

$$\tilde{f}_u(r_0, \dots, r_{d-1}) = \sum_{k_0}^{l-1} \cdots \sum_{k_{d-1}}^{l-1} u_k \chi_k(r) \quad (2.2)$$

$$\chi_k(r) = \prod_{j=0}^{d-1} \prod_{\substack{i=0 \\ i \neq k_j}}^{l-1} \frac{r_j - i}{k_j - i}, \quad (2.3)$$

where χ_k is the Lagrange basis polynomial. Note that $\tilde{f}_u : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ and $q \geq l$. A similar definition can be used for a matrix $A \in \mathbb{F}_q^{n \times n}$, by reshaping into a vector in $\mathbb{F}_q^{n^2}$.

The polynomials can be evaluated over a stream of updates in space $O(d)$ and time (field operations) per update $O(ld)$ (Cormode et al. [2011]). The time cost of our verifier to evaluate an LDE at one location, r , is $O(nld)$ (for sparse data, n can be replaced with the number of non-zeros in the input).

2.5.3 Sum-Check Protocol

Our final tool is the sum-check protocol of Lund et al. [1992]. Sum-check is a multi-round protocol for verifying the sum

$$G = \sum_{k_0=0}^{l-1} \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(k_0, k_1, \dots, k_{d-1}) \text{ for } g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q. \quad (2.4)$$

For our purposes, g will be a polynomial derived from the LDE of a dataset of size $n = l^d$ (i.e. the d -dimensional tensor representation of the data), and each polynomial used in the protocol will have degree λ , with $\lambda = O(l)$; however, we keep the parameter λ for completeness. Provided that all the checks are passed then the verifier is convinced that (with high probability) the value G was as claimed in (2.4). The original descriptions of the sum-check protocol [Lund et al., 1992; Arora and Barak, 2009] use $l = 2$, however we shift to using arbitrary l , similar to Aaronson and Wigderson [2009] and Cormode et al. [2011, 2012]. The protocol goes as follows:

Stream Processing: V randomly picks $r \in \mathbb{F}_q^d$ and computes $g(r_0, \dots, r_{d-1})$.

Round 1: P computes and sends G and $g_0 : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where

$$g_0(k_0) = \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(k_0, k_1, \dots, k_{d-1}).$$

V checks that $G = \sum_{k_0=0}^{l-1} g_0(k_0)$, computes $g_0(r_0)$ and sends r_0 to P .

\vdots

Round $j + 1$: P has received r_0, \dots, r_{j-1} from V , and sends $g_j : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where

$$g_j(k_j) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(r_0, \dots, r_{j-1}, k_j, \dots, k_{d-1}).$$

V checks if $g_{j-1}(r_{j-1}) = \sum_{k_j=0}^{l-1} g_j(k_j)$, computes $g_j(r_j)$ and sends r_j to P .

\vdots

Round d : P sends $g_{d-1} : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where

$$g_{d-1}(k_{d-1}) = g(r_0, \dots, r_{d-3}, r_{d-2}, k_{d-1}).$$

V checks that $g_{d-2}(r_{d-2}) = \sum_{k_{d-1}=0}^{l-1} g_{d-1}(k_{d-1})$, computes $g_{d-1}(r_{d-1})$, and finally checks this is $g(r_0, \dots, r_{d-2}, r_{d-1})$.

P can express the polynomial g_j as a set $G_j = \{(g_j(x), x) : x \in [\lambda + 1]\}$. In each round V sums the first l elements of this set, and checks it is $g_{j-1}(r_{j-1})$ for $j > 0$, then evaluates the LDE of G_j at r_j , giving a computation cost per round of $O(l + \lambda)$. The verifier also has to do some work in the streaming phase, evaluating the function g at r , with time cost $O(n\lambda d)$. The prover's computation time comes from having to evaluate g at l^{d-j} points in the j th round, and so ultimately evaluating g at $\sum_{j=1}^{d-1} l^{d-j} = O(n)$ points, with a cost per point of $O(\lambda d)$ (we subsequently show how this can be reduced in our protocols for linear algebra). The costs of performing sum-check are summarized as follows:

Communication $O(\lambda)$ per round, for d rounds.

Prover costs $O(n\lambda d)$ time for computation.

Verifier costs $O(d)$ memory cost, $O(n\lambda d)$ time to compute LDE and checking cost $O(d(l + \lambda))$.

Soundness and Completeness This protocol has perfect completeness, and a soundness error of $\frac{\lambda d}{|\mathbb{F}|}$.

In our implementations, we will optimize our methods to ‘stop short’ the sum-check protocol and terminate at round $d - 1$ (this idea is implicit in the work of Aaronson and Wigderson [Aaronson and Wigderson, 2009, Section 7.2]). In this setting, the verifier finds the set

$$\{g(r_0, \dots, r_{d-3}, r_{d-2}, k_{d-1}) : k_{d-1} \in [l]\}.$$

in the stream processing stage, and then checks this against the claimed set of values provided by the prover in round $d - 1$. This increases the space used by the verifier to maintain these l LDE evaluations. However, this does not affect the asymptotic space usage of the verifier, since we assume that V already keeps space proportional to l to handle P 's messages. It does not affect the streaming overhead time; consider the ℓ LDE evaluations the verifier must keep, each update affects only one LDE evaluation (the one with which it shares the final coordinate). Equivalently, this can be viewed as running l instances of sum-check in parallel on the data divided into l partitions. Hence, this appears as an all-round improvement, at least in theory.

2.6 SIPs and Complexity Classes

Section 2.2 discussed the work that led up to a complete characterisation of polynomial-time verifiers in interactive proofs. Unfortunately for SIPs, a similar result has not yet found. We now investigate the history of SIPs, to give further context for our results, particularly in Chapter 5 when we introduce ZK-SIPs. For more detailed insight into the history, Thaler [2014] produced a survey of streaming verification algorithms. We will briefly recap this here, highlighting the relevant work.

An important thing to begin with is that whilst, for our purposes, the concepts of streaming interactive proofs were introduced by Chakrabarti et al. [2009] and Cormode et al. [2011], these are not the first papers to consider space-bounded verifiers. Shortly after the $\mathbf{IP}=\mathbf{PSPACE}$ result Lipton and Condon [Condon, 1992; Condon and Lipton, 1989] gave some classifications what could be solved by an IP with a space-bounded verifier. However, their work considered verifiers with read-only access to the input, and then bounded working space. Our streaming setting is contained within this classification. In streaming the input is read once sequentially, and this sequencing is critical to the difficulty of the problem. Other research with a similar space bound has been done with multiprover interactive proofs, where there are several provers [Feige and Shamir, 1989; Tauman Kalai et al., 2013].

2.6.1 Annotations and Arthur-Merlin Games

The research of Chakrabarti et al. [2009] was the first to relate streaming annotations to interactive proofs. We discussed this non-interactive SIP model in Section 2.3.3. Annotated data streams are non-interactive, the prover sends a single message to the verifier. Chakrabarti et al. [2009] also introduce the concept of a ‘prescient’ protocol. A prescient SIP has the first message arrive at any time, before the verifier reads the input, or even interleaved with the input. The counter to this is an ‘online’ SIP, where the message arrives after the input (or more specifically, the message doesn’t contain information about any of the

input the verifier is yet to see). A classic example of the difference between them would be that of the INDEX problem. In this problem, the input is a stream $A \in \mathbb{F}_q^n$, followed by an index $j \in [n]$. The verifier wishes to recover A_j . Alone, this would require linear space for the verifier in the streaming setting. A prescient protocol would have the prover send j before the input, for a $(\log(q), \log(q))$ -protocol. A lower bound on online (h, s) -protocols, presented in Chakrabarti et al. [2009], shows that $hs = \Omega(n)$.

In Chakrabarti et al. [2009], and future works, the costs for online protocols are shown using communication complexity lower bounds. This is a natural step in the world of streaming. The relationship between communication and streaming [Alon et al., 1999] allows us to use lower bounds from communication complexity to get lower bounds for the storage space needed in a streaming algorithm. A streaming algorithm can be simulated by several parties in a communication complexity setting. For INDEX, Alice would hold A , and Bob would have j . The communication complexity would be the amount of memory Alice must send to Bob in order for him to output A_j . This represents a streaming algorithm where the input A is seen by the verifier, and the message from Alice to Bob would be the memory state of the verifier after seeing A .

This is used to get the above lower bound for online protocols, through the introduction of Online-MA communication. In this setting, we have three parties, Alice, Bob and Merlin. Merlin is the prover and Alice and Bob represent the verifier, as before in the INDEX setting, we have Alice holding A , Bob holding j . Alice and Merlin send a message to Bob, and Bob must output A_j , or \perp if he thinks Merlin is dishonest. Chakrabarti et al. [2013] studies this model and creates a ‘complexity zoo’ of several complexity classes that can be built in this setting.

2.6.2 Tricks from IPs for SIPs

As mentioned in Section 2.2, IP research after $\mathbf{IP}=\mathbf{PSPACE}$ and the PCP theorem continued in a more practical direction. A major result in this direction was the remarkable GKR protocol of Goldwasser et al. [2008]. They introduced a protocol built around the sum check protocol of Lund et al. [1992] that allowed a verifier to check an arithmetic circuit of polynomial size and polylog depth using polylog space and communication, in polylog rounds. This provided an efficient proof system for problems in the complexity class \mathbf{NC} , that is, problems solvable by efficient parallel algorithms. The arithmetic circuit used in the protocol is ‘log-space uniform’, meaning that a log-space algorithm can return the neighbours of any given gates in the circuit.

This celebrated result was utilised by Cormode et al. [2011], who showed that the GKR protocol could be used by a streaming verifier, and hence introduced the idea of efficient powerful streaming interactive proofs. This result kick-started a flurry of activ-

ity in SIPs, where several classes of problems, such as frequency moments and certain graph problems, were given increasingly efficient protocols [Cormode et al., 2013, 2012; Chakrabarti et al., 2013; Thaler, 2013]. It is here where our work in Chapter’s 3 and 4 reside. Our work is amongst other research efforts to give truly practical specialised SIPs [Thaler, 2013; Chakrabarti and Ghosh, 2019; Chakrabarti et al., 2020a].

2.6.3 What can be done with a streaming interactive proof?

Whenever we have a model of computation, the natural question is to ask what it can do. For many models, we use computational complexity classes to summarise the problems they can solve. For example, **NP** is the class of problems that can be verified in polynomial time, and **PSPACE** is the class of problems that can be solved with a polynomial amount of space. As we saw in Section 2.2, this is equivalent to **IP**, the problems that can be verified by a polynomial verifier with polynomial rounds of interaction.

These models are all similar; the verifier has read-only access to some input. This is completely different from the streaming model in that the input can be read only once, and the order of the input is vitally important to the difficulty of the task. Take the **INDEX** problem, which has a stream A of length n , followed by an index $j \in [n]$. The problem requires the streaming algorithm to return A_j . This is a very difficult problem requiring a lot of space if j is the last thing to arrive, but a trivial problem if j arrives before A . However, some classifications for streaming interactive proofs have been made.

An important distinction to make is that of communication complexity classes and computational complexity classes. The above classes are computational complexity classes, but for streaming problems, we shift to the idea of communication complexity.

The communication complexity of a problem $f(x, y)$, introduced by Yao [1979], is the amount of communication between Alice holding x and Bob holding y , in order for Bob to produce $f(x, y)$. In the earlier **INDEX** problem, x is A , and y is j , and they want to solve $f(x, y) = A_j$. This is an interesting analogue to streaming introduced by Alon et al. [1999]. Consider a streaming algorithm for **INDEX** as earlier, the stream begins with A , and ends with j , and the streaming algorithm wishes to output A_j . The amount of space this algorithm needs is exactly the communication complexity of the above problem. Consider the streaming algorithm S , after viewing A , its memory state will be some s_1 . It then views j and needs to output A_j . This memory state s_1 is equivalent to the message Alice would send to Bob. For further reading on this aspect, the textbook of Rao and Yehudayoff [2020] has a good summary of recent results.

Relating this to streaming interactive proofs, as was done in Chakrabarti et al. [2013], we see that to replicate the interaction between a prover and a verifier, we can relate this to a multiparty communication complexity problem. The results of Chakrabarti

et al. [2013] cover several communication complexity classes that arise when considering constant-round streaming interactive proofs. The paper produces several interesting results, including a lower bound for streaming INDEX that we will see in Chapter 5. Furthermore, it is shown that the hierarchical classes of problems solvable by a k -round SIP collapses at $k = 4$, that is any constant-round protocol using more than 4 rounds is no more powerful than a 4-round SIP (up to polynomial blow ups in cost).

As we will discuss at the end of Chapter 5, these results are only for constant round SIPs. There are some classifications for logarithmic round SIPs and beyond, but a comprehensive study of these has yet to be done.

One large set of problems that are known to be solvable by polylog-round SIPs is log-space uniform NC [Goldwasser et al., 2008; Cormode et al., 2012]. These are the problems that can be represented by a log-space uniform arithmetic circuit. We define **SIP** as the class of problems that can be solved by a log space verifier engaging in a log-round protocol with polylog communication. However, extensions of what are possible with SIPs are largely unstudied, and we will discuss conjectures in this area in Chapter 5.

Chapter 3

Non-Interactive Proofs for Data Analysis

This chapter is formed from the 2018 AISTATS paper “Cheap Checking for Cloud Computing: Statistical Analysis via Annotated Data Streams” written by Graham Cormode and Chris Hickey.

3.1 Motivation

The massive leap in popularity of machine learning techniques can be attributed in part not simply to novel algorithms, but also to dramatic increases in scale: much larger models with many parameters to set optimally, and much larger training data sets to determine these parameters. However, this presents a challenge to data owners who do not have a convenient data centre at their disposal. The size of data and computational cost in order to extract accurate models begins to look prohibitive. At the same time, a potential answer has emerged, in the form of outsourced computation. That is, instead of building the infrastructure needed to store and analyse large quantities of data, computation can be ‘rented’ on demand. Initially cloud offerings provided only the barebones of a remote system, but current options provide many tools, libraries and algorithms available to take “off the shelf”.

One doubt remains. If we send data off to the cloud, and request some analysis to be performed, what guarantee do we get that the processing has been done to our satisfaction? The provider has an economic incentive to cut corners: to perform the computation on only a sample of provided data, or to terminate an iterative parameter search before convergence has occurred, for example. Such short cuts yield plausible but suboptimal models. So how could we be assured that the best model has been found, without repeating the computation ourselves or having multiple providers repeat the work, substantially driving up costs?

In this chapter, we adopt the annotated data streams approach (Section 2.3.3) for verifying the models founded by an outsourced provider. Rather than repeating all or some of the computation, we instead provide protocols in which the cloud also provides some extra information that allows us to check a strict adherence to the required computation. The overhead for the cloud provider is minimal – often, the required information is a relatively low cost function of the input data or natural by-products of the target computation. These do not restrict the cloud to use any particular implementation or algorithm; just that they demonstrate that the output meets certain necessary properties. The key part of these protocols is that the information required is very easy for the original data owner to check, based on appropriately defined fingerprints of the input. These fingerprints can be computed flexibly and incrementally from the input as it arrives in any order, so the data owner does not even need to retain a complete copy of the input. The overhead for the data owner is therefore low: it is typically dominated by the cost of sending the data to the cloud and receiving the output of the computation. If the data owner’s checks pass, then they are assured that the computation has been performed by the cloud satisfactorily, with a very high degree of certainty. One can then think of these protocols as providing effective “checksums for computation”.

As discussed in greater detail in Section 2.3.3, work on annotated data streams draws on the theory of Interactive Proofs. This model was developed in the early 1990s as an alternate perspective on computational complexity. An early celebrated result was that the set of computations that could be effectively checked by a “weak” verifier corresponded exactly to the powerful class of computations that could be performed using polynomial space (PSPACE) [Babai, 1985; Goldwasser and Sipser, 1986; Lund et al., 1992; Shamir, 1992]. Such results were initially thought to be of purely theoretical interest. A decade ago [Goldwasser et al., 2008], this topic was revisited from a perspective closer to our own: to what extent could arbitrary programs be checked without fully repeating them? Several strong models were proposed, allowing a large class of computations to be checked in this way. However, the costs were still typically large: programs have to be compiled into non-standard formats (such as circuits with gates performing arithmetic operations), and the overheads for the cloud can be very substantial, often hundreds or even millions of times slower than directly performing the computation.

We break away from this paradigm, and achieve protocols that have minimal overheads by deliberately narrowing the scope of the computations considered. By focusing on a collection of important tasks in machine learning based on linear algebra, we can provide bespoke protocols based on ‘fingerprinting’ the input data that take advantage of specific structure in the target problem. These problems are of sufficient generality that our efforts are repaid.

3.1.1 Chapter Outline

We begin with primitives for ubiquitous steps in data analysis: matrix multiplication and inversion, Cholesky decomposition and eigenvalue finding (Section 3.2). In Section 3.3, we present applications of these to tasks of interest: regression, principal component analysis, and linear discriminant analysis. These core tasks are sufficient to show the power of this paradigm. We provide empirical validation of our claims in Section 3.4.

This work represents some first steps in verifying outsourced computation of machine learning. The next steps are to extend this work to more complex models and algorithms currently enjoying popularity in machine learning, such as deep learning and beyond. Since, at the risk of drastic oversimplification, almost all of machine learning can be performed via numerical data encodings (vectors, matrices and tensors) combined with optimization, we are optimistic that the foundations laid in this work will naturally extend to further protocols for common mining and modelling tasks.

3.1.2 Related Work

As an extension to the discussion in Section 2.3, we briefly survey the most related work in this area. Chakrabarti et al. [2009] introduced the annotated streams model and provided protocols for frequency moments in data streams, and several graph problems, including triangle counting, connectivity and bipartite matchings. They also introduced a square matrix multiplication protocol extending the classical result of Freivalds [1979]. Subsequently, Cormode et al. [2013] and Chakrabarti et al. [2020b] provided further protocols for graph problems, using linear and integer programs to validate optimal matchings and shortest paths. More recently, Daruki et al. [2015] extended results on matrix analysis, provided more general protocols for matrix multiplication, and a protocol for eigenvalue (but not eigenpair) checking. For matrices $A \in \mathbb{F}_q^{k \times n}$ and $B \in \mathbb{F}_q^{n \times k'}$, they show an $(kk'h \log(q), v \log(q))$ -protocol, where $hv \geq n$. These protocols are used to perform shape fitting and clustering, although they shift away from annotated data streams and towards an interactive proof model which allows several exchanges of messages between helper and verifier. The annotated data stream model is generalized by definitions of *streaming interactive proofs* (SIPs) [Cormode et al., 2011, 2012]. As mentioned in Section 2.3.3, note that annotated data stream verification protocols can be considered as single message SIPs.

3.2 Linear Algebraic Checks

In this section, we define protocols for checking a variety of linear algebraic primitives, based on careful use of fingerprints. We use the notation A_i^{\rightarrow} to denote the i th row of the matrix A , and A_i^{\downarrow} to denote the i th column of A .

In many of the protocols, we will stream in a matrix, and then receive the matrix again from the helper, and we want to know whether or not these matrices are the same. This is the EQUALITY problem. The following almost immediate extension of Lemma 1 allows us to check with high probability that two streamed matrices are identical (EQUALITY).

Proposition 1. *There is a $(nm \log(q), \log(q))$ -protocol for EQUALITY on $A \in \mathbb{F}_q^{n \times m}$.*

Proof. To prove that this primitive works, we need to show that it satisfies completeness and soundness. These follow fairly immediately from Lemma 1. If $A = \tilde{A}$ the verifier will always accept, and if $A \neq \tilde{A}$ then the verifier will be tricked by a malicious prover with probability $1 - \frac{nm-1}{q}$. The costs involved are $O(\log(q))$ for the verifier's space cost, and the total communication is the size of A , so $O(nm \log(q))$. Hence, this is a valid $(nm \log(q), \log(q))$ -protocol. \square

Algorithm 1: EQUALITY

Input : $A \in \mathbb{F}_q^{n \times m}$

Output: Verification that the Prover sent A

Verifier

| Choose $x \in_R \mathbb{F}$

| Stream in A and compute $F_x(A)$

Prover

| Send \tilde{A} claiming $\tilde{A} = A$

Verifier

| Receive \tilde{A} and compute $F_x(\tilde{A})$

Check

| $F_x(A) = F_x(\tilde{A})$

3.2.1 Fingerprinting the Gramian Matrix

We first show how to efficiently build a fingerprint of the Gramian matrix $G = A^T A$ given a stream that specifies A (GRAMIAN MATRIX). Recall for $p, q \in \mathbb{F}_q^m$ the outer product of $p \otimes q$ is pq^T , so $(p \otimes q)_{ij} = p_i q_j$.

Lemma 2. For $A \in \mathbb{F}_q^{n \times m}$,

$$F_x(A^T A) = \sum_{i=1}^n F_{x^m}^{\text{vec}}(A_i^{\rightarrow}) F_x^{\text{vec}}(A_i^{\rightarrow})$$

Proof. Given $p, q \in \mathbb{F}_q^m$,

$$\begin{aligned} F_x(p \otimes q) &= \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} p_i q_j x^{im+j} \\ &= \sum_{i=0}^{m-1} \left[p_i x^{im} \sum_{j=0}^{m-1} q_j x^j \right] \\ &= \sum_{i=0}^{m-1} [p_i x^{im} F_x^{\text{vec}}(q)] = F_{x^m}^{\text{vec}}(p) F_x^{\text{vec}}(q). \end{aligned}$$

Using the outer product definition of matrix multiplication, $AB = \sum_{i=0}^{n-1} A_i^{\downarrow} B_i^{\rightarrow}$. We have

$$F_x(A^T A) = \sum_{i=0}^{n-1} F_x \left((A^T)_i^{\downarrow} A_i^{\rightarrow} \right) = \sum_{i=0}^{n-1} F_{x^m}^{\text{vec}}(A_i^{\rightarrow}) F_x^{\text{vec}}(A_i^{\rightarrow}) \quad \square$$

Hence, we can compute the fingerprint of $A^T A$ from A row-by-row and summing the product of row vector fingerprints. This immediately implies a protocol to verify that a matrix G provided by the prover is the Gramian: simply use the above identity to compute $f_x(A^T A)$ from the stream, and check that this is equal to $f_x(G)$.

Proposition 2. There is a $(m^2 \log(q), n \log(q))$ -protocol for GRAMIAN MATRIX on $A \in \mathbb{F}_q^{n \times m}$ (Algorithm 2).

Proof. The protocol is outlined in Algorithm 2. As in Proposition 1, soundness and completeness follow immediately from properties of fingerprints in Section 2.5.1. The communication cost of $m^2 \log(q)$ comes from the prover sending G . The verifier needs to keep n intermediate fingerprints of each row in this protocol. Note that if the matrix A arrives in row major format, the verifier's space requirement will be $O(\log(q))$. \square

As mentioned in the proof, the verifier runs in $O(\log(q))$ if the matrix A arrives in row major format, i.e. the first row appears first, then the second, and so on. A fundamental part of our improvements follow from this notion. Given that the prover is already sending $A^T A$, the communication is $O(m^2 \log(q))$, and for a sacrifice of $nm \log(q)$ additional communication, the verifier can request that the prover also sends A in row major format, if the original stream of A wasn't.

Corollary 1. *There is a $((m^2 + nm) \log(q), \log(q))$ -protocol for GRAMIAN MATRIX on $A \in \mathbb{F}_q^{n \times m}$ (Algorithm 3).*

Proof. The soundness and completeness hold as in Proposition 2. The extension here is the two stage check that $A = \tilde{A}$ and then that $G = \tilde{A}^T \tilde{A}$. \square

Algorithm 2: $(m^2 \log(q), n \log(q))$ GRAMIAN MATRIX (sublinear polynomial space)

Input : $A \in \mathbb{F}_q^{n \times m}$

Output: Verification that the Prover sent $G = A^T A \in \mathbb{F}_q^{m \times m}$

Verifier

Choose $x \in_R \mathbb{F}$

Initialise $\forall i \in [n], F_{x^m}^{\text{vec}}(A_i^{\rightarrow}) = 0$ and $F_x^{\text{vec}}(A_i^{\rightarrow}) = 0$

Stream in A

For A_{ij}

$F_{x^m}^{\text{vec}}(A_i^{\rightarrow}) += A_{ij} x^{mj}$

$F_x^{\text{vec}}(A_i^{\rightarrow}) += A_{ij} x^j$

Form $F_x(A^T A) = \sum_{i=1}^n F_{x^m}^{\text{vec}}(A_i^{\rightarrow}) F_x^{\text{vec}}(A_i^{\rightarrow})$

Prover

 Send G claiming $G = A^T A$

Run

 EQUALITY($A^T A, G$)

3.2.2 Matrix Multiplication

Matrix multiplication has been considered in prior work on annotated streaming. Daruki et al. [2015] showed that any protocol for this problem must have the product of the communication cost and space cost at least $\Omega((k + k')n)$. Our protocol achieves this lower bound up to logarithmic factors (noting that any protocol which reports (AB) requires $\Omega(kk')$ communication for this step). The previous best rectangular matrix multiplication protocol, achieved by Daruki et al. [2015], was a $(kk'h \log(q), v \log(q))$ -protocol for any $hv \geq n$, that used the inner product protocol of Chakrabarti et al. [2009]. Our protocol can be understood as setting $v = 1$, but removing the high overhead factor of n from the communication cost in this case. Chakrabarti et al. [2009] does provide a square matrix multiplication protocol with costs equivalent to our protocol, with $n = k = k'$.

We generalize the previous protocol to solve matrix multiplication. Given two matrices, $A \in \mathbb{F}_q^{k \times n}$ and $B \in \mathbb{F}_q^{n \times k'}$, an extension of Lemma 2 where we replace $A^T \rightarrow A$ and $A \rightarrow B$ gives us

$$F_x(AB) = \sum_{i=0}^{n-1} F_{x^{k'}}^{\text{vec}}(A_i^{\downarrow}) F_x^{\text{vec}}(B_i^{\rightarrow}). \quad (3.1)$$

Algorithm 3: $((m^2 + nm) \log(q), \log(q))$ GRAMIAN MATRIX (log space)

Input : $A \in \mathbb{F}_q^{n \times m}$

Output: Verification that the Prover sent $G = A^T A \in \mathbb{F}_q^{m \times m}$

Verifier

Choose $x \in_R \mathbb{F}_q$
Stream in A , compute $F_x(A)$

Prover

Send G claiming $G = A^T A$, then $\forall i \in [n]$ send $\tilde{A}_i^{\rightarrow}$

Verifier

Receive G and compute $F_x(G)$
Initialise $F_x(\tilde{A}^T \tilde{A}) = 0$

For $\tilde{A}_i^{\rightarrow}$

$F_x(\tilde{A}^T \tilde{A}) += F_{x^m}^{\text{vec}}(\tilde{A}_i^{\rightarrow}) F_x^{\text{vec}}(\tilde{A}_i^{\rightarrow})$

Check

$F_x(A) = F_x(\tilde{A})$
 $F_x(\tilde{A}^T \tilde{A}) = F_x(G)$

Equation (3.1) allows for an efficient matrix multiplication protocol, where the only additional work of the prover is to repeat the input matrices in a convenient order, as in Algorithm 3. To find the fingerprint of AB we need to see each column of A and row of B interleaved in order, i.e. $M^P = [\tilde{A}_0^{\downarrow}, \tilde{B}_0^{\rightarrow}, \dots, \tilde{A}_{n-1}^{\downarrow}, \tilde{B}_{n-1}^{\rightarrow}]$. The verifier uses fingerprints to check that the reordered versions of A and B agree with the versions present in the stream \mathcal{S} , and that the claimed matrix product \widetilde{AB} has the same fingerprint as the fingerprint computed via (3.1). Note again that if the verifier has control over the stream that will define A and B , and can make it arrive as it appears in M^P , then it can immediately store $F_x(AB)$ as required to apply the EQUALITY protocol (Algorithm 1) to determine whether the claimed product $\widetilde{AB} = A \cdot B$.

Theorem 1. *There is a $((k'k + kn + k'n) \log(q), \log(q))$ -protocol for verifying matrix multiplication with $A \in \mathbb{F}_q^{k \times n}$, $B \in \mathbb{F}_q^{n \times k'}$.*

Proof. Algorithm 4 shows the matrix multiplication algorithm in detail. The verifier will keep a fingerprint of A and B , in order to be able to check consistency with the sent \tilde{A} and \tilde{B} . The prover sends \widetilde{AB} claiming it to be AB , and the verifier fingerprints it. The verifier now needs to confirm that $\widetilde{AB} = AB$, which it will do using (3.1). When the prover sends the columns of \tilde{A} and rows of \tilde{B} , the verifier simultaneously forms the fingerprints both these matrices, and the product matrix $\tilde{A}\tilde{B}$. The checks at the end follow the logic that if, with high probability, $\tilde{A} = A$ and $\tilde{B} = B$, then $\tilde{A}\tilde{B} = \widetilde{AB} = AB$.

The communication costs comes from sending A , B and AB , which gives us the

$(k'k + kn + nk) \log(q)$. The verifier only stores a handful of fingerprints, hence the $O(\log(q))$ memory requirement. \square

Algorithm 4: MATRIX MULTIPLICATION

Input : $A \in \mathbb{F}_q^{k \times n}$, $B \in \mathbb{F}_q^{n \times k'}$

Output: Verification the helper sent AB

Verifier

Choose $x \in_R \mathbb{F}_q$

Stream in A and B , compute $F_{x^{k'}}(A)$, $F_x(B)$

Prover

Send \widetilde{AB} claiming $\widetilde{AB} = AB$, then send $[\tilde{A}_i^\downarrow, \tilde{B}_i^{\rightarrow}] \forall i \in [n]$

Verifier

Receive \widetilde{AB} and compute $F_x(\widetilde{AB})$

Initialise $F_x(\tilde{A}\tilde{B}) = 0$

For $[\tilde{A}_i^\downarrow, \tilde{B}_i^{\rightarrow}]$

$F_x(\tilde{A}\tilde{B}) += F_{x^{k'}}^{\text{vec}}(\tilde{A}_i^\downarrow) F_x^{\text{vec}}(\tilde{B}_i^{\rightarrow})$

Check

$F_{x^{k'}}(A) = F_{x^{k'}}(\tilde{A})$

$F_x(B) = F_x(\tilde{B})$

$F_x(\tilde{A}\tilde{B}) = F_x(\widetilde{AB})$

As with GRAMIAN in Algorithm 2, the verifier could store the individual fingerprints

$$\left\{ F_{x^{k'}}^{\text{vec}}(\tilde{A}_i^\downarrow), F_x^{\text{vec}}(\tilde{B}_i^{\rightarrow}) : i \in [n] \right\}$$

as we stream A and B . This will avoid the $O((k'n + nk) \log(n))$ communication cost of sending A and B again, however the verifier needs space $O(n \log(q))$. Our protocols will largely be based on the algorithm for Theorem 1 as this is the most versatile regarding the arbitrary input stream, and requires the least space from the verifier, which is often the bottleneck in SIPs. Additional informal justification, which we delve into deeper in Chapter 4, would be to consider the real life situation of multiplying two large matrices. If we have two 1000×1000 64-bit matrices, storing 1000 fingerprints requires 8 kilobytes, versus storing only a handful of bytes for a single fingerprint; on the other hand, the communication cost will be 8 megabytes for the verifier with the larger memory space, and 24 megabytes with the smaller memory space. We argue the thousand-fold memory saving versus the three-fold communication sacrifice lends favour to the low-space verifier protocol.

Corollary 2. *There is a $(k'k \log(q), n \log(q))$ -protocol for verifying matrix multiplication*

with $A \in \mathbb{F}_q^{k \times n}$, $B \in \mathbb{F}_q^{n \times k'}$.

When our input matrices are constituted of fixed precision rationals, we choose q as follows:

Corollary 3. *Given $A \in \mathcal{F}_{\rho, M}^{k \times n}$ and $B \in \mathcal{F}_{\rho, M}^{n \times k'}$, the matrix AB is in $\mathcal{F}_{2\rho, nM^2}^{n \times n}$, and we choose $q > n(2M + 1)^2 2^{2\rho}$ so that the product can be represented exactly in the field.*

Choosing q to be this large means that when we move A and B from $\mathcal{F}_{\rho, M}$ to $\tilde{A}, \tilde{B} \in \mathbb{F}_q$ by multiplying by 2^ρ , and then compute $\tilde{A}\tilde{B}$ all these values remain in \mathbb{F}_q , and by scaling back down by a factor of $2^{2\rho}$ we get our result in the desired format, without wraparound or rounding errors. The memory required to store elements of \mathbb{F}_q is $\log(q) = O(\log(M) + \log(b)\rho)$, which is proportional to the space required to store the original matrices in $\mathcal{F}_{\rho, M}$ is $\log(M) + \log(b)\rho$.

Lower bounds on computing matrix fingerprints. The verifier needs very little memory for the above protocol, since the matrix is provided in a convenient order. We simply require

$$\left\{ F_{x^{k'}}^{\text{vec}} \left(\tilde{A}_i^\downarrow \right), F_x^{\text{vec}} \left(\tilde{B}_i^\rightarrow \right) : i \in [n] \right\}$$

More generally, we would like to be able to find $F_x(AB)$ from just $F_x(A)$ and $F_x(B)$, without requiring the helper to repeat these. This would reduce communication and simplify the protocol; however, we show this is not possible.

Theorem 2. *Any function g with $F_x(AB) = g(F_x(A), F_x(B))$ for $A, B \in \mathbb{F}_q^{n \times n}$ requires that fingerprints F_x are at least $\Omega(n)$ bits in size.*

Proof. We make use of a hard problem from communication complexity to show the space lower bound. In the DISJOINTNESS problem, two players Alice and Bob each have a bit string, $a, b \in \{0, 1\}^n$, and they wish to see whether for any $i \in [n]$ they have $a_i = b_i = 1$. If we had a function $F_x : \mathbb{F}_q^{n \times n} \rightarrow \mathbb{F}_q$ they could create $n \times n$ matrices A and B with a and b on the diagonals and 0's elsewhere. Then Alice could send Bob $F_x(A)$, Bob could compute $F_x(B)$, and then find $F_x(AB)$ using g . Observe that $AB = \mathbf{0}$ iff strings a and b are disjoint, and is non-zero otherwise. So by comparing $F_x(AB)$ to $F_x(\mathbf{0})$, we can determine the answer to the disjointness problem. The fingerprints must be at least $\Omega(n)$ bits from the corresponding communication complexity of DISJOINTNESS [Yao, 1979]. \square

3.2.3 Eigenvalue Check

We next wish to verify the eigenpairs of a symmetric matrix A , i.e. a matrix such that $\forall i, j, A_{ij} = A_{ji}$. We wish to find pairs (λ_i, v_i) , for all $i \in [n]$, such that A acts on each of

the vectors by only scaling them by λ_i and that the vectors are orthonormal, i.e.

$$Av_i = \lambda_i v_i \quad v_i^T v_j = \delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

However, there is the major problem that (λ_i, v_i) are real, potentially irrational numbers, and we are working with finite precision (i.e. over a finite field). For this and subsequent problems, we need to apply scaling and rounding, as mentioned earlier. There is a tension here, since matrix computations can include values which are very large compared to the input values. We also need to ensure that our approximation tolerance always allows an honest helper to find a satisfying answer, but prevents a dishonest helper from getting a wildly wrong answer accepted.

Prior study on eigenvalue checking has been done by Cormode et al. [2013]. The following results are our study on the eigenpair problem, but follow largely the same strategies, and get largely the same results, as those in Cormode et al. [2013].

Mapping these eigenpairs to the finite field can be delicate, since they may not align with coordinates in the field. Our protocol relies on an integer scaling factor T , which is used to multiply up values from the original domain. The field size q must grow by a corresponding factor to accommodate the large range of values. To tolerate this, we relax to allow approximate eigenvectors as defined below. We first show that rounding to the scaled field \mathbb{F}_{qT} is always possible. Consider a particular eigenvector v_i , and write $\hat{v}_i = Tv_i + r$ so $\hat{v}_i \in \mathbb{F}_{qT}^n$, with $r_j \in [-\frac{1}{2}, \frac{1}{2}]$, and $\hat{\lambda}_i = T\lambda_i + \rho$, where $\rho \in [-\frac{1}{2}, \frac{1}{2}]$ and $\hat{\lambda}_i \in \mathbb{F}_{qT}$. Note that when we scale up these eigenpairs, we also must scale up A . If we were simply scaling up perfectly, so we could exactly represent our eigenpairs in the field, hence $r = 0$ and $\rho = 0$, we would now be working with the eigenpairs for TA , as

$$TA\hat{v}_i = T^2Av_i = T^2\lambda_iv_i = \hat{\lambda}_i\hat{v}_i,$$

The following theorem shows that a ‘‘rounded’’ eigenpair does indeed continue to act like an eigenpair, and the error is quantifiable by the verifier. This theorem has error bounds dependent on the frobenius norm, $\|A\|_F = \sqrt{\sum_{i,j=0}^{n-1} A_{ij}^2}$, which can be calculated by the verifier by requesting A from the prover atomically, and checking consistency with fingerprints.

Theorem 3. *For symmetric $A \in \mathbb{F}_q^{n \times n}$ if $\{(\lambda_i, v_i) : i \in [n]\}$ are the eigenpairs of A with*

approximated eigenpairs using scaling factor $T \in \mathbb{N}$;

$$\begin{aligned}\widehat{v}_i &= Tv_i + r & \text{for } r \in \left[-\frac{1}{2}, \frac{1}{2}\right]^n, \widehat{v}_i \in \mathbb{F}_{qT}^n \\ \widehat{\lambda}_i &= T\lambda_i + \rho & \text{for } \rho \in \left[-\frac{1}{2}, \frac{1}{2}\right], \widehat{\lambda}_i \in \mathbb{F}_{qT}\end{aligned}$$

then

$$\begin{aligned}\|TA\widehat{v}_i - \widehat{\lambda}_i\widehat{v}_i\|_\infty &\leq \left[T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4} \right], \text{ and} \\ |\widehat{v}_i^T \widehat{v}_j - T^2\delta_{ij}| &\leq \left[T\sqrt{n} + \frac{n}{4} \right].\end{aligned}$$

Proof. Using the following bounds:

$$|\lambda_i| \leq \|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} \quad \|v_i\|_2 = 1 \quad \|r\|_2 \leq \frac{\sqrt{n}}{2} \quad |\rho| \leq \frac{1}{2}$$

And for arbitrary vectors $x, y \in \mathbb{F}^n$ and matrices $X \in \mathbb{F}^{n \times n}$:

$$\|x\|_\infty \leq \|x\|_2 \quad \|Xx\|_2 \leq \|X\|_2\|x\|_2 \quad \|X\|_2 \leq \|X\|_F \quad |x^T y| \leq \|x\|_2\|y\|_2$$

We can then expand out the maximum norm on these vectors to show the bounds for the Theorem, using as well the fact that $Av_i = \lambda_i v_i$ and $v_i^T v_j = \delta_{ij}$.

$$\begin{aligned}\|TA\widehat{v}_i - \widehat{\lambda}_i\widehat{v}_i\|_\infty &\leq \|TA\widehat{v}_i - \widehat{\lambda}_i\widehat{v}_i\|_2 \\ &= \|T^2Av_i + TAr - T^2\lambda_i v_i - T\rho v_i - T\lambda_i r - \rho r\|_2 \\ &\leq T\|A\|_2\|r\|_2 + T|\rho|\|v_i\|_2 + T|\lambda_i|\|r\|_2 + |\rho|\|r\|_2 \\ &\leq \frac{T\sqrt{n}}{2}\|A\|_2 + \frac{T}{2} + \frac{T\sqrt{n}}{2}\|A\|_2 + \frac{\sqrt{n}}{4} \\ &\leq T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4}\end{aligned}$$

$$\begin{aligned}|\widehat{v}_i^T \widehat{v}_j - T^2\delta_{ij}| &\leq |\widehat{v}_i^T \widehat{v}_j - T^2\delta_{ij}| \\ &\leq |(Tv_i + r)^T (Tv_j + r) - T^2\delta_{ij}| \\ &\leq |Tv_i^T r + Tr^T v_j + r^T r| \\ &\leq 2T\|r\|_2 + \|r\|_2^2 \\ &\leq T\sqrt{n} + \frac{n}{4}.\end{aligned}$$

□

These bounds show how the error scales with the scaling factor T . To better interpret this, we next show that this error can be made arbitrarily small by increasing T .

Theorem 4. *If we have*

$$\|TA\hat{v}_i - \hat{\lambda}_i\hat{v}_i\|_\infty \leq \left[T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4} \right] \quad \text{and} \quad |\hat{v}_i^T \hat{v}_j - T^2 \delta_{ij}| \leq \left[T\sqrt{n} + \frac{n}{4} \right].$$

Then we have (in \mathbb{R})

$$\frac{|T\lambda_i - \hat{\lambda}_i|}{T} \leq E_{A,n}(T),$$

where $E_{A,n}(T)$ is a function of T dependent on n and A with $E_{A,n}(T) \rightarrow 0$ as $T \rightarrow \infty$.

Proof. We want to bound $\|\hat{v}_i\|_\infty^2$. We can use

$$T\sqrt{n} + \frac{n}{4} + 1 \geq \left[T\sqrt{n} + \frac{n}{4} \right] \geq \left| \hat{v}_i^T \hat{v}_i - T^2 \right| \geq \left| \|\hat{v}_i\|_2^2 - |T^2| \right|$$

With this, if we assume $T \geq \frac{5n+4}{4\sqrt{n}}$ (as T will be large), we obtain the bound

$$\begin{aligned} \|\hat{v}_i\|_2^2 &\geq T^2 - T\sqrt{n} - \frac{n}{4} - 1 \\ &\geq T^2 - 2T\sqrt{n} + n \quad \left(\text{as } T \geq \frac{5n-4}{4\sqrt{n}} \implies n - T\sqrt{n} \leq -\left(\frac{n}{4} + 1\right) \right) \\ &= (T - \sqrt{n})^2 \end{aligned}$$

As A is symmetric, we can write $A = VDV^T$, where V is the orthogonal matrix of eigenvectors, and D is the diagonal matrix of corresponding eigenvalues.

$$\begin{aligned} \|TA\hat{v}_i - \hat{\lambda}_i\hat{v}_i\|_\infty &\geq \frac{1}{\sqrt{n}} \left\| VDV^T\hat{v}_i - \hat{\lambda}_i\hat{v}_i \right\|_2 \\ &= \frac{1}{\sqrt{n}} \left\| TV(D - \hat{\lambda}_i I)V^T\hat{v}_i \right\|_2 \\ &= \frac{1}{\sqrt{n}} \left\| (TD - \hat{\lambda}_i I)V^T\hat{v}_i \right\|_2 \\ &\geq \frac{1}{\sqrt{n}} \min_j \left(|T\lambda_j - \hat{\lambda}_i| \right) \|V^T\hat{v}_i\|_2 \\ &= \frac{1}{\sqrt{n}} \left(|T\lambda_j - \hat{\lambda}_i| \right) \|\hat{v}_i\|_2 \\ &\geq \frac{T - \sqrt{n}}{\sqrt{n}} \min_j \left(|T\lambda_j - \hat{\lambda}_i| \right) \end{aligned}$$

This tells us that the error in the approximated eigenvalue is

$$\frac{\min_j \left(|T\lambda_j - \hat{\lambda}_i| \right)}{T} \leq \frac{\sqrt{n} \left(T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4} + 1 \right)}{T(T - \sqrt{n})} = E_{A,n}(T)$$

So if we have a desired error $\epsilon > 0$, and wish to ensure that there is a (true) eigenvalue ϵ -close to the approximate eigenvalue, $\frac{\min_j (|T\lambda_j - \hat{\lambda}_i|)}{T} < \epsilon$, we need to ensure we can pick a T s.t. $E_{A,n}(T) - \epsilon$. As T tends to infinity, this bound positively approaches 0, as $\|A\|_F$ and n are independent of T , we can always choose a T so the $O(T^2)$ denominator sufficiently dominates the $O(T)$ numerator. \square

This means that if we want to find eigenvalues within certain error ϵ , we simply have to check that the bounds of Theorem 4 hold, using a T^* satisfying $\epsilon = E_{A,n}(T)$. $E_{A,n}(T)$ is $O\left(\frac{n\|A\|_F}{T}\right)$, hence it suffices to pick $T = O\left(\frac{n^{3/2}\|A\|_F}{\epsilon}\right)$. It's worth noting that the verifier can use the prover to compute $\|A\|_F$ by having the prover repeat A element-by-element, and can check the same matrix is sent using fingerprinting, and compute the norm in order to determine the scaling factor. This extra step doesn't affect the asymptotic costs, and we do not include it in Algorithm 5.

Algorithm 5: SYMMETRIC EIGENSOLVER

Input : $A \in \mathbb{F}_q^{n \times n}$, symmetric, $\epsilon > 0$

Output: Verification the prover sent eigenpairs with ϵ precision

Verifier

Choose $x \in_R \mathbb{F}$
 Stream in A and compute $F_x(A)$, $\|A\|_F$
 Pick $T = O\left(\frac{n^{3/2}\|A\|_F}{\epsilon}\right)$ and send to P
 Compute $F_x(TA) = TF_x(A)$

Prover

| Send D .

Verifier

| Compute $F_x(TA - D) = F_x(TA) - F_x(D)$

Run

| GRAMIAN MATRIX(V)
 | MATRIX MULTIPLICATION($TA - D, V$)

Verifier

| Whilst receiving $(TA - D)V$ and $V^T V$

Check For each column of $(TA - D)V$

$$\|((TA - D)V)_i\|_\infty \leq \left\lceil T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4} \right\rceil$$

Check

$$\|V^T V - TI\|_{\max} \leq \left\lceil T\sqrt{n} + \frac{n}{4} \right\rceil$$

Theorem 5. *There is an annotated streaming $\left(n^2 \log\left(\frac{n^{3/2}\|A\|_F}{\epsilon}\right), \log\left(\frac{n^{3/2}\|A\|_F}{\epsilon}\right)\right)$ -protocol for finding the eigenvalues of a symmetric matrix $A \in \mathbb{F}_q^{n \times n}$ to a precision of $\epsilon > 0$.*

Proof. The protocol is laid out in Algorithm 5. The verifier can compute $\|A\|_F$ and a fingerprint of A as the input is received, then pick $T = O\left(\frac{n^{3/2}\|A\|_F}{\epsilon}\right)$, and send this to the prover. The prover provides the claimed scaled matrices of (approximate) eigenvectors V and corresponding eigenvalues D . We then make use of the matrix multiplication protocol to compute $(TA - D)V$. If the eigenvectors were exact, this would be $\mathbf{0}$; since they are approximate, we allow each entry in the result matrix to be at most $\left\lceil T\sqrt{n}\|A\|_F + \frac{T}{2} + \frac{\sqrt{n}}{4} \right\rceil$ (computed over the reals and rounded to the finite field), and can verify that each entry of the product satisfies this bound. We also need to check that the eigenvectors are (almost) orthogonal, that is that each entry of $(V^T V - TI)$ is at most $\left\lceil T\sqrt{n} + \frac{1}{4}n \right\rceil$. Thus we need two invocations of the matrix multiplication protocol, evaluated over a sufficiently large fi-

nite field. Setting T according to the above bounds gives the claimed costs. Note that when we say ‘Run’ in the pseudocode of Algorithm 5, we are always running another annotated streaming protocol, the verifier is never sending a message, and these protocols continue to be non-interactive. \square

Once again, we consider the consequences of our input being $A \in \mathcal{F}_{\rho, M}^{n \times n}$, and choose the value of q as follows

Corollary 4. *Given $A \in \mathcal{F}_{\rho, M}^{n \times n}$ and desired precision ϵ , our matrices V and D are in $\mathcal{F}_{2^{\rho'}, nM^2}^{n \times n}$, where $\rho' = \max\{\rho, -\log \epsilon\}$. Therefore, to keep a direct correspondence between verification done in the field, and our real values, we choose $q > n^2(2M + 1)^2 2^{2\rho'}$.*

This value of q is determined by the matrix multiplication step, and the fact that the largest possible eigenvalue of A is bounded by $n\|A\|_{\max}$, where $\|A\|_{\max}$ is M .

3.2.4 Matrix Inversion

For matrix inversion we again scale the field by an integer factor T . Over this expanded field, given an $n \times n$ matrix A , the prover wants to find a matrix B such that $AB = TI$. To allow for rounding, we can relax the requirement of exact equality, and instead seek a matrix B that acts approximately like an inverse. We first show that such a matrix is guaranteed to exist. In order to do this, we consider the condition number of our input, $\kappa(A) = \|A^{-1}\|_2 \cdot \|A\|_2$. We need our problems to be well-conditioned (i.e. a low condition number).

Lemma 3. *For an invertible $A \in \mathbb{F}_q^{n \times n}$, the prover can find a matrix $B \in \mathbb{F}_{qT}^{n \times n}$ satisfying*

$$\|AB - TI\|_{\max} \leq T\epsilon \quad (3.2)$$

if $T \geq \frac{n^2 \kappa(A)}{2\epsilon}$ and $\epsilon > 0$

Proof. Consider the true inverse of A computed over the reals, A^{-1} . Then we can define $B = TA^{-1} + E$ so that $B \in \mathbb{F}_{qT}^{n \times n}$ and $\forall i, j$ and $|E_{i,j}| \leq \frac{1}{2}$. We can then map the entries of B directly into the field \mathbb{F}_{Tq} . We then have

$$\|AB - TI\|_{\max} \leq \|AE\|_2 \leq \frac{1}{2}n^2\|A\|_{\max}$$

That is, the error is at most $\frac{n^2 \kappa(A)}{2T}$. However, we need $B \in \mathbb{F}_{qT}^{n \times n}$, so we need to ensure T is sufficiently large to deal with the maximum entry in A^{-1} . A^{-1} can contain very large values, depending on the condition number $\kappa(A) = \|A^{-1}\|_2 \cdot \|A\|_2$. Since $\|A\|_2 \geq$

Algorithm 6: INVERT MATRIX

Input : $A \in \mathbb{F}_q^{n \times n}$, ϵ , $\kappa(A)$

Output: Verification the helper sent B approximating A^{-1} with ϵ precision

Verifier

Choose $x \in_R \mathbb{F}$

Stream in A and compute $f_x(A)$, $\|A\|_{\max}$

Pick $T > \frac{n^2 \kappa(A)}{\epsilon}$ and send to P

Run

MATRIX MULTIPLICATION(A, B)

Verifier

Whilst receiving AB

Check

$$\|AB - TI\|_{\max} \leq \frac{1}{2}n^2\|A\|_{\max}$$

$\|A\|_{\max}$, we see that

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2 \geq \|A\|_{\max} \|A^{-1}\|_{\max}$$

So, given a bound on $\kappa(A)$, we choose a field large enough to represent $\frac{\kappa(A)}{\|A\|_{\max}} \geq \|A^{-1}\|_{\max}$. For this theorem, and several involving this new matrix inversion protocol, we hence consider the condition number in our scaling factor size. The algorithm requires the condition number as an input, which is unfortunate, but is a regular issue in numerical linear algebra. We can set the parameter T to be as large as needed to make this error value some small ϵ , so set it to be $T > \frac{n^2 \kappa(A)}{\epsilon}$. \square

Theorem 6. *There is a streaming interactive $\left(n^2 \log\left(\frac{n\kappa(A)q}{\epsilon}\right), \log\left(\frac{n\kappa(A)q}{\epsilon}\right)\right)$ -protocol to invert an invertible matrix $A \in \mathbb{F}_q^{n \times n}$ with the above criteria, (3.2).*

Proof. The protocol is based on the above Lemma: we require the helper to provide such a matrix B under the extended field \mathbb{F}_{Tq} , along with the claimed value of AB . We then run the above protocol for matrix multiplication, and check that each entry of AB meets the required size bound. This ensures that the required condition on entries is met. The cost of storing the fingerprints is $O(\log(nq\kappa(A)/\epsilon))$, and the communication costs come from sending B over \mathbb{F}_{Tq} , which is $O(n^2 \log(nq\kappa(A)/\epsilon))$. \square

In the case that the matrix is singular, the helper could demonstrate this by showing that there is an eigenvalue of A that is 0 to our precision.

3.2.5 Cholesky Decomposition

If we have a positive semi-definite matrix $A \in \mathbb{F}_q^{n \times n}$, the Cholesky Decomposition of A involves finding a lower triangular matrix $L \in \mathbb{R}^{n \times n}$, with $A = LL^T$. As before, we seek an approximate answer, $\widehat{L} \in \mathbb{F}_{q^T}^{n \times n}$.

Theorem 7. *Given a positive semi-definite matrix $A \in \mathbb{F}_q^{n \times n}$, and $T \geq \frac{2n\|A\|_F}{\epsilon}$ there is an annotated steaming $\left(n^2 \log\left(\frac{qn\|A\|_F}{\epsilon}\right), \log\left(\frac{qn^{\frac{3}{2}}\|A\|_F}{\epsilon}\right)\right)$ -protocol to find $\widehat{L} \in \mathbb{F}_{q^T}^{n \times n}$ satisfying*

$$\|\widehat{L}\widehat{L}^T - T^2A\|_{\max} \leq T^2\epsilon$$

Proof. The protocol has the helper provide $\widehat{L} \in \mathbb{F}_{q^T}^{n \times n}$ in order, and the helper executes the matrix multiplication protocol. It is straightforward to check that \widehat{L} is lower triangular as it is presented. We also know, from the numerical stability of the Cholesky decomposition [Trefethen and Bau III, 1997], that the values in L are subject to the same blow-up that can only be as much as \sqrt{n} . For a $\widehat{L} = TL + E$ with $E_{i,j} \in [-\frac{1}{2}, \frac{1}{2}]$, we have

$$\begin{aligned} \|\widehat{L}\widehat{L}^T - T^2A\|_{\max} &\leq \|TLE^T + TEL^T + EE^T\|_2 \\ &\leq 2T\|L\|_2\|E\|_2 + \|E\|_2^2 \\ &\leq Tn^{\frac{3}{2}}\|A\|_F + \frac{n^2}{4} \end{aligned}$$

So we have an error ϵ at most $\frac{n^{\frac{3}{2}}\|A\|_F}{T} + \frac{n^2}{4T^2}$. Picking $T \geq 2n^{\frac{3}{2}}\|A\|_F/\epsilon$ suffices for any given ϵ (since $\|A\|_F \geq 1$). Via the matrix multiplication protocol, we just check $\|\widehat{L}\widehat{L}^T - T^2A\|_{\max} \leq T^2\epsilon$. The resulting protocol has communication cost $n^2 \log\left(\frac{qn^{\frac{3}{2}}\|A\|_F}{\epsilon}\right)$ and memory cost of $\log\left(\frac{qn^{\frac{3}{2}}\|A\|_F}{\epsilon}\right)$, as claimed. \square

3.2.6 Symmetric Generalised Eigenvalues

The Cholesky Decomposition allows us to solve the symmetric generalised eigenvalue problem for $A, B \in \mathbb{F}_q^{n \times n}$, with A symmetric, and B symmetric positive semi-definite:

$$\text{Find } V, D \in \mathbb{R}^{n \times n} \text{ such that } AV = BVD$$

The prover can do this by finding the Cholesky Decomposition of $B = LL^T$. Then the prover can find the eigenvalues of the symmetric matrix $C = L^{-1}AL^{-T}$ to get matrices V', D' with $L^{-1}AL^{-T}V = CV' = V'D'$. We set $V = L^{-T}V'$ and $D = D'$ to see

$$L^{-1}AL^{-T}V' = V'D' \quad \implies \quad L^{-1}AV = L^TV'D \quad \implies \quad AV = BVD$$

Algorithm 7: CHOLESKY DECOMPOSITION

Input : Positive semi-definite $A \in \mathbb{F}_q^{n \times n}$, ϵ

Output: Verification the helper sent \hat{L} approximating the lower triangular L with ϵ precision such that $A = LL^T$

Verifier

Choose $x \in_R \mathbb{F}$
Stream in A and compute $f_x(A)$, $\|A\|_F$
Pick $T > \frac{2n^{\frac{3}{2}}\|A\|_F}{\epsilon}$ and send to P

Run

GRAMIAN MATRIX(\hat{L})

Prover

Whilst sending $(\hat{L}\hat{L}^T)_{ij}$ send \hat{A}_{ij} .

Verifier

Whilst receiving \hat{L}
Check For $i < j$
| $\hat{L}_{ij} = 0$
Whilst receiving $\hat{L}\hat{L}^T$ and \hat{A}
Check
| $\|\hat{L}\hat{L}^T - T^2A\|_{\max} \leq \frac{1}{2}n^2\|A\|_{\max}$
| $F_x(\hat{A}) = F_x(A)$

Theorem 8. *There is a streaming annotated $(n^2 \log(qT), \log(qT))$ -protocol with $T = \kappa(B)q^2n^4/\epsilon$ for verifying $\hat{V}, \hat{D} \in \mathbb{F}_{qT}^{n \times n}$ approximately solving the generalised eigenvalue problem for $A \in \mathbb{F}_q^{n \times n}$ symmetric, and $B \in \mathbb{F}_q^{n \times n}$ symmetric positive semi-definite:*

$$AV = BVD$$

with ϵ the maximum absolute error between \hat{D} and D .

Proof. The Cholesky Decomposition allows us to solve the symmetric generalised eigenvalue problem for $A, B \in \mathbb{F}_q^{n \times n}$, with A symmetric, and B symmetric positive semi-definite;

$$\text{Find } V, D \in \mathbb{R}^{n \times n} \text{ such that } AV = BVD$$

The prover can do this by finding the Cholesky Decomposition of B , L and then performing finding the eigenvalues of the symmetric matrix $C = L^{-1}A(L^{-1})^T$ to get matrices V', D' with $CV' = V'D'$. $D = D'$, and $V = L^{-1}V'$ are the solutions we desire.

With our approximations, we use our matrix inversion and Cholesky Decomposition proto-

cols to find, using scaling factor T_1 , we have that \hat{C} will be in $\mathbb{F}_{qT_1}^{n \times n}$.

$$\hat{C} = \widehat{(\hat{L})^{-1} A (\hat{L})^{-1}}^T$$

So we have

$$\begin{aligned} \hat{L}\hat{L}^T &= T_1^2 B + E_1 & E_1 &\in \left[-\frac{n\|B\|_F}{2T_1}, \frac{n\|B\|_F}{2} \right]^{n \times n} \\ \widehat{(\hat{L})^{-1}} &= T_1 I + E_2 & E_2 &\in \left[-n\|\hat{B}\|_F - \frac{n^2}{4}, n\|\hat{B}\|_F + \frac{n^2}{4} \right]^{n \times n} \end{aligned}$$

From an honest prover, we receive approximate eigenpairs, \hat{U}, \hat{D} with diagonal $\hat{\lambda}$, of \hat{C} from the helper, with scaling factor T giving error ϵ^δ , satisfying

$$\begin{aligned} \|T\hat{C}\hat{U} - \hat{D}\hat{U}\|_{\max} &\leq \left[Tn\|C\|_F + \frac{T\sqrt{n}}{2} + \frac{n}{4} \right] \\ \|\hat{U}^T \hat{U} - T^2 I\|_{\max} &\leq \left[T\sqrt{n} + \frac{n}{4} \right]. \end{aligned}$$

Let $D^\delta, U^\delta \in \mathbb{R}^{n \times n}$ be the true eigenvalues and eigenvectors of \hat{C} , So

$$\hat{C}U^\delta = U^\delta D^\delta.$$

We know that $\|TD^\delta - \hat{D}\|_{\max}$ will be at most $T\epsilon^\delta$. Furthermore let $\hat{L}^T V^\delta = U^\delta$, so

$$\begin{aligned} \hat{C}U^\delta &= U^\delta D^\delta \\ \widehat{(\hat{L})^{-1} A (\hat{L})^{-1}}^T \hat{L}^T V^\delta &= \hat{L}^T V^\delta D^\delta \\ \hat{L} \widehat{(\hat{L})^{-1} A (\hat{L})^{-1}}^T \hat{L}^T V^\delta &= \hat{L} \hat{L}^T V^\delta D^\delta \\ (T_1 I + E_2) A (T_1 I + E_2)^T V^\delta &= (T_1^2 B + E_1) V^\delta D^\delta \\ \left(A + \frac{E_2 A + A E_2^T + E_2 E_2^T}{T_1^2} \right)^T V^\delta &= \left(B + \frac{E_2}{T_1^2} \right) V^\delta D^\delta \end{aligned}$$

By using the eigenvalue perturbation theory of Trefethen and Bau III [1997], we can say that there exists $V \in \mathbb{R}^{n \times n}$, $D \in \mathbb{R}^{n \times n}$ with diagonal λ , so

$$\lambda_i = \lambda_i^\delta + v_i^{\delta T} \left(\frac{E_2 A + A E_2^T + E_2 E_2^T}{T_1^2} - \lambda_i^\delta \frac{E_1}{T_1^2} \right) v_i^\delta$$

So

$$\begin{aligned}
|\lambda_i - \lambda_i^\delta| &= |v_i^{\delta T} \left(\frac{E_2 A + A E_2^T + E_2 E_2^T}{T_1^2} - \lambda_i^\delta \frac{E_1}{T_1^2} \right) v_i^\delta| \\
&\leq \|v_i^{\delta T}\|_2 \left\| \left(\frac{E_2 A + A E_2^T + E_2 E_2^T}{T_1^2} - \lambda_i^\delta \frac{E_1}{T_1^2} \right) \right\|_2 \|v_i^\delta\|_2 \\
&\leq \left\| \frac{E_2 A + A E_2^T + E_2 E_2^T - \lambda_i^\delta E_1}{T_1^2} \right\|_2 \\
&\leq \frac{\|E_2 A\|_2 + \|A E_2^T\|_2 + \|E_2 E_2^T\|_2 + \|\lambda_i^\delta E_1\|_2}{T_1^2} \\
&\leq \frac{2\|E_2\|_2 \|A\|_2 + \|E_2\|_2^2 + \|\hat{C}\|_2 \|E_1\|_2}{T_1^2} \\
&\leq \frac{2n \left(n\|B\|_2 + \frac{n^2}{4} \right) \|A\|_2 + n^2 \left(n\|B\|_2 + \frac{n^2}{4} \right)^2 + \|\hat{C}\|_2 n \left(\frac{n\|B\|_2}{2} \right)}{T_1^2} \\
&\leq \frac{\left(2n^2\|B\|_2 + \frac{n^3}{2} \right) \|A\|_2 + \left(n^4\|B\|_2^2 + \frac{\|B\|_2 2n^5}{2} + \frac{n^6}{16} \right) + \left(\frac{\|\hat{C}\|_2 n^2 \|B\|_2}{2} \right)}{T_1^2} \\
&\leq \frac{\left(2n^2\|\hat{B}\|_2 + \frac{n^3}{2} \right) \|A\|_2 + \left(n^4\|\hat{B}\|_2^2 + \frac{\|B\|_2 2n^5}{2} + \frac{n^6}{16} \right) + \left(\frac{\|\hat{C}\|_2 n^2 \|B\|_2}{2} \right)}{T_1^2} \\
&\leq \frac{32n^2\|B\|_2 \|A\|_2 + 8n^3\|A\|_2 + 16n^4\|B\|_2^2 + 8\|B\|_F n^5 + n^6 + 8\|\hat{C}\|_2 n^2 \|B\|_2}{T_1^2}
\end{aligned}$$

As, $A, B \in \mathbb{F}_q$, we have $\|A\|_2, \|B\|_2 \leq qn$, $\|\hat{C}\|_2 \leq qnT_1$

$$\begin{aligned}
|\lambda_i - \lambda_i^\delta| &\leq \frac{32n^2(nq)^2 + 8n^3nq + 16n^4(nq)^2 + 8nqn^5 + n^6 + 8qnT_1n^2nq}{T_1^2} \\
&\leq \frac{32n^4q^2 + 8n^4q + 16n^6q^2 + 8n^6q + n^6 + 8qnT_1n^3q}{T_1^2} \\
&\leq \frac{8n^4(4q^2 + q) + n^6(16q^2 + 8q + 1) + 8qnT_1n^3q}{T_1^2} \\
&\leq \frac{q^3n^4}{T_1}
\end{aligned}$$

If we have that $q \geq 20$, $n \geq 3$ and $T_1 \geq n^2\kappa(B)$. We also have

$$|T\lambda_i^\delta - \hat{\lambda}_i| \leq T\epsilon^\delta$$

So

$$\begin{aligned} |T\lambda_i - \hat{\lambda}_i| &\leq |T\lambda_i - T\lambda_i^\delta| + |T\lambda_i^\delta - \hat{\lambda}_i| \\ &\leq T \left(\frac{q^3 n^4}{T_1} + \epsilon^\delta \right) \end{aligned}$$

If we want $\frac{|T\lambda_i - \hat{\lambda}_i|}{T}$ to be equal to ϵ we must choose T_1 such that

$$\frac{|T\lambda_i - \hat{\lambda}_i|}{T} \leq \frac{q^3 n^4}{T_1} + \epsilon^\delta$$

Therefore, to get the generalised eigenvalues to an error of ϵ we must choose T, T_1 such that

$$\begin{aligned} T &= \frac{q^2 n^{\frac{5}{2}}}{\epsilon^\delta} \geq \frac{qn^{\frac{3}{2}} \|\hat{C}\|}{\epsilon^\delta} \\ T_1 &= \frac{\kappa(B)q^3 n^4}{\epsilon - \epsilon^\delta} \end{aligned}$$

Where $\epsilon^\delta < \epsilon$.

If we take $\epsilon^\delta = \frac{\epsilon}{2}$, then we have

$$\begin{aligned} T &= \frac{2q^2 n^{\frac{5}{2}}}{\epsilon} \\ T_1 &= \frac{2\kappa(B)q^3 n^4}{\epsilon} \end{aligned}$$

And our total protocol is therefore, assuming $q > n$, $(n^2 \log(\kappa(B)q^3 n^4 / \epsilon), \log(\kappa(B)q^3 n^4 / \epsilon))$. □

3.3 Statistical Analysis

With the primitives outlined above, we can construct protocols for several common statistical analysis tasks.

3.3.1 Ordinary Least Squares

We first consider ordinary least squares regression (OLS), where we aim to find the optimal linear relationship between a data set X and a set of observations y .

Definition 6. Let $S = \langle \{y_1, X_1\}, \dots, \{y_n, X_n\} \rangle$ where $y_i \in \mathbb{F}_q$ is an observation of the set of d predictors $X_i \in \mathbb{F}_q^d$. If we define $X \in \mathbb{F}_q^{n \times (d+1)}$ with the i 'th row being $[1 \ X_i]$, OLS

Algorithm 8: SYMMETRIC GENERALISED EIGENVALUES

Input : Symmetric $A \in \mathbb{F}_q^{n \times n}$; Symmetric, positive semi-definite $B \in \mathbb{F}_q^{n \times n}$;
 $\epsilon > 0$

Output: Verification the helper sent solutions to the generalised eigenvalue problem $AV = BVD$ to precision ϵ

Verifier

Choose $x \in_R \mathbb{F}$
Stream in A, B and compute $F_x(A), F_x(B)$
Pick $T > \frac{q^2 n^4}{\epsilon}$ and send to P

Run

CHOLESKY DECOMPOSITION(B) $\rightarrow L$
INVERT MATRIX(L) $\rightarrow L^{-1}$
MATRIX MULTIPLICATION(L^{-1}, A) $\rightarrow L^{-1}A$
MATRIX MULTIPLICATION($L^{-1}A, (L^{-1})^T$) $\rightarrow L^{-1}AL^{-T}$
SYMMETRIC EIGENSOLVER($L^{-1}AL^{-T}$) $\rightarrow V', D$
MATRIX MULTIPLICATION($(L^T)^{-1}, V'$) $\rightarrow V$

involves finding the linear relationship $y = X\beta + \epsilon$ with $\epsilon \in \mathbb{R}^n$ and $\beta \in \mathbb{R}^{d+1}$ minimizing $\sum_{i=1}^n \epsilon_i^2$.

The OLS problem is solved with the Moore-Penrose pseudo inverse, $(X^T X)^{-1} X^T$. The optimal β is then $(X^T X)^{-1} X^T y$ [Planitz, 1979].

Theorem 9. *There is an annotated streaming $(\max\{(d+1)^2, (d+1)n\} \log(q), \log(q))$ -protocol for OLS for $X \in \mathbb{F}_q^{n \times (d+1)}$, $y \in \mathbb{F}_q^n$ as above.*

Proof. The algorithm requires two applications of our matrix multiplication protocol to check that the β provided satisfies $(X^T X)\beta = X^T y$. This avoids explicitly inverting $X^T X$. Using Theorem 1, the instantiation costs for $X^T y$ are $((d+1)n, 1)$ and for $(X^T X)\beta$ are $((d+1)^2, 1)$. Our total communication and space costs for OLS will therefore be $O(\max\{(d+1)^2, (d+1)n\} \log(q))$ and $O(\log(q))$ respectively. As $X^T X$, X^T and y are in \mathbb{F}_q , we can find a $\beta \in \mathbb{F}_q$ and perform an exact equality check. \square

3.3.2 Principal Component Analysis

Given a large data matrix $S \in \mathbb{F}_q^{n \times d}$ where S_{ij} represents the j th observation of the i th variable, PCA finds the principal components of the data, which can be used for dimensionality reduction or classification. PCA maximises the variation captured by successive vectors in \mathbb{R}^n , i.e. $\text{var}(v_1^T S) \geq \text{var}(v_2^T S) \geq \dots \geq \text{var}(v_n^T S)$, with each $\lambda_i = \text{var}(v_i^T S)$ maximised with respect to $\lambda_j, \forall j < i$. These v_i are the principal components and we can

Algorithm 9: ORDINARY LEAST SQUARES

Input : $X \in \mathbb{F}_q^{n \times (d+1)}$, $y \in \mathbb{F}_q^n$
Output: Verification the helper sent $\beta \in \mathbb{F}_q^n$ with $X\beta = y$

Verifier
 | Choose $x \in_R \mathbb{F}_q$
 | Stream in A and y , compute $F_x(A)$, $F_x(y)$ (treat y as a $n \times 1$ matrix).

Run
 | GRAMIAN MATRIX(X) $\rightarrow F_x(X^T X)$
 | MATRIX MULTIPLICATION(X, y) $\rightarrow F_x^{\text{vec}}(X^T y)$
 | MATRIX MULTIPLICATION($X^T X, \beta$) $\rightarrow F_x^{\text{vec}}((X^T X)\beta)$

Verifier
Check
 | $F_x^{\text{vec}}(X^T y) = F_x^{\text{vec}}((X^T X)\beta)$
end

perform dimensionality reduction on S by choosing the k columns of $V = [v_1, \dots, v_n]$ corresponding to the k largest λ_i and forming $V_{1..k}^T S = S' \in \mathbb{F}_q^{n \times k}$. This is equivalent to finding vectors corresponding to approximate eigenvalues of the covariance matrix of S .

Definition 7. For a data set $S \in \mathbb{F}^{n \times d}$, the (estimated) **Covariance Matrix** of S is

$$\begin{aligned} \text{Cov}(S)_{ij} &= \frac{1}{n-1} \text{Cov}(S_i^\downarrow, S_j^\downarrow) \\ &= \frac{1}{n-1} \mathbb{E} \left[\left(S_i^\downarrow - \mathbb{E} [S_i^\downarrow] \right) \left(S_j^\downarrow - \mathbb{E} [S_j^\downarrow] \right) \right] \end{aligned}$$

Note the $\frac{1}{n-1}$ comes from us eliminating the bias from sampling the n points of S_i^\downarrow , for each $i \in [d]$.

The covariance matrix is the matrix with each element $\text{Cov}(S)_{ij}$ being the covariance of the i th column of S and the j th column of S . It aims to capture the variability of two random variables, i.e. if large values of S_i^\downarrow correspond to large values of S_j^\downarrow , then the covariance will be positive, likewise if these values tend to show the opposite behaviour (large values of S_i^\downarrow correspond to small values of S_j^\downarrow) then the covariance will be negative. This can be useful in statistical analysis for summarising trends in data sets.

In our setting, the equation in Definition 7 doesn't allow us to stream S and compute the fingerprint. However, by rewriting the formula, and using the earlier trick of getting the prover to resend the matrix in a 'nice' order, we can efficiently compute the fingerprint of $\text{Cov}(S)$ so we can use it in our protocols. Using outer product, we can write the covariance matrix as [Park and Park, 2018], $\text{Cov}(S) = (S - \bar{S})^T (S - \bar{S})$. There is some abuse of

notation here, \bar{S} is a $n \times d$ matrix with each row element being the mean of the respective column of S .

Lemma 4. *There is a $(nd \log(q), \log(q))$ -protocol for computing the covariance matrix of $S \in \mathbb{F}^{n,d}$.*

Proof. Using the primitives of Section 3.2, Algorithm 10 shows how we can produce the fingerprint of the (scaled) covariance matrix using

$$F_x((n-1) \text{Cov}(S)) = \sum_{i=0}^d F_x^{\text{vec}} \left(\left(S_i^\downarrow - \bar{S} \right)^T \left(S_i^\downarrow - \bar{S} \right) \right)$$

Note we scale the covariance matrix to keep it in the field \mathbb{F}_q . We will use the GRAMIAN MATRIX protocol on $S_i^\downarrow - \bar{S}$, and can compute $F_x \left(S_i^\downarrow - \bar{S} \right)$ by finding $F_x \left(S_i^\downarrow \right)$ and $F_x \left(\bar{S} \right)$. $F_x \left(\bar{S} \right)$ can be found in the initial stream as

$$\begin{aligned} F_x \left(\bar{S} \right) &= \sum_{i=0}^{n-1} \sum_{j=0}^{d-1} \left[\frac{x^{in+j} \sum_{k=0}^{n-1} S_{kj}}{n} \right] \\ &= \sum_{i=0}^{n-1} \sum_{j=0}^{d-1} \sum_{k=0}^{n-1} \left[\frac{x^{in} x^j S_{kj}}{n} \right] \\ &= \left[\sum_{i=0}^{n-1} \frac{x^{in}}{n} \right] \sum_{j=0}^{d-1} \sum_{k=0}^{n-1} [x^j S_{kj}] \\ &= \left[\frac{1-x^{n^2}}{(1-x)n} \right] \sum_{j=0}^{d-1} \sum_{k=0}^{n-1} [x^j S_{kj}] \end{aligned}$$

For an integer x , $\frac{1-x^{n^2}}{(1-x)}$ will be an integer, hence remain in the field \mathbb{F}_q . However, we will need to scale by a factor of n . Our algorithm therefore runs as follows:

Algorithm 10: COVARIANCE FINGERPRINT

Input : $S \in \mathbb{F}_q^{n \times d}$

Output: $f_x(n^2(n-1)\text{Cov}(S))$

Verifier

| Choose $x \in_R \mathbb{F}$

| Stream in S , compute $F_x(S)$ and $F_x(n\bar{S})$

Run

| GRAMIAN MATRIX($nF_x(S) - F_x(n\bar{S})$) $\rightarrow n^2(n-1)\text{Cov}(S)$

□

Using the prior protocols to check matrix multiplication and approximate eigenvec-

tors (Sections 3.2.2 and 3.2.3), we can check PCA results to any desired precision $\epsilon > 0$ with the costs stated in the following theorem.

Theorem 10. *Given $S \in \mathbb{F}_q^{n \times d}$ and $\epsilon > 0$, we can verify that PCA has been done to the desired precision using a $\left(dn \log \left(\frac{qn^3 d^{\frac{3}{2}} \|S\|_F}{\epsilon} \right), \log \left(\frac{qn^3 d^{\frac{3}{2}} \|S\|_F}{\epsilon} \right) \right)$ -protocol.*

Proof. We generate the fingerprint of the covariance matrix using the $(dn \log(qn^3), \log(qn^3))$ -protocol presented in Algorithm 10.

We use the approximate eigenvalue check (Section 3.2.3) with scaling factor $T = O\left(\frac{d^{\frac{3}{2}} n^3 \|\text{Cov}(S)\|_F}{\epsilon}\right)$ to ensure that V has the necessary properties, i.e. is almost orthogonal, and each claimed eigenvector acts on the covariance matrix as an approximate stretch. Each principal component v_i corresponds to $\text{var}(\hat{v}_i^T S) = \hat{\lambda}_i$ and $|\text{var}(\hat{v}_i^T S) - T\hat{\lambda}_i| \leq T\epsilon$, allowing us to reduce the dimensionality of S , confident of how much variance we are removing with each principal component. Soundness and correctness then follow from the invoked protocols.

Algorithm 11: PRINCIPAL COMPONENT ANALYSIS

Input : $S \in \mathbb{F}_q^{n \times d}$

Output: Verification V are the principal components of S with direction v_i describing $\frac{\lambda_i}{\sum_{j=1}^d \lambda_j}$ of the variance in $V^T S$.

Run

| COVARIANCE FINGERPRINT(S) $\rightarrow F_x(n^2(n-1) \text{Cov}(S))$

Prover

| Send $n^2(n-1) \text{Cov}(S)$

Verifier

| Compute $n^2(n-1) \|\text{Cov}(S)\|_F$ and send $T = O\left(d^{\frac{3}{2}} n^3 \|\text{Cov}(S)\|_F\right)$

Run

| SYMMETRIC EIGENSOLVER($n^2(n-1) \text{Cov}(S), \epsilon$) $\rightarrow (V, D)$

□

3.3.3 Fisher Linear Discriminant Analysis

In Linear Discriminant Analysis (LDA), we again have a large data matrix $S \in \mathbb{F}_q^{n \times d}$, with S_{ij} representing the j th observation of the i th variable, however, we also have a classification for each observation, ω_m for $m \in [k]$, where we have k classes. The aim is to do dimensionality reduction, as in PCA, while **simultaneously** maximizing the class discrim-

inatory information between the classes in the reduced dimension.

If we have k classes, we wish to transform S to a new set $S' \in \mathbb{F}_q^{n \times (k-1)}$, i.e. $S' = W^T S$ where $W \in \mathbb{F}_q^{d \times (k-1)}$ is a matrix projecting S to S' and S' has the maximum **Fisher linear discriminant** $J(W) = \frac{\det(W^T S_B W)}{\det(W^T S_W W)}$ where we have

$$\textbf{Within-class scatter } S_W = \sum_{i=1}^k \text{Cov}(S_i)$$

$$\textbf{Between-class scatter } S_B = \sum_{i=1}^k n_i (\mu_i - \mu)(\mu_i - \mu)^T$$

We first treat the two-class case, then generalize to k classes.

Two Classes

To find $w \in \mathbb{F}_q^d$ we split S into two matrices, $S_1 \in \mathbb{F}_q^{n_1 \times d}$ holding the observations in class 1 with average $\mu_1 \in \mathbb{F}_q^{n_1}$, and $S_2 \in \mathbb{F}_q^{n_2 \times d}$ for the observations in class 2 with average $\mu_2 \in \mathbb{F}_q^{n_2}$. Then our two scatter matrices are:

$$S_W = \text{Cov}(S_1) + \text{Cov}(S_2) \text{ and } S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

To ensure that we can easily represent elements in the finite field, we actually find the (scaled-up) matrices $(n_1 n_2) S_W$ and $(n_1 n_2) S_B$. We wish to maximise the function $J(w) = \frac{w^T S_B w}{w^T S_W w}$. As such, these rescalings do not affect the result. Using the KKT conditions [Kleinberg and Tardos, 2006], we shift this to solving $S_B w = \lambda S_W w$, for some λ .

Theorem 11. *There is a $(dn \log(qn^6), \log(qn^6))$ -protocol for verifying LDA with 2 classes on $S \in \mathbb{F}_q^{n \times d}$.*

Proof. We can simplify $S_B w = \lambda S_W w$ for 2 classes, since for all vectors $v \in \mathbb{R}^d$,

$$S_B v = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T v$$

As $(\mu_1 - \mu_2)^T v$ is just a scalar, $S_B v$ simply scales $(\mu_1 - \mu_2)$. Hence we get that $S_B w = \lambda S_W w$ is equivalent to $\alpha(\mu_1 - \mu_2) = \lambda S_W w$. To verify we have been sent the correct w , all we need do is check that $S_W w = (\mu_1 - \mu_2)$, and our result will lie in the (scaled-up) field, since all the scalars and vectors involved do. The memory space is $O(\log(qn^6))$, where we have $\log(qn^6)$ due to the scaling of the scatter matrices by $N = n_1^2(n_1 - 1)n_2^2(n_2 - 1)$. \square

Algorithm 12: TWO CLASS LDA

Input : $S \in \mathbb{F}_q^{n \times d}$

Output: Verification that $w \in \mathbb{F}_q^d$ maximises $J(w)$.

Verifier

| While streaming S compute $F_x^{\text{vec}}(\mu_1)$, $F_x^{\text{vec}}(\mu_2)$ and

Run

| COVARIANCE FINGERPRINT(S_1) $\rightarrow F_x(n_1^2(n_1 - 1) \text{Cov}(S_1))$

| COVARIANCE FINGERPRINT(S_2) $\rightarrow F_x(n_2^2(n_2 - 1) \text{Cov}(S_2))$

Verifier

| Compute $F_x(NS_W) = n_2^2(n_2 - 1)F_x(n_1^2(n_1 - 1) \text{Cov}(S_1))$
 $+ n_1^2(n_1 - 1)F_x(n_2^2(n_2 - 1) \text{Cov}(S_2))$

Run

| MATRIX MULTIPLICATION(NS_W, w) $\rightarrow F_x^{\text{vec}}(NS_W w)$

Verifier

Check

| $F_x^{\text{vec}}(NS_W w) = NF_x^{\text{vec}}(\mu_1) - NF_x^{\text{vec}}(\mu_2)$

end

k Classes

With k classes, we now need to find a matrix $W \in \mathbb{F}_q^{d \times k-1}$. We still want to maximise the ratio between the between-class and within-class scatter, $J(W)$. The simplification to an eigenvalue problem is more complex here, but from Devijver and Kittler [1982] we see it reduces to finding (at most) $k - 1$ eigenvectors with non-zero real eigenvalues of the matrix $S_W^{-1}S_B$.

Consider $A, B \in \mathbb{F}_q^{n \times n}$, with A symmetric, and B symmetric positive semi-definite (psd) the task is to find $V, D \in \mathbb{R}^{n \times n}$ such that $AV = BVD$ where $A = S_B$ and $B = S_W$. S_W is the sum of positive semi-definite matrices, and as such is positive semi-definite, this allows us to use the Symmetric Generalised Eigenvalue Method.

Theorem 12. *There is a $\left(nd \log \left(\frac{q^3 n^{3k+4}}{\epsilon}\right), \log \left(\frac{q^3 n^{3k+4}}{\epsilon}\right)\right)$ -protocol for verifying LDA with k classes on $S \in \mathbb{F}_q^{n \times d}$ to precision $\epsilon > 0$.*

Proof. We stream in and fingerprint S , which then allows us to receive in the data class by class to build up the scatter matrices, with this we then use the method in Section 3.2.6 to find \hat{V} and \hat{D} with error ϵ . This is a $\left(nd \log \left(\frac{q^3 n^{3k+4}}{\epsilon}\right), \log \left(\frac{q^3 n^{3k+4}}{\epsilon}\right)\right)$ -protocol, note however that we are only interested in columns of \hat{V} corresponding to non-zero eigenvalues. We can use these get our $r \leq k - 1$ vectors to form W and compute $W^T S$ using the matrix multiplication protocol to get the desired result. Algorithm 13 shows the processes of the

Algorithm 13: K CLASS LDA

Input : $S \in \mathbb{F}_q^{n \times d}$, with k classes $S_k \in \mathbb{F}_q^{n_k \times d}$ and $\sum_{i=1}^k N_i = n$, $\epsilon > 0$

Output: Verification that $W \in \mathbb{F}_q^{d \times k-1}$ maximises $J(W)$ to our precision.

Verifier

| While streaming S compute $F_x^{\text{vec}}(F_x^{\text{vec}}(\mu_1), \dots, F_x^{\text{vec}}(\mu_k))$,
| $F_x^{\text{vec}}(F_{x^d}^{\text{vec}}(\mu_1), \dots, F_{x^d}^{\text{vec}}(\mu_k))$ as well as μ and

Run

| COVARIANCE FINGERPRINT(S_1) $\rightarrow F_x(n_1^2(n_1 - 1) \text{Cov}(S_1))$
| \vdots
| COVARIANCE FINGERPRINT(S_k) $\rightarrow F_x(n_k^2(n_k - 1) \text{Cov}(S_k))$

Verifier

| Compute a running fingerprint $F_x(NS_W) = \sum_{i=0}^{k-1} NF_x(\text{Cov}(S_1))$ for
| $N = \prod_{i=0}^{k-1} n_i^2(n_i - 1)$.

Prover

| Send $\hat{\mu}_i$ for $i \in [k]$

Verifier

| Compute $F_x(S_B) = \sum_{i=0}^{k-1} n_i F_{x^d}^{\text{vec}}(\hat{\mu}_i - \mu) F_x^{\text{vec}}(\hat{\mu}_i - \mu)$

Check

| $F_x^{\text{vec}}(F_x^{\text{vec}}(\hat{\mu}_1), \dots, F_x^{\text{vec}}(\hat{\mu}_k)) = F_x^{\text{vec}}(F_x^{\text{vec}}(\mu_1), \dots, F_x^{\text{vec}}(\mu_k))$
| $F_x^{\text{vec}}(F_{x^d}^{\text{vec}}(\hat{\mu}_1), \dots, F_{x^d}^{\text{vec}}(\hat{\mu}_k)) = F_x^{\text{vec}}(F_{x^d}^{\text{vec}}(\mu_1), \dots, F_{x^d}^{\text{vec}}(\mu_k))$

end

Run

| SYMMETRIC GENERALISED EIGENVALUES(NS_W, w) $\rightarrow F_x(V), F_x(D)$

Prover

| Send the $r < k - 1$ largest non-zero eigenvectors, W , and their eigenvalues
| Then send the remaining eigenvectors, and their eigenvalues

Run

| MATRIX MULTIPLICATION(W^T, S) $\rightarrow F_x(W^T S)$

Verifier**Check**

| The sent messages from the prover are consistent with $F_x(V), F_x(D)$

end

algorithm in detail. The scaling factor we use will be scaled n^{3k} to find S_W , then $q^3 n^4$ to run SYMMETRIC GENERAL EIGENVALUES. □

3.4 Practical Results

We performed a series of experiments with a field size of $q = 2^{31} - 1$, using implementations in C of our protocols for matrix multiplication and eigen-decomposition. We used a machine with an Intel i7 processor and 16 GB of memory. We worked on matrices up to 1000 by 1000, as this allowed us to get a distinguishing result between the verifier running the computation and the verifier using our checking protocols. We used the GNU Scientific Library for C, using *gsl_blas_dgemm* for matrix multiplication and *gsl_eigen_symm* for the eigen-decomposition. The verification was done on the result produced by the self-computation. We created dense random matrices with entries drawn uniformly. Note that our protocols do not depend on any structure in the matrices, so uniform random data suffices to test them. Where we require symmetric matrices, we set $A_{ij} = A_{ji} \forall i < j$.

Our verification protocols were built as discussed above, with the time taken to find the fingerprints of all relevant matrices and check the necessary bounds being the verification time seen in Figure 3. We tested the verification protocols against a variety of perturbed matrices, i.e. we added a error matrix to the solution with values greater than the error allowable in the protocol, and all of these were successfully rejected. The memory costs for our verification, even in the case of $A \in \mathbb{F}_q^{1000 \times 1000}$ was only a few hundred bytes, a significant reduction over self computation, which would be in the order of several megabytes. The communication cost is the cost of sending the matrix, simply the size of the matrix multiplied by a small constant.

Consequently, the main aim of our experiments is to quantify the potential timing gains of verification over self-computation.

For our matrix multiplication protocol (Figure 3.1), we consider the number of scalar multiplications required to find the result of multiplying $A \in \mathbb{F}_q^{k \times n}$ and $B \in \mathbb{F}_q^{n \times k}$, that is, k^2n . When we plot the time taken against k^2n , we see the expected linear relationship between self-computation and time taken. The verifier needs to compute the fingerprint of A , B , which will involve nk multiplications, and AB , which involves k^2 multiplications. This agrees with the observation in the graph; that the self-computation time is linear, and the verification time is sublinear.

In eigen-decomposition (Figure 3.2), we see the expected cubic increases in running time for self-computation, with a significantly lower increase for the verification. The verification running time depends primarily on the $O(n^2)$ cost of computing the needed fingerprints, as above, and as such, we see the asymptotically shallower lower curve.

In the next chapter, we will examine a lot more the practicality of the different properties of a streaming interactive proof, such as prover overhead and streaming cost.

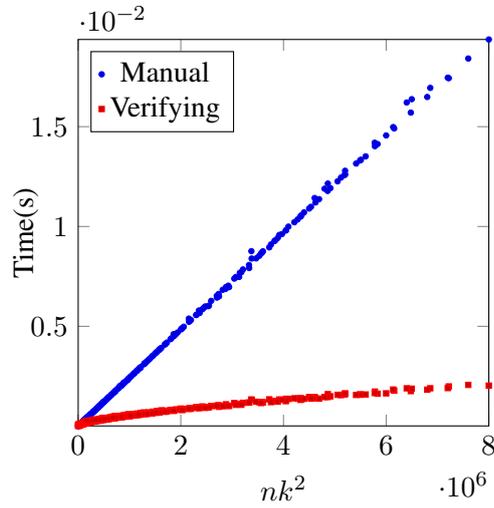


Figure 3.1: Matrix Multiplication of $A \in \mathbb{F}_q^{k \times n}$ and $B \in \mathbb{F}_q^{n \times k}$ and 500 runs per test, for $n, k \in \{10i : i \in [20]\}$. The blue marks represent the manual multiplication time, the red marks represent the time the verifier spends checking.

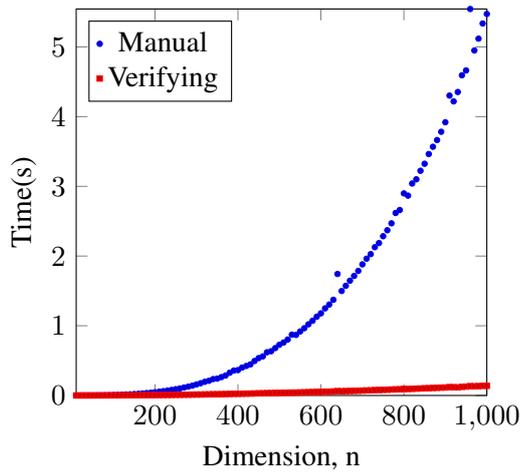


Figure 3.2: Eigen-decomposition of symmetric $A \in \mathbb{F}_q^{n \times n}$ with $\epsilon = 0.01$ and 150 runs per test.

Chapter 4

Achieving Optimal Costs through Interactivity

This chapter is formed from the 2019 ISAAC paper “Efficient Interactive Proofs for Linear Algebra.” written by Graham Cormode and Chris Hickey.

4.1 Motivation

The pitch for cloud computing services is that they allow us to outsource the effort to store and compute over our data. The ability to gain cheap access to both powerful computing and storage resources makes this a compelling offer. Streaming interactive proofs allow us to explore the questions of trust and reliability in this setting. The question we wish to answer here is simple: What costs do we want to reduce in a streaming interactive proof?

In this work, we focus on studying primitive computations within linear algebra — a core set of tools with applications across engineering, data analysis and machine learning. We make four main contributions:

- We consider protocols with variable interactivity for inner product and matrix multiplication and present lightweight tunable verification protocols for these problems. We also produce an entirely new protocol for vector-matrix-vector multiplication.
- Our protocols allow us to trade off computational effort and communication size against the number of rounds of interaction. We show it is often desirable to have fewer rounds of interaction.
- We optimize the costs for the cloud, and show that the protocols impose a computational overhead that is typically much smaller than the cost of the computation itself.

- Our experimental study confirms our analysis, and demonstrates that the absolute cost is minimal, with the client’s cost significantly less than performing the computation independently.

4.1.1 Quantifying Costs

We use the definitions from Section 2, namely Definition 3. Our principle concern now will be the costs described in Section 2.4.

Definition 8. *For a function f we say that there is a d -round (h, v) -protocol if there is a valid protocol for f with*

- **Verifier Memory s** — *Verifier uses $O(s)$ working memory.*
- **Communication h** — *The total communication between the two parties is $O(h)$. Note that we do not include the cost of sending the claimed solution in this cost.*
- **Interactivity d** — *at most $2d$ messages sent between H and V .*

Furthermore, we quantify the computational costs by

- **Verifier Streaming Cost** — *The work during the initial stream.*
- **Verifier Checking Computation** — *The work for the interactive stage.*
- **Helper Overhead** — *The additional work outside of solving the problem.*

Problem Statement. We seek optimal or near optimal verification protocols for core linear algebra operations. The canonical (and previously studied) example is the multiplication of two matrices $A \in \mathbb{F}_q^{k \times n}$, $B \in \mathbb{F}_q^{n \times k'}$, where \mathbb{F}_q is the finite field of integers modulo q , for some prime $q > M^2 n$, where $M = \max_{i,j}(A_{ij}, B_{ij})$ or chosen sufficiently large to not incur overflows. Our protocols work on any prime size finite field, consistent with prior work. This allows computation over fixed precision rational numbers, with appropriate scaling. For ease of exposition, we assume in this paper that $n = k = k'$, although all our algorithms work with rectangular matrices. The resulting matrix AB is assumed to be too large for the verifier to conveniently store, and so our aim is for the helper to allow the verifier to compute a *fingerprint* of AB [Rabin, 1981], defined formally in Section 2.5.1, that can be used to check the helper’s claimed answer.

4.1.2 Prior Work

As discussed in Chapter 2, interactive proofs were introduced in the 1980s, primarily as a tool for reasoning about computational complexity. The notion that interactive proofs could be a *practical* tool for verifying outsourced computation was advocated by Goldwasser et al. [2008]. This paper introduced the powerful GKR (or ‘muggles’) protocol for verifying arbitrary computations specified as arithmetic circuits. Several papers, to cite but a few, have aimed to optimize the costs of the GKR protocol [Cormode et al., 2012; Vu et al., 2013; Thaler, 2013], or to provide systems for verifying general purpose computation under a variety of computational or cryptographic models [Setty et al., 2012a,b; Parno et al., 2016]. The latter of which tackle large classes of problems using *arguments*, which consider a computationally bounded prover. We consider only proofs as we can achieve highly efficient protocols without requiring restriction on the prover, or use of cryptographic assumptions; the prover runs efficiently in the honest case, and we remain secure against unbounded provers. Furthermore, some costs associated with such verification still remain high, such as requiring a large amount of pre-processing on the part of the helper, which can only be amortized over a large number of invocations. For the common and highly symmetric algebraic computations we work with in this paper, it is beneficial to build a specialised protocol.

Other work has considered engineering protocols for specific problems that are more lightweight, and so trade generality for greater practicality. The motivation is that some primitives are sufficiently ubiquitous that having special purpose protocols will outweigh the effort to design them. An early example of this is given by Freivalds’ algorithm for verifying matrix multiplication [Freivalds, 1979]. This and similar algorithms unfortunately don’t directly work for verifiers that can’t store the entire input. This line of work was initiated for problems arising in the context of data stream processing, such as frequency analysis of vectors derived from streams [Chakrabarti et al., 2009]. Follow-up work addressed problems on graph data [Cormode et al., 2011; Chakrabarti and Ghosh, 2019; Bera et al., 2020; Chakrabarti et al., 2020a,b], data mining [Daruki et al., 2015] and machine learning (Chapter 3, Sabater et al.).

These papers tend to consider either the non-interactive case (minimizing the number of rounds), or have a poly-logarithmic number of rounds (minimizing the total communication). We wish to explore the middle ground. For example, Cormode et al. [2011] introduces an interactive inner product protocol which can accommodate a variable number of rounds. This work assumes that setting the number of rounds to be $\log(n)$ will be universally optimal, an assumption we reassess in this work. Similarly, in Thaler [2013] the matrix multiplication protocol takes place over $O(\log(n))$ rounds. Our observation is that the pragmatic choice may fall between these extremes of non-interactive and highly inter-

Method	Proof Length	Verifier Space	Rounds	Prover Time	Verifier Stream Time and Check Time
This Work	$O(ld)$	$O(l + d)$	$d - 1$	$O(n \log(l))$	$O(nld) O(ld)$
Binary sum check; Cormode et al. [2011]	$O(\log(n))$	$O(\log(n))$	$\log(n)$	$O(n)$	$O(n \log(n))$ $O(\log(n))$
FFT & LDEs; Cormode et al. [2012]	$O(n^{1-a})$	$O(n^a)$	1	$O(n \log(n))$	$O(n)$ $O(\log(n))$

Table 4.1: Different SIPs for Inner Product with $u, v \in \mathbb{F}_q^n$, with $n = l^d$ and $a \in [0, 1]$, we exclude the $\log(q)$ factors for storing field elements. Note that Cormode et al. [2011] introduce the sum-check protocol for l and d , but do the prover analysis for $l = 2$, and do not achieve this work’s prover run time for $l \neq 2$.

active. Taking into account latency and round-trip time between participants, the preferred setting might be a constant number of rounds, which yields a communication cost which is a small polynomial in the input size, but which is not significantly higher in absolute terms from the minimal poly-logarithmic cost.

We summarize the current state of the art for the problems of computing inner product (Table 4.1) and matrix multiplication (Table 4.2), and show the results we obtain here for comparison. For both tables, we see that our results with variable interactivity achieve the same asymptotic costs at the extremes, aside from the non-interactive matrix multiplication protocol. This allows us to maintain the state of the art lower bounds of these problems, whilst maintaining the desired flexibility.

Lastly, we comment again that our results are restricted to the information-theoretically secure model of Interactive Proofs, and are separate from recent results in the computational (cryptographic) security model [Ben-Sasson et al., 2016; Canetti et al., 2018]. Furthermore discussion of interactivity of proofs in the computational setting become irrelevant when one considers the Fiat-Shamir protocol, which allows an interactive proof to be replaced by a non-interactive protocol by sacrificing information security (hence shifting to *arguments* over *proofs* [Fiat and Shamir, 1986]).

4.1.3 Contributions and outline

Our main contribution is an investigation into the time-optimal number of rounds for a variety of protocols. We adapt and improve protocols for inner product and matrix multiplication, as well as introducing an entirely new protocol for vector-matrix-vector multipli-

Method	Proof Length	Verifier Space	Rounds	Prover Time	Verifier Stream Time and Check Time
This Work	$O(ld)$	$O(l + d)$	d	$O(n^2)$	$O(n^2ld) O(ld)$
Binary sum check; Thaler [2013]	$O(\log(n))$	$O(\log(n))$	$\log(n) + 1$	$O(n^2)$	$O(n^2 \log(n))$ $O(\log(n))$
Fingerprints; Chapter 3	$O(n^2)$	$O(1)$	1	$O(1)$	$O(n^2) O(n^2)$

Table 4.2: Different SIPs for Matrix Multiplication with $A, B \in \mathbb{F}_q^{n \times n}$ and $n = ld$, we exclude the $\log(q)$ factors for storing field elements.

ation. We then perform experiments in order to evaluate the time component of each stage of interaction.

We begin in Section 4.2 by re-evaluating how to measure the communication cost of a protocol, and propose to combine the competing factors of latency and bandwidth into a total time cost. This motivates generalized protocols that take a variable number of rounds, where we can pick a parameter setting to minimize the total completion time.

In Section 4.3 we build on previous protocols of Cormode et al. [2011, 2012] to construct novel efficient *variable round* protocols for core linear algebra operations. We begin by revisiting variable round protocols for inner product. We leverage these to obtain new protocols for matrix multiplication and vector-matrix-vector multiplication (which does not appear to have been studied previously) with similar asymptotic costs.

In Section 4.4, we thoroughly analyse the practical computation costs of the resulting protocols, and compare to existing verification methods. We perform a series of experiments to back up our claims, and draw conclusions on what the most appropriate parameter setting for streaming interactive proofs in practice. We show that it can be preferable to use fewer rounds, despite some apparently higher costs.

4.2 How Much Interaction Do We Want?

Prior work has sought to find ‘optimal’ protocols which minimize the total communication cost. This is achieved by *increasing* the number of rounds of interaction, with the effect of driving down the amount of communication in each round. The minimum communication is typically attained when the number of rounds is polylogarithmic [Cormode et al., 2012]. The non-interactive case represents another extreme in this regard, requiring a single message from the helper to verifier. This allows the parties to work asynchronously at the cost of larger total communication.

In this section we argue that the right approach is neither the non-interactive case *nor* the highly-interactive case. Rather, we argue that a compromise of ‘moderately interactive proofs’ can yield better results. To do so we consider the overall time required to process the proof.

The key observation is that the time to process a proof depends not just on the amount of communication, but also the number of rounds. In the protocols from Table 4.1 and 4.2, each round cannot commence until the previous round completes, hence we incur a time penalty as a function of the *latency* between the two communicating parties. The duration of a round depends on the bandwidth between them. Thus, we aim to combine number of rounds and message size into a single intuitive quantity based on bandwidth and latency that captures the total wall-clock time cost of the protocol.

For matrix multiplication, the variable round protocols summarized in Table 4.2 spread the verification over d rounds, and have a total communication cost proportional to $dn^{1/d}$. Hence, we write the time to perform the communication of the protocol as $T = 2d\mathcal{L} + \frac{2dn^{1/d}\log(|\mathbb{F}|)}{\mathcal{B}}$, where latency (\mathcal{L}) is measured in seconds, and bandwidth (\mathcal{B}) in bits per second. This expression emerges due to the $2d$ changes in direction over the protocol, and considering a protocol that sends a total of $2dn^{1/d}$ field elements (from the analysis in Section 4.3.2).

We measured the cost using typical values of \mathcal{L} and \mathcal{B} observed on a university campus network, where the ‘ping’ time to common cloud service providers (Google, Amazon, Microsoft) is of the order of 20ms, and the bandwidth is around 100Mbps. From the above equation for T we see that, for a field size $|\mathbb{F}|$, the value of $2n^{1/d}d\log(|\mathbb{F}|)/\mathcal{B}$ is dominated by $2d\mathcal{L}$ for even small d under such parameter settings. Hence, we should prefer fewer rounds as latency increases. Figure 4.1 shows the number of rounds which minimizes the communication time as a function of the size of the input. We observe that the answer is a small constant, at most just two or three rounds, even for the largest input sizes, corresponding to exabytes of data.

4.3 Protocols for Linear Algebra Primitives

Using the previously discussed primitives for SIPs, we show how they have been used in inner product [Cormode et al., 2012]. We then use this to construct a new variable round method for matrix multiplication, and extend it to achieve a novel vector-matrix-vector multiplication protocol.

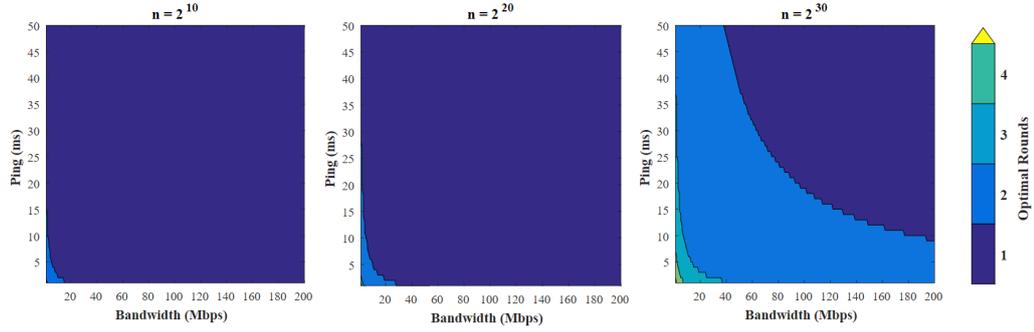


Figure 4.1: Optimal number of rounds for matrix multiplication of various sizes when considering only communication time, with a field size $q = O(n^3)$.

4.3.1 Inner Product

Given two vectors $a, b \in \mathbb{F}_q^n$, the verifier wishes to receive $a^T b \in \mathbb{F}_q$ from the helper. We give a straightforward generalization of the analysis of a protocol in Cormode et al. [2011], as an application of sum-check on the LDEs of a and b . This variable round protocol has costs detailed below.

Theorem 13. *Given $a, b \in \mathbb{F}_q^n$, there is a $(d - 1)$ -round $(ld, l + d)$ -protocol with $n = l^d$ for verifying $a^T b$ with helper computation time $O\left(\frac{n \log(n)}{d}\right)$, verifier overhead $O(nld)$, and checking cost $O(ld)$.*

The analysis from Cormode et al. [2011] sets $l = 2$ and $d = \log(n)$, and the computational cost for the verifier is $O(\log(n))$ while the cost for the helper is $O(n \log(n))$. For general l and d these costs become $O(ld)$ and $O(nld)$ for the verifier and helper respectively.

In Cormode et al. [2012] it is shown how the helper's cost can be reduced to $O(n \log(n))$ for $d = 2$ and $l = \sqrt{n}$ using the Discrete Fast Fourier Transform to make a fast non-interactive protocol. We extend this for arbitrary d and l , and show how by combining with sum-check we can keep the helper's computation low, proving Theorem 13.

Lemma 5. *Given $a, b \in \mathbb{F}_q^n$ the sum*

$$a^T b = \sum_{k_0=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \tilde{f}_a(k_0, \dots, k_{d-1}) \tilde{f}_b(k_0, \dots, k_{d-1}) \quad (4.1)$$

can be verified using a $(d - 1)$ -round $(ld, l + d)$ -protocol with helper computation time $O\left(\frac{n \log(n)}{d}\right)$, and verifier computation time $O(ld)$, overhead time $O(nld)$.

Proof. First, set

$$g(k_0, \dots, k_{d-1}) = \tilde{f}_a(k_0, \dots, k_{d-1}) \tilde{f}_b(k_0, \dots, k_{d-1}).$$

$g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ is a degree $2l$ polynomial in each variable. Now, consider round $j + 1$ of the sum-check protocol, where the helper is required to send

$$g_j(x) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_d=0}^{l-1} g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_d).$$

Here, g is degree $2l$ polynomial, sent to V as a set $G_j^\Sigma = \{(g_j(x), x) : x \in [2l]\}$. To compute this set we have H find the individual summands as

$$G_j = \left\{ (g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1}), x) : x \in [2l], k_{j+1}, \dots, k_{d-1} \in [l] \right\}.$$

Naive computation of all the values in G_j takes time $O(nd)$ each, for a total cost per round of $O(nl^{d-j}d)$. However, instead of computing the LDE at l^{d-j} points with cost $O(ld)$ we can sum l^{d-j} convolutions of length $2l$ vectors to obtain the same result.

In order to prove this claim, we consider the computation of $a^T a$ (also referred to as F_2). The general case of $a^T b$ follows the same steps but the notation quickly becomes (even more) cumbersome. So, given a vector $a \in \mathbb{F}_q^n$, we want to find $\sum_{i=0}^{n-1} a_i^2$. This is equivalent to finding the inner product of a with itself.

Consider a $d - 1$ round protocol for the F_2 problem on $a \in \mathbb{F}_q^n$. We have $n = l^d$, and so for each round of interaction the helper sends

$$g_j(x) = \sum_{k_{j+1}=1}^l \cdots \sum_{k_{d-1}=1}^l \tilde{f}_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})^2,$$

where the input is reshaped as the d -dimensional $A \in \mathbb{F}^{l \times l \times \dots \times l}$. There are $d - 1$ such polynomials to send over the course of the protocol, and each one has degree $2l - 1$.

Round 1. Consider first the opening round

$$g_0(x) = \sum_{k_1=1}^l \cdots \sum_{k_{d-1}=1}^l \tilde{f}_A(x, k_1, \dots, k_{d-1})^2.$$

This can be found by materializing the set of values $G_0 = \left\{ (\tilde{f}_A(x, k_1, \dots, k_d), x) : x \in [2l], k_1, \dots, k_{d-1} \in [l] \right\}$, and then summing over k_1, \dots, k_d to obtain G_0^Σ .

For the first half of the G_0^Σ , the computation is closely linked to the original input,

and so we can simply compute the partial sums

$$\sum_{k_1=1}^l \cdots \sum_{k_{d-1}=1}^l \tilde{f}_A(x, k_1, \dots, k_{d-1})^2.$$

These sums partition the input, so the total time is $O(n)$ to obtain the values for all $x \in [l]$.

However, for x values in the range $l + 1 \dots 2l$, we need to evaluate the LDE at locations not present in the original input. To avoid the higher cost associated with naive computation of all terms, we expand the definition of LDEs:

$$\begin{aligned} \tilde{f}_A(k_0, \dots, k_{d-1}) &= \sum_{p_0=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} A_{p_0 p_1 \dots p_{d-1}} \chi_{p_0 p_1 \dots p_{d-1}}(k_0, \dots, k_{d-1}) \\ \chi_{p_0 p_1 \dots p_{d-1}}(k_0, \dots, k_{d-1}) &= \prod_{j=0}^{d-1} \prod_{i=0, i \neq p_j}^{l-1} \frac{k_j - i}{p_j - i} \end{aligned}$$

In what follows, we can make use of the fact that not all input values contribute to every LDE evaluation needed. We expand as follows:

$$\begin{aligned} g_0(x) &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \tilde{f}_A(x, k_1, \dots, k_{d-1}) \\ &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \sum_{p_1=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} \right. \\ &\quad \left. \left(A_{p_0 p_1 \dots p_{d-1}} \left[\prod_{i=0, i \neq p_0}^{l-1} \frac{x - i}{p_0 - i} \right] \left[\prod_{i=0, i \neq p_1}^{l-1} \frac{k_1 - i}{p_1 - i} \right] \cdots \left[\prod_{i=0, i \neq p_{d-1}}^{l-1} \frac{k_{d-1} - i}{p_{d-1} - i} \right] \right) \right)^2 \\ &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \left[A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{x - i}{p_0 - i} \right] \right)^2 \\ &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \left[\left(A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{1}{p_0 - i} \right) \left(\prod_{i=0}^{l-1} (x - i) \right) \left(\frac{1}{x - p_0} \right) \right] \right)^2 \\ &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\left(\prod_{i=0}^{l-1} (x - i) \right) \sum_{p_0=0}^{l-1} \left[\left(A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{1}{p_0 - i} \right) \left(\frac{1}{x - p_0} \right) \right] \right)^2 \end{aligned}$$

Note in the second step we use that

$$\sum_{p_j=0}^{l-1} \prod_{i=0, i \neq p_j}^{l-1} \frac{k_j - i}{p_j - i} = \begin{cases} 0 & p_j \neq k_j \\ 1 & p_j = k_j \end{cases}$$

We now introduce the helper functions

$$g(p) = \frac{1}{p} \quad h(x) = \prod_{i=1}^l (x - i) \quad q(p) = \prod_{i=0, i \neq p}^{l-1} \frac{1}{p - i} \quad (4.2)$$

to simplify the notation. We define the vectors

$$b_{k_1 \dots k_{d-1}}(p) = \begin{cases} A_{p, k_1 \dots k_{d-1}} q(p) & \text{for } p \in [0, l-1], k_1, \dots, k_{d-1} \in [0, l-1] \\ 0 & \text{for } p \in [l, 2l-1], k_1, \dots, k_{d-1} \in [0, l-1] \end{cases}$$

and use these to rewrite in terms of convolutions

$$\begin{aligned} g_0(x) &:= \sum_{k_1=1}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(h(x) \sum_{p_0=0}^{l-1} [b_{k_1 \dots k_{d-1}}(p_0) g(x - p_0)] \right)^2 \\ &= h(x)^2 \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} (\text{conv}(b_{k_1 \dots k_{d-1}}, g)[x])^2 \\ &= h(x)^2 \left(\sum_{k_2=1}^l \cdots \sum_{k_d=1}^l \text{DFT}^{-1}(\text{DFT}(b_{k_1 \dots k_{d-1}}) \cdot \text{DFT}(g)) \right) [x]^2 \end{aligned}$$

Thus, by precomputing some arrays of values, we reduce the computation to several convolutions that can be evaluated quickly via fast Fourier transform. Observe that this FFT does not need to be computed over the same field as the matrix multiplication: we can choose any suitably large field for which there is an FFT (say, real vectors of size 2^j for some j), and then map the result back into \mathbb{F}_q . Forming $b_{k_1 \dots k_d}(p)$ takes time $O(l^d)$. We have to do $O(l^{d-1})$ convolutions on vectors of length $O(l)$, so each convolution takes time $O(l \log(l))$. Since $\log(l) = \log(n^{\frac{1}{d}})$, we can write the helper's time cost for the first round as $O(\frac{n}{d} \log(n))$.

Round j . Similar rewritings are possible in subsequent rounds. Initially, it may seem that things are more complex for G_j , as each $\tilde{f}_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})$ appears to require full inspection of the input to evaluate at (r_0, \dots, r_{j-1}) . However, we can again define an ancillary array $b_{k_1 \dots k_{d-1}}$ to more easily compute this. In the sum-check protocol

after the helper sends G_0 , it receives r_0 , with which we define the array over $[l]^{d-1}$:

$$A_{r_0 k_1 \dots k_{d-1}}^{(1)} = \sum_{p=0}^{l-1} b_{k_1 \dots k_{d-1}}(p) \prod_{i=0, i \neq p}^{l-1} (r_0 - i)$$

This allows the Helper to form G_1 using the same idea as above, but with $A^{(1)}$ instead of A . Working in terms of $A^{(1)}$ reduces the Helper's cost from $O(l^{d-1}ld)$ for computing the $\tilde{f}_A(r_0, k_1, \dots, k_{d-1})$ for each $k_i \in [l]$ to just $O(l^2)$ when combined with using $b_{k_1 \dots k_{d-1}}$.

In more detail, and with more generality, let us consider the j th round, where we are forming G_j and G_j^Σ . We define

$$A_{r_0, \dots, r_{j-1}, k_j \dots k_{d-1}}^{(j)} = \sum_{p=0}^{l-1} b_{k_j \dots k_{d-1}}(p) \prod_{i=0, i \neq p}^{l-1} (r_{j-1} - i)$$

Then we have the following computation for $x \in [l, 2l - 1]$:

$$\begin{aligned} g_j(x) &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \tilde{f}_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} \left(A_{p_0 \dots p_{d-1}} \left[\prod_{i=0, i \neq p_0}^{l-1} \frac{r_0 - i}{p_0 - i} \right] \cdots \left[\prod_{i=0, i \neq p_{j-1}}^{l-1} \frac{r_{j-1} - i}{p_{j-1} - i} \right] \right. \right. \\ &\quad \left. \left. \left[\prod_{i=0, i \neq p_j}^{l-1} \frac{x - i}{p_j - i} \right] \left[\prod_{i=0, i \neq p_{j+1}}^{l-1} \frac{k_{j+1} - i}{p_{j+1} - i} \right] \cdots \left[\prod_{i=0, i \neq p_{d-1}}^{l-1} \frac{k_{d-1} - 1}{p_{d-1} - 1} \right] \right) \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_j=0}^{l-1} \left[A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{x - i}{p_j - i} \right] \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_j=0}^{l-1} \left[\left(A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{1}{p_j - i} \right) \left(\prod_{i=0}^{l-1} (x - i) \right) \left(\frac{1}{x - p_j} \right) \right] \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\left(\prod_{i=0}^{l-1} (x - i) \right) \sum_{p_j=0}^{l-1} \left[\left(A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{1}{p_j - i} \right) \left(\frac{1}{x - p_j} \right) \right] \right)^2 \end{aligned}$$

We make use of the same set of helper functions specified in equation (4.2), and define the vectors

$$b_{k_{j+1}\dots k_d}(p) = \begin{cases} A_{r_0\dots r_{j-1}p k_{j+1}\dots k_{d-1}}^{(j)} q(p) & \text{for } p \in [0, l-1], k_{j+1}, \dots, k_d \in [0, l-1] \\ 0 & \text{for } p \in [l, 2l-1], k_{j+1}, \dots, k_{d-1} \in [0, l-1] \end{cases}$$

We can now continue to express the computation in terms of convolutions

$$\begin{aligned} g_j(x) &:= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(h(x) \sum_{p_j=0}^{l-1} [b_{k_{j+1}\dots k_{d-1}}(p_j) g(x - p_j)] \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} (h(x) \text{conv}(b_{k_{j+1}\dots k_{d-1}}, g)[x])^2 \\ &= h(x)^2 \left(\sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \text{DFT}^{-1}(\text{DFT}(b_{k_j\dots k_d}) \cdot \text{DFT}(g)) \right) [x]^2 \end{aligned}$$

We can think of $A^{(j)}$ as a shrinking input array, where $A^{(j)} \in \mathbb{F}^{l \times l \times \dots \times l}$ is $d - j$ dimensional, and

$$\begin{aligned} b_{k_{j+1}\dots k_d}(p_j) &= A_{r_1\dots r_{j-1}p_j k_{j+1}\dots k_d}^{(j)} \prod_{i=1, i \neq p_j}^l \frac{1}{p_j - i} \\ A_{r_0, \dots, r_{j-1}, k_j \dots k_{d-1}}^{(j)} &= \sum_{p_{j-1}=0}^{l-1} A_{r_1\dots r_{j-2}p_{j-1}k_j \dots k_d}^{(j-1)} \prod_{i=0, i \neq p_{j-1}}^{l-1} \frac{r_{j-1} - i}{p_{j-1} - i} \end{aligned}$$

Using this formulation, the dominant computation cost in round j will be from the FFT, which involves l^{d-j-1} convolutions of cost $O(\frac{l}{d} \log(n))$ each. Thus the final cost for the round is $O(\frac{l^{d-j}}{d} \log(n))$. The cost of running the entire protocol requires $d - 1$ rounds, making the computational cost for the helper

$$O\left(\sum_{j=0}^{d-2} \frac{l^{d-j}}{d} \log(n)\right) = O\left(n \log(n) \frac{\sum_{j=0}^{d-2} l^{-j}}{d}\right) = O\left(\frac{n \log(n)}{d}\right)$$

since $l \geq 2$. Note that when $d = \log(n)$ and $l = 2$, we achieve $O(n)$ time for the helper. The cost increases with fewer rounds, up to a maximum of $O(n \log n)$ for a constant round protocol.

Cost summary. For the verifier, the checking computation cost is $O(ld)$, which emerges from the d rounds, where in each round the verifier sums the first l elements of G_j^Σ , before

evaluating the LDE of G_j^Σ at r_j , making for a total cost of $O(l)$. The streaming overhead for the verifier involves evaluating the LDE of the input A , for a cost of $O(nld)$. The verifier requires $O(l + d)$ memory to find the LDE of a at $r \in \mathbb{F}^d$. The communication will be $O(ld)$ as we have the helper sending d sets G_j of size $O(l)$. Hence, we summarize the various costs as

Rounds $d - 1$

Communication $O(ld)$

Verifier Memory $O(l + d)$

Helper Computation Time $O\left(\frac{n \log(n)}{d}\right)$

Verifier Time $O(nld)$

Verifier Checking Computation Time $O(ld)$

The total cost of each convolution is $O(l \log(l))$. Summing these l^{d-j} convolutions gives the cost of the j th round for the helper as $O\left(\frac{l^{d-j} \log(n)}{d}\right)$. Summing $\sum_{j=0}^{d-1} l^{d-j}$ over the d rounds gives us our cost of $O\left(\frac{n \log(n)}{d}\right)$. The remaining costs are as in our version of the sum-check protocol (Section 2.5.3). \square

4.3.2 Matrix Multiplication

By combining the power of LDEs with the matrix multiplication methods from Chapter 3, we can create a protocol with only marginally larger costs than inner product.

Theorem 14. *Given two matrices $A, B \in \mathbb{F}_q^{n \times n}$, we can verify the product $AB \in \mathbb{F}_q^{n \times n}$ using a d -round $(ld, l + d)$ -protocol with verifier time $O(n^2ld)$, checking time $O(ld)$ and prover computation time $O(n^2)$ (on top of time to perform matrix multiplication).*

Proof. We make use of the matrix fingerprints from Section 2.5.1, and generate the fingerprint of AB for some $x \in \mathbb{F}_q$ by expressing matrix multiplication as a sum of outer products.

$$F_x(AB) = \sum_{i=0}^{n-1} F_{x^n}^{\text{vec}}(A_i^\downarrow) F_x^{\text{vec}}(B_i^{\rightarrow}) \quad (4.3)$$

where A_i^\downarrow denotes the i th column of A and B_j^{\rightarrow} is the j th row of B . Effectively, the prover is sending a matrix C , claiming $C = AB$, then the protocol has the prover convince the verifier that $F_x(AB) = F_x(C)$, and hence the two matrices are equal.

We also define:

$$A_{\text{col}} = (F_{x^n}^{\text{vec}}(A_1^\downarrow), \dots, F_{x^n}^{\text{vec}}(A_n^\downarrow)) \text{ and } B_{\text{row}} = (F_x^{\text{vec}}(B_1^\rightarrow), \dots, F_x^{\text{vec}}(B_n^\rightarrow)).$$

Our fingerprint $F_x(AB)$ is then given by the inner product of A_{col} and B_{row} . We apply the inner product protocol of Theorem 13, hence we need to show the verifier can evaluate the LDE of the product of these two vectors at a random point,

$$\sum_{k_{d-1}=0}^{l-1} \tilde{f}_{A_{\text{col}}}(r_0, \dots, r_{d-2}, k_{d-1}) \tilde{f}_{B_{\text{row}}}(r_0, \dots, r_{d-2}, k_{d-1}),$$

which we denote as $\Sigma \tilde{f}_{A_{\text{col}}}(r) \tilde{f}_{B_{\text{row}}}(r)$. We can construct this value in the initial stream by storing, for each value of k_{d-1} , $\tilde{f}_{A_{\text{col}}}(r_0, \dots, r_{d-1}, k_{d-1})$ and $\tilde{f}_{B_{\text{row}}}(r_0, \dots, r_{d-1}, k_{d-1})$, which is done in space $O(ld)$ for the verifier. Each of these requires an initial verifier overhead of $O(ld)$ for each of the n^2 elements, then checking requires $O(ld)$ as in Theorem 13. The helper has to fingerprint the matrices to form A_{col} and B_{row} , at a cost of $O(n^2)$. The result follows by using the generated fingerprint to compare to the fingerprint of the claimed result AB (which is provided by the helper in some suitable form, and excluded from the calculation of the protocol costs). \square

Note that the helper is not required to follow any particular algorithm to compute the matrix product AB . Rather, the purpose of the protocol is for the helper to assist the verifier in computing a fingerprint of AB from its component matrices. The time cost of this is much faster: linear in the size of the input.

Fingerprinting versus LDEs. Our protocol in Theorem 14 is stated in terms of fingerprints. In Thaler [2013], a d -round protocol is presented which uses

$$\tilde{f}_{AB}(R_1, R_2) = \sum_{k_0=0}^1 \cdots \sum_{k_{\log(n)-1}=0}^1 \tilde{f}_A(R_1, k) \tilde{f}_B(k, R_2).$$

This uses the inner product definition of matrix multiplication, whilst we use the outer product property of fingerprints. Finding $\tilde{f}_{AB}(R_1, R_2)$ during the initial streaming has cost per update $O(\log(n))$. For our method, we find $\Sigma \tilde{f}_{A_{\text{col}}}(r) \tilde{f}_{B_{\text{row}}}(r)$, which has cost $O(ld)$. In the case $l = 2, d = \log(n)$, we see these two methods are very similar. The methods differ in how we respond to receiving the result, AB . In Thaler [2013], the verifier computes the LDE of AB at a time cost of $O(n^2 ld)$, while our method takes time $\tilde{O}(n^2)$ to process the claimed AB , as we simply fingerprint the result. Thaler's method possesses some other advantages, for example it can chain matrix powers (finding A^m) without the Helper having

to materialize the intermediate matrices. Nevertheless, in data analysis applications, it is often the case that only a single multiplication is required.

4.3.3 Vector-Matrix-Vector Multiplication

Vector-matrix-vector multiplication appears in a number of scenarios. A simple example arises in the context of graph algorithms: suppose that helper wishes to demonstrate that a graph, specified by an adjacency matrix A , is bipartite. Let v be an indicator vector for one part of the graph, then $v^T A v = (\mathbf{1} - v)^T A (\mathbf{1} - v) = 0$ iff v is as claimed. More generally, the helper can show a k colouring of a graph using k vector-matrix-vector multiplications between the adjacency matrix and the k disjoint indicator vectors for the claimed colour classes.

We reduce the problem of vector-matrix-vector multiplication (which yields a single scalar) to inner product computation, after reshaping the data as vectors. Formally, given $u, v \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$, we can compute $u^T A v$ as

$$u^T A v = \sum_{i=1}^n \sum_{j=1}^n u_i A_{ij} v_j = (uv^T)_{\text{vec}} \cdot A_{\text{vec}}$$

$u^T A v$ is equal to computing the inner product of A and uv^T written as length n^2 vectors. Protocols using this form will need to make use of an LDE evaluation of uv^T . We show that this can be built from independent LDE evaluations of each vector.

Lemma 6. *Given $u, v \in \mathbb{F}^n$ and $r \in_R \mathbb{F}^d$, with $n = l^d$*

$$\tilde{f}_{uv^T}(r_0, \dots, r_{2d-1}) = \tilde{f}_u(r_0, \dots, r_{d-1}) \tilde{f}_v(r_d, \dots, r_{2d-1})$$

Proof. We abuse notation a little to treat uv^T as a vector of length n^2 , and we assume that $n = l^d$ (if not, we can pad the vectors with zeros without affecting the asymptotic behaviour). We write $R_1 = (r_0, \dots, r_{d-1})$ and $R_2 = (r_d, \dots, r_{2d-1})$. The proof follows by expanding out expression (2.3) to observe that $\chi_k(r_0 \dots r_{2d-1}) = \chi_{k_0, \dots, k_{d-1}}(R_1) \chi_{k_d, \dots, k_{2d-1}}(R_2)$ and so

$$\begin{aligned} \tilde{f}_{uv^T}(r_0, \dots, r_{2d-1}) &= \sum_{k_0=0}^{l-1} \dots \sum_{k_{2d-1}=0}^{l-1} [(uv^T)_k \chi_k(r)] \\ &= \sum_{i_0=0}^{l-1} \dots \sum_{i_{d-1}=0}^{l-1} \sum_{j_0=0}^{l-1} \dots \sum_{j_{d-1}=0}^{l-1} (u_i v_j) \chi_i(R_1) \chi_j(R_2) \\ &= \tilde{f}_u(R_1) \tilde{f}_v(R_2) \end{aligned}$$

Verifier Tasks		Helper Tasks	
Streams A, B and computes $\Sigma \tilde{f}_{A_{col}}(r) \tilde{f}_{B_{row}}(r)$	α	a	Finds AB
Computes $f_x(AB)$	β_0	b ₀	Sends AB
Sends x	δ		
Checks $\sum_{i=0}^{l-1} (G_0)_i = f_x(AB)$ Computes $\tilde{f}_{G_0}(r_0)$	β	b	Computes G_0
Sends r_0	δ	c	Sends G_0
⋮			⋮
Checks $\sum_{i=0}^{l-1} (G_{d-2})_i = \tilde{f}_{G_{d-3}}(r_{d-3})$ $\tilde{f}_{G_{d-2}}(r_{d-2}) = \Sigma \tilde{f}_{A_{col}}(r) \tilde{f}_{B_{row}}(r)$	β	b	Computes G_{d-2}
		c	Sends G_{d-2}

Figure 4.2: Detailed Matrix Multiplication Protocol

□

The essence of the proof is that we can obtain all the needed cross-terms corresponding to entries of uv^T from the product involving all terms in \tilde{f}_u and all terms in \tilde{f}_v .

We can employ the protocol for inner product using \tilde{f}_A and \tilde{f}_{uv^T} , which we can compute in the streaming phase, as $\tilde{f}_{uv^T} = \tilde{f}_u \tilde{f}_v$ to give us Theorem 15.

Theorem 15. *Given $u, v \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$, we can verify $u^T Av$ using a $(d-1)$ round $(ld, l+d)$ -protocol for $n^2 = l^d$, with helper computation $O\left(\frac{n^2 \log(n)}{d}\right)$, verifier streaming time cost $O(nld)$ and checking time cost $O(l)$.*

4.4 Practical Analysis

To evaluate these protocols in practice, we focus on the core task of matrix multiplication. In order to discuss the time costs associated with execution of our protocols in more detail, we break down the various steps into components as illustrated in Figure 4.2. Here, we use Greek characters to describe the costs for the verifier: the initial streaming overhead ($t[\alpha]$), the checks performed in total in each round ($t[\beta]$), as well as the time to send responses ($t[\delta]$). For the helper, we identify four groups of tasks, denoted by Latin characters: the computation of the matrix product itself ($t[a]$), the communication of this result to the veri-

fier ($t[b_0]$), and the time per round to compute and send the required message ($t[b]$ and $t[c]$ respectively).

Recall our discussion in Section 4.2 on the effects of communication bandwidth and latency on the optimal number of rounds. In our simple model we focused on the tasks most directly involved with communication (the verifier round cost $t[\delta]$ and helper round cost $t[c]$). We implicitly treated the corresponding round computation costs ($t[\beta]$ and $t[b]$) as nil. As the construction and sending of the solution ($t[a]$ and $t[b_0]$) will dominate the first stage of the protocol, we focus our experimental study on measuring values of $t[b]$, $t[\beta_0]$ and $t[\beta]$ to quantify a reasonable estimate for the length of time the interactive phase of the protocol takes with bandwidth \mathcal{B} and latency \mathcal{L} .

We account for the cost required for computation and communication separately to find the total time, T , as follows:

$$T = t[\text{work}] + t[\text{comm}] = (t[\beta_0] + t[\beta] + t[b]) + \left(2d\mathcal{L} + \frac{2dl \log(|\mathbb{F}|)}{\mathcal{B}}\right).$$

T is the total time for the protocol from receiving the answer to producing a conclusion of the veracity of the result. We can omit the verifier’s streaming computation time $t[\alpha]$ from the total protocol run time, as this can be overlapped with the helper’s computation of the true answer, which should always dominate.

In what follows, we instantiate this framework and determine the costs of implementing protocols. These demonstrate that while computation cost for matrix multiplication ($t[a]$) grows superquadratically, the streaming cost ($t[\alpha]$) is linear in the input size n . The dominant cost during the protocol is $t[\beta_0]$, to fingerprint the claimed answer; other computational costs in the protocol are minimal. Factoring in the communication based on real-world latency and bandwidth costs, we conclude that latency dominates, and indeed we prefer to have fewer rounds. In all our experiments, the optimal number of rounds is just 2. Extrapolating to truly enormous values of n suggest that still three rounds would suffice.

4.4.1 Setup

The experiments were performed on a workstation with an Intel Core i7-6700 CPU @ 3.40GHz processor, and 16GB RAM. Our implementations were written in single-threaded C using the GNU Scientific Library with BLAS for the linear algebra, and FFTW3 library for the Fourier Transform. The programs were compiled with GCC 5.4.0 using the `-O3` optimization flag, under Linux (64-bit Ubuntu 16.04), with kernel 4.15.0. Timing was done using the `clock()` function for all readings except $t[\beta]$, which used `getrusage()` as the timings were so small.

For the various tests performed, the matrices and vectors were generated using the

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	12	0.230 ± 0.02	9 ± 2
4	6	0.120 ± 0.01	14 ± 1
8	4	0.099 ± 0.01	35 ± 7
16	3	0.097 ± 0.01	35 ± 7
64	2	0.110 ± 0.01	43 ± 5

(a) $n = 2^{12}$, $t[\beta_0] = 149 \pm 15$ ms

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	16	3.5 ± 0.2	6 ± 1
4	8	2.0 ± 0.1	9 ± 1
16	4	1.6 ± 0.1	46 ± 3
256	2	1.8 ± 0.1	1700 ± 200

(b) $n = 2^{16}$, $t[\beta_0] = 38.0 \pm 6.5$ s

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	18	14.1 ± 0.9	6 ± 1
4	9	8.0 ± 0.5	11 ± 3
8	6	6.3 ± 0.5	30 ± 3
64	3	7.1 ± 0.6	270 ± 30
512	2	7.8 ± 0.7	6400 ± 650

(c) $n = 2^{18}$, $t[\beta_0] = 603 \pm 63$ s

Table 4.3: Interaction phase costs

n	$t[a]$ (s)
2^{10}	0.61 ± 0.06
2^{11}	5.61 ± 0.7
2^{12}	47.9 ± 4.3
2^{13}	403 ± 34

Table 4.4: Matrix Multiplication Timings

`Crand()` function. Note that the work of the protocols is not affected by the data values, so we are not much concerned with how the inputs are chosen. The arithmetic field used was \mathbb{F}_q with $q = 2^{31} - 1$ (larger fields, such as $q = 2^{61} - 1$ or $q = 2^{127} - 1$ could easily be substituted to obtain much lower probability of error, at a small increase in time cost). The work of the verifier and work of the helper were both simulated on the same machine.

4.4.2 Matrix Multiplication Results

Table 4.3 shows the experimental results for the matrix multiplication protocol for matrix sizes ranging from $n = 2^{12}$ to 2^{18} . Note, this means we are tackling matrices with tens of billions of entries. For completeness, we timed BLAS matrix multiplication on our machine for $n = 2^{10}$ to 2^{13} to give an idea of the comparative magnitude of a (Table 4.4), although further results were restricted by machine memory. Due to memory limitations, we tested our algorithms using freshly drawn random values in place of stored values of the required vectors or matrices. This does not affect our ability to compare the data, and allows us to increase the data size beyond that of the machine memory.

The computation cost $t[a]$ grows with the cost of matrix multiplication, which is superquadratic in n , while $t[\alpha]$ grows linearly with the size of the input, which is strictly

Table 4.5: Time taken for interactions (ping 20ms, bandwidth 100Mbps, $|\mathbb{F}| = 2^{31} - 1$)

n	l	d	Latency cost (ms)	Bandwidth cost (ms)
2^{12}	2	12	440	0.014
	4	6	200	0.012
	8	4	120	0.015
	16	3	80	0.020
	64	2	40	0.041
2^{16}	2	16	600	0.019
	4	8	280	0.018
	16	4	120	0.031
	256	2	40	0.163
2^{18}	2	18	680	0.022
	4	9	320	0.020
	8	6	200	0.026
	64	3	80	0.082
	512	2	40	0.328

Table 4.6: Verifier matrix multiplication time (ping 20ms, bandwidth 100Mbps, $|\mathbb{F}| = 2^{31} - 1$).

n	l	d	$t[\text{comm}]$ (s)	$t[\text{work}]$ (s)	T (s)
2^{12}	2	12	0.44	0.149	0.589
	4	6	0.20		0.349
	8	4	0.12		0.269
	16	3	0.08		0.229
	64	2	0.04		0.189
2^{16}	2	16	0.60	38	38.6
	4	8	0.28		38.3
	16	4	0.12		38.1
	256	2	0.04		38.0
2^{18}	2	18	0.68	603	604
	4	9	0.32		603
	8	6	0.20		603
	64	3	0.08		603
	512	2	0.04		603

quadratic in n . Further, the verifier does not need to retain whole matrices in memory, and can compute the needed quantities with a single linear pass over the input.

We next study the helper’s cost across all d rounds to compute the responses in each step of the protocol. Our analysis bounds this total cost as $O(\frac{n \log(n)}{d})$. However, we observe that in our experiments, this quantity tends to decrease as d decreases. We conjecture that while the cost does decrease each round, the amount of data needed to be handled quickly decreases to a point where it is cache resident, and the computation takes a negligible amount of time compared to the data access. Thus, this component of the helper’s time cost is driven by the number of rounds during which the relevant data is still ‘large’, which is greater for larger d .

When we look at the contributory factors to $t[\text{work}]$, we observe that the dominant term is by far $t[\beta_0]$, where the verifier reads through the claimed answer and computes the fingerprint. Thus, arguably, the *computational* cost of any such protocol once the prover finds the answer is dominated by the time the verifier takes to actually inspect the answer: all subsequent checks are minimal in comparison. This justifies our earlier modelling assumption to omit computational costs in our balancing of latency and bandwidth factors.

We now turn to the time due to communication, summarized in Table 4.5. Here, we can clearly see the huge difference of several orders of magnitude between the latency cost, $2d\mathcal{L}$, versus the bandwidth cost, $\frac{2dl \log(|\mathbb{F}|)}{B}$. Note that these timing figures are simulated, based on the average values of latency and the corresponding average bandwidth found when pinging several cloud servers such as Google, Amazon and Microsoft from a university network. The dependencies on both latency and bandwidth are linear. Conse-

quently, if the latency were reduced to 10ms, this would halve the times in the Latency cost column; similarly, if bandwidth were doubled, this would halve the times in the Bandwidth cost column. We observe then that for all but very low bandwidth scenarios, the latency cost will dominate.

Finally, we put these pieces together, and consider the total protocol time from both computation and communication components. We obtain the total time by summing $t[\text{work}]$ and $t[\text{comm}]$, in Table 4.6. These results confirm our earlier models, and the fastest time is achieved with a very small number of rounds. For all values of n tested in these experiments, we see the optimal value of d is 2, the minimally interactive scenario. The trend is such that, because of the sheer domination of latency and $t[\beta_0]$, it is unlikely that more than two or three rounds will ever be needed for even the largest data sets. As n increases, the size of $t[\text{work}]$ grows faster than $t[\text{comm}]$, predominantly due to $t[\beta_0]$. Therefore to minimize the cost of verification one should prefer a small constant number of rounds.

4.5 Concluding Remarks

Our experimental study supports the claim that fewer rounds of interaction are preferable to allow efficient interactive proofs for linear algebra primitives. For large instances in our experiments, the optimal number of rounds is just two. These primitives allow simple implementation of more complex tools such as regression and linear predictors, such as those in Chapter 3. Other primitive operations, such as scalar multiplication and addition, are trivial within this model (since LDE evaluations and fingerprints are linear functions), so these primitives collectively allow a variety of computations to be efficiently verified. Further operators, such as matrix (pseudo)inversion and factorization are rather more involved, not least since they bring questions of numerical precision and representation handled in Chapter 3. Nevertheless, it remains open to show more efficient protocols for other functions, such as matrix exponentiation, and to allow sequences of operations to be easily ‘chained together’ to verify more complex expressions.

Chapter 5

Space-Bounded Commitment Schemes and Spatial Zero Knowledge

This chapter is formed from the as yet unpublished joint work of Graham Cormode, Chris Hickey and Tom Gur.

5.1 Introduction

The previous two chapters cover practical, efficient interactive proofs for common data analysis problems. We shift our focus here, away from a focus on time and efficiency, but towards security.

We look now into the idea of zero knowledge, introduced alongside interactive proofs in Goldwasser et al. [1985]. This is a well studied notion within interactive proofs, and we wish to extend work in this area to the streaming setting. We do this using *commitment schemes*.

A commitment scheme is a cryptographic analogue of sending a locked box. It allows a party to commit to a message, m , using a secret key k , to produce $c = \text{commit}(m, k)$, which can be sent to others without revealing the value of m . The commitment c can be decommitted to using a key k , where $m = \text{decommit}(c, k)$. As a primitive, commitment schemes can be used to form the basis of zero-knowledge proof systems.

We want to build *streaming* zero knowledge proof systems. In order to do this, we will attempt to create commitment schemes where one of the parties is space-bounded. This will let us develop the primitives for spatial zero knowledge. We first produce inefficient protocols for the classic streaming problem of INDEX. We then introduce an improvement to

the commitment scheme allowing for an efficient streaming zero knowledge implementation of the sum-check protocol.

We also discuss the applications of such a protocol. We discuss the class of problems that can be solved by a streaming zero knowledge problem, and compare this to the problems that can be solved by streaming interactive proofs.

5.2 Commitment Schemes

In the standard commitment scheme setting [Blum, 1983], we have two parties, a ‘sender’ and a ‘receiver’. The sender has a message m and wishes to commit this to the receiver by sending $c = \text{commit}(m, k)$. Formally defining commitment schemes involves defining what properties we need a commitment to have. We have two main properties we care about:

Hiding The receiver can not learn the message from just the commitment.

Binding The sender can not decommit to anything but the message.

If a commitment scheme has both of these properties, then we are happy; the sender knows its message is safely hidden when it’s committed, and the receiver knows the sender can’t decommit to an arbitrary message. The precise definition of a commitment scheme delves into three further ideas of hiding/binding security: Perfect, statistical, and computational. Loosely speaking, when applied to the concepts of hiding and binding these mean the following:

- Computational requires the sender or receiver (or both) to run in polynomial time, and effectively uses hardness assumptions to achieve the above properties.
- Statistical security tells us both properties hold almost certainly, in the sense that both parties could potentially cheat and find out about the message, or change the message after commitment, but they’d have to be very lucky.
- Perfect security means that the receiver couldn’t learn anything at all about the message from the commitment, a random guess at the message would be all it could do, and even then it wouldn’t know it was right. In terms of binding, perfect security means that the sender absolutely can not decommit to anything but the message it committed to.

With the intuition down, we present the formal definition of a commitment scheme.

Definition 9. A **commitment scheme** is a pair of functions $\text{Commit} : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$ and $\text{Decommit} : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$, where \mathcal{M} is the set of messages, \mathcal{K} is the set of keys, and \mathcal{C} is the set of possible commitments. Note that $f : \mathbb{N} \rightarrow \mathbb{R}$ is a **negligible function** if $\forall c \in \mathbb{N} \exists N_c$ such that $\forall x > N_c |f(x)| \leq \frac{1}{x^c}$. We formally define hiding and binding as follows:

Hiding For a message $m \in \mathcal{M}$, let $\text{Dist}(m) = \{\text{Commit}(m, k) : k \in_R \mathcal{K}\}$ denote the probability distributions of the commitments to m over all possible choices of $k \in \mathcal{K}$. The commitment scheme is **hiding** if $\forall m_0, m_1 \in \mathcal{M}$ with $m_0 \neq m_1$, $\text{Dist}(m_0)$ and $\text{Dist}(m_1)$ are indistinguishable in the following sense.

For the scheme to be **computationally hiding**, the commitments for two different messages must be indistinguishable to a probabilistic polynomial time receiver. Formally, for any probabilistic polynomial time ‘distinguisher’ D that has input c taken from $\text{Dist}(m_0)$ or $\text{Dist}(m_1)$ and outputs

$$D(c) = \begin{cases} 0 & \text{If } D \text{ decides } c \text{ was a commitment to } m_0 \\ 1 & \text{If } D \text{ decides } c \text{ was a commitment to } m_1 \end{cases}$$

We have

$$\left| \mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_0)) - \mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_1)) \right| \leq \delta(n),$$

for some negligible function $\delta(n)$ with $n = \text{Max}_{k \in \mathcal{K}}(|k|)$. Effectively, we’re saying the probability the distinguisher says c was a commitment to m_0 correctly is about the same as the probability it says c was a commitment to m_1 incorrectly.

In order to achieve a **statistically hiding** commitment scheme, we say the above holds for any computationally unbounded distinguisher. We have a **perfectly hiding** commitment scheme by then also requiring $\delta(n) = 0 \forall n$.

Binding The commitment scheme is **Computationally Binding** if any for any PPT algorithm that generates $(m, m', k, k') \in \mathcal{M}^2 \times \mathcal{K}^2$ with $m \neq m'$ we have

$$\mathbb{P}(\text{Commit}(m, k) = \text{Commit}(m', k')) \leq \delta(n).$$

For some negligible function $\delta(n)$ with $n = \text{Max}_{k \in \mathcal{K}}(|k|)$. Again, intuitively this is saying that the probability that a polynomial time sender can find a pair of keys and messages with the same commitment is very, very small.

As with the hiding property, we achieve **statistically binding** commitment schemes by allowing any computationally unbounded probabilistic algorithm. We get **perfectly binding** commitment schemes by setting $\delta(n) = 0 \forall n$.

An immediate consequence from this definition is the impossibility of simultaneously perfect binding and hiding commitment schemes [Ostrovsky et al., 1992].

Corollary 5. *A time-based commitment scheme, as in Definition 9, can not be both perfectly binding and perfectly hiding.*

Proof. If the commitment scheme is perfectly binding, then $\forall(m, m', k, k') \in \mathcal{M}^2 \times \mathcal{K}^2$ with $m \neq m'$,

$$(\text{Commit}(m, k) \neq \text{Commit}(m', k'))$$

So for each $m \in \mathcal{M}$ and $k \in \mathcal{K}$ there is a unique commitment $c \in \mathcal{C}$. Hence a computationally unbounded receiver, upon receiving a commitment $c \in \mathcal{C}$ could go through each possible combination of $(m, k) \in \mathcal{M} \times \mathcal{K}$ and compute $c' = \text{Commit}(m, k)$ until it finds $c' = c$, and thus the scheme isn't perfectly hiding. \square

And in fact, it can be shown further that we can't have a commitment scheme without a computationally bounded party [Haitner, 2011].

Corollary 6. *Any time-based commitment scheme must be computationally binding or computationally hiding.*

These previous two results specifically refer to time-based commitment schemes. The definition we're working with at the moment is based off computation restrictions on the sender and receiver based on a polynomial run-time. However, these above results do not hold in the space-bounded setting, as we will see.

5.2.1 Space-Bounded Commitment Schemes

In our work, we will be investigating the case when one of the parties is *spatially* bounded. We want to adapt Definition 9 to allow for space bounds, as opposed to time bounds. This will allow us to make commitment schemes that have security in the streaming model, where we're dealing with inputs and messages too big to store.

Definition 10. *Extending Definition 9, we have a pair of functions $\text{Commit} : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$ and $\text{Decommit} : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$, where \mathcal{M} is the set of messages, \mathcal{K} is the set of 'keys', and \mathcal{C} is the set of possible commitments. The restricted receiver must be able to evaluate Commit and Decommit for any key-message pair in space $O(s)$. We define our commitment scheme to be **spatially binding** or **spatially hiding** if it satisfies the following properties:*

Spatially Hiding *For a message $m \in \mathcal{M}$, let $\text{Dist}(m) = \{\text{Commit}(m, k) : k \in_R \mathcal{K}\}$ denote the probability distributions of the commitments to m over all possible choices of $k \in \mathcal{K}$. The scheme is $O(s, \delta)$ -**spatially hiding** if the commitments for two*

different messages must be indistinguishable to a receiver with space $O(s)$ and a security parameter $\delta : \mathbb{N} \rightarrow [0, 1]$. That is, for any ‘distinguisher’ D with space $O(s)$ that has input c taken from $\text{Dist}(m_0)$ or $\text{Dist}(m_1)$ and outputs

$$D(c) = \begin{cases} 0 & \text{If } D \text{ decides } c \text{ was a commitment to } m_0 \\ 1 & \text{If } D \text{ decides } c \text{ was a commitment to } m_1, \end{cases}$$

we have

$$|\mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_0)) - \mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_1))| \leq \delta(n),$$

With $n = \text{Max}_{k \in \mathcal{K}}(|k|)$.

Spatially Binding The commitment scheme is $O(s)$ –**Spatially Binding** if any for any algorithm (representing the sender) using $O(s)$ space that generates $(m, m', k, k') \in \mathcal{M}^2 \times \mathcal{K}^2$ with $m \neq m'$ we have

$$\mathbb{P}(\text{Commit}(m, k) = \text{Commit}(m', k')) \leq \epsilon(n),$$

for $\epsilon(n)$ with $n = \text{Max}_{k \in \mathcal{K}}(|k|)$.

The space required for the commitment scheme, $O(s)$, represents a minimum for the verifier. We will see in our implementations for zero knowledge later, they maintain security dependent on a security parameter κ .

In time-bounded commitment schemes, we saw that a commitment scheme must be computationally binding or hiding. This is not the case for space-bounded commitment schemes. The main reason for this is the change in the way these commitment schemes can be broken. Time-bounded commitment schemes break their computational binding or hiding conditions by repeatedly searching for a solution, and they know when they find a key. On the other hand, for spatially binding or hiding commitment schemes, the malicious party will have to guess the correct key when they see the input, as they’re unable to store enough data to uncover the commitments after the input. This means that there is a fundamental bound on the probability of solving the problem immediately, and in making this sufficiently small, we immediately achieve the statistical spatial binding or hiding condition.

5.2.2 Aside: Negligible functions in the Space-Bounded setting

In computational commitment schemes (i.e. commitment schemes with computational hiding or binding) our definition of negligible function makes a lot of sense. We want to make

sure a polynomial time machine couldn't stumble upon a 'bad' scenario, when

$$|\mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_0)) - \mathbb{P}(D(c) = 0 \mid c \leftarrow \text{Dist}(m_1))|$$

or

$$\text{Commit}(m, k) = \text{Commit}(m', k').$$

Our current definition of negligible functions are based on the idea that the sender/receiver can keep repeating computations until it finds one that works, and so we want to show it's unlikely to successfully find something it shouldn't when restricted to a polynomial number of attempts. When we have a spatial bound, the weakened party has a different strategy now, the security comes from the fact that they can't find something *at the time they see* the large input stream. They can't explicitly repeat computations as they don't have all the input stored to check against.

Take the binding property with a space bounded sender. The sender wants to find $(m, m', k, k') \in \mathcal{M}^2 \times \mathcal{K}^2$ with $\text{Commit}(m, k) = \text{Commit}(m', k')$. The Commit function will be a streaming algorithm, and a meaningful space bound here will stop the sender from storing everything required to run the algorithm in a non-streaming fashion. The situation is that the sender is restricted to trying a handful of (m, k) pairs, and hence we just need to be satisfied that it is unlikely that any of these will be a matching pair.

Our definitions require the spatially binding and spatially hiding properties to hold with 'sufficiently small' probability. This will be dependent on the specific memory usage of the commitment scheme, but as in time-bounded commitment schemes, we will be looking to make this super-polynomially small.

5.3 Motivation: Spatial Zero Knowledge

Commitment schemes are a vital primitive that we can use to build useful cryptographic protocols. One class of these protocols are *zero knowledge proofs*.

5.3.1 Zero Knowledge

In a normal IP (Definition 1) the prover can reveal any amount of information regarding the witness that $x \in L$. Zero knowledge IPs consider trying to keep the witness a secret, and attempting to make the verifier learn nothing more than the fact that $x \in L$. To capture the knowledge gained by the verifier, we define the *view* of the verifier.

Definition 11. *The verifier's view of a protocol consists of the verifier's input x , random choices V_R and received messages. Let $\text{View}_V^{(V,P)}(x)$ denote the view of verifier V with the*

prover P , so if there were k messages m_1, \dots, m_k exchanged in the interactive proof, then

$$\text{View}_V^{(V,P)}(x) = (x, V_R, m_1, \dots, m_k)$$

Now we're ready to define the classical time-constrained verifier zero knowledge.

Definition 12. Given a language L , a **perfect zero knowledge protocol** is a triplet (P, V, S) where (P, V) is an IP, and S is an expected polynomial time probabilistic Turing machine with white box access to V and knowledge that $x \in L$, where for any polynomial time probabilistic Turing machine V^* and $x \in L$

$$\text{View}_V^{(V,H)}(x) = S(x, V^*).$$

The idea here is that the verifier learns no information as the probabilistic polynomial time S , without access to the prover, could simulate the exact same outputs the verifier can with access to the prover. We can additionally define 'statistical' and 'computational' zero knowledge, using similar methods as in Definition 9.

Definition 13. Given a language L , we can further classify zero knowledge protocols for our triplet (P, V, S) as follows.

Statistical Define the following two distributions over all inputs $x \in \mathcal{X}$ (the set of inputs)

$$\text{Dist}_S = \{S(x) : x \in \mathcal{X}\} \quad \text{Dist}_{(P,V)} = \{\text{View}_V^{(V,P)}(x) : x \in \mathcal{X}\}$$

We say the protocol is statistical zero knowledge if Dist_S and $\text{Dist}_{(P,V)}$ are statistically close, i.e. an unbounded probabilistic 'distinguisher' D sampling from Dist_S or $\text{Dist}_{(P,V)}$ can distinguish them with probability at most $\delta(n)$, where n is the size of the input and $\delta(n)$ is a negligible function. We denote this property of statistically close for two distributions A and B by $A \stackrel{\sim}{=} B$.

Computational We say the protocol is computational zero knowledge if Dist_S and $\text{Dist}_{(P,V)}$ are Computationally close, i.e. an **polynomial-time** probabilistic 'distinguisher' D sampling from Dist_S or $\text{Dist}_{(P,V)}$ can distinguish them with probability at most $\delta(n)$, where n is the size of the input and $\delta(n)$ is a negligible function. We denote this property of Computationally close for two distributions A and B by $A \stackrel{\sim}{=}_{\text{comp}} B$.

Commitment schemes allow us to build these zero knowledge protocols with ease (Damgård [1998]). A classic example would be in a public key cryptosystem. In this setting, each user i in the system has two keys, a public key p_i known to everyone, and a personal private key q_i . Each key pair has the property that when we encrypt a message with

$p_i(M) = C$, we can decrypt it with q_i , so $M = q_i(C)$. Furthermore, it is computationally hard (for all intents and purposes - impossible) to discover a pair (M, C) without both p_i and q_i . If user i wants to prove to someone they are indeed user i , they need a zero knowledge protocol, they want to show they know q_i without revealing anything other than that proof of knowledge.

Without commitment schemes, consider the following protocol where the prover, with knowledge of p_i and q_i , wishes to prove to the verifier it knows q_i .

1. The verifier chooses a message M , and sends $C = p_i(M)$.
2. The prover computes $M' = q_i(C)$, and sends M' to the verifier.
3. The verifier checks if $M = M'$, if so, the prover must have q_i .

This might seem okay, but a vital bit of information has been leaked! Consider what the verifier has at the end of the protocol, M , C and M' . This is all good if $M = M'$, but what if the verifier never chose M ? The verifier could choose a random C , send it to the prover, and get back M' , and now has a pair (C, M') . If we tried to define a simulator, it would need to create a pair (C, M') , which we know isn't feasible. This is where we need commitments, and run the protocol as follows:

1. The verifier chooses a message M , and sends $C = p_i(M)$.
2. The prover computes $M' = q_i(C)$, and sends **a commitment to M'** to the verifier.
3. **The verifier sends M to the prover.**
4. **The prover checks $M = M'$ and decommits to M' .**
5. The verifier checks if $M = M'$, if so, the prover must have q_i .

Here we can see, as long as the commitment scheme is computationally hiding the verifier can't learn M' and must first reveal it knows M , alleviating us from the problems above.

Using the analogue that the polynomial time verifier is the receiver, we can create computational zero knowledge protocols using computationally hiding commitment schemes. There is a weaker classification of interactive proofs called *interactive arguments*, where the prover is now assumed to also be a polynomial time machine, and we can thus use computationally binding commitment schemes in this setting as well.

Our next plan will be to define a similar notion of zero knowledge proofs for *streaming interactive proofs*, which we call 'Spatial Zero Knowledge'. It is worth mentioning that streaming interactive proofs can work with some currently existing computational zero knowledge protocols, and the streaming side of things is incidental, as the space constraint isn't an issue. We wish to create protocols that actively use the space bounds on the verifier, and achieve something clearly distinct from computational.

5.3.2 Spatial Zero Knowledge

We first define streaming interactive proofs, which are the space-bounded analogue of the interactive proofs from definition 1. Note that the verifier’s space can be smaller than the size of the input, and in many cases will be. Furthermore, whilst the verifier is no longer restricted to running in polynomial time, this is more so to demonstrate the strength of the security induced by the space bound; in our protocols we aim to have verifiers running in polynomial time.

As with zero knowledge interactive proofs, our concern is that the messages the prover sends will leak information besides $x \in L$ that the verifier couldn’t find out without the prover. To deal with this, we introduce the idea of a simulator, who attempts to generate the view of the verifier when interacting with the prover. The prior definition however implies the verifier keeps the entirety of the messages. We need to adapt this definition for the verifier who can’t store the entire message.

Definition 14. *The **space-bounded verifier’s view** of a protocol for a verifier with space $O(s)$ consists of the verifier’s summaries of the input x , random choices V_R and received messages. Let $\text{View}_V^{(V,P)}(x)$ denote the view of verifier V with the prover P , so if there were k messages in the interactive proof, then*

$$\text{View}_V^{(V,P)}(x) = (S_0(x, V_R), S_1(S_0(x, V_R), m_1), \dots, S_k(S_{k-1}(\dots), m_k)),$$

where $S_0, S_1, \dots, S_k : \mathbb{F}^* \rightarrow \mathbb{F}^s$ are the verifier’s summarising functions.

Intuitively, these can be thought of ‘snapshots’ of the verifier’s memory at the ‘critical points’ of the protocol, i.e. the end of each prover message. Our simulator again has white-box access to the verifier, so knows the verifier’s randomness, as well as the summarising functions. This will let the simulator generate the transcripts.

Definition 15. *Consider language L and a triplet (P, V, S) where (P, V) is a SIP, with space bound $O(s)$ on the verifier, and S is a simulator with white-box access to the verifier’s code, space $O(s)$ and knowledge that $x \in L$. This triplet forms a **spatial zero knowledge protocol** for L if for any $O(s)$ space-bounded V^* , and $x \in L$ we have*

$$\text{View}_{V^*}^{(V^*,P)}(x) \sim_{\text{stat}} S(x, V^*)$$

Note that in the above definition $\text{View}_{V^*}^{(V^*,P)}(x)$ and $S(x, V^*)$ are potentially large sets of data. The simulator is expected to output a stream of data that forms $S(x, V^*)$, in the same way that the prover’s message can be long streams to the verifier.

5.3.3 Why statistically indistinguishable?

The verifier’s view and the simulator’s output are being compared by a $O(s)$ space distinguishing algorithm D . Relating how the distinguisher works in non-spatial zero knowledge, we could have these classifications

Computational Spatial Zero Knowledge, $\text{View}_{V^*}^{(V^*,P)}(x) \sim_{\text{comp}} S(x, V^*)$

The distinguisher, in polynomial time, can’t distinguish between the view and the simulator’s output.

Statistical Spatial Zero Knowledge, $\text{View}_{V^*}^{(V^*,P)}(x) \sim_{\text{stat}} S(x, V^*)$

The distinguisher can’t distinguish the view and simulator’s output except some sufficiently small constant probability.

Perfect Spatial Zero Knowledge, $\text{View}_{V^*}^{(V^*,P)}(x) = S(x, V^*)$

The simulator and view of the verifier are exactly the same.

Looking at these, we see that computational spatial zero knowledge doesn’t make too much sense. We don’t have a time constraint on the verifier during the protocol, so there is no real reason to include one afterwards. Furthermore, the notion of ‘perfect spatial zero knowledge’ is one of contention. We are unsure as it stands if it is a feasible idea in the streaming setting, and this remains an open question.

This leaves us with ‘statistical spatial zero knowledge’ being the only classification that makes sense, which is why in Definition 15 we use \sim_s for the distinguisher.

5.3.4 Achieving Spatial Zero Knowledge Through Linear Space-Bounded Commitments

In a streaming interactive proof, the verifier will receive messages from the prover, challenge these messages, and check the responses. The messages from the prover leak information. We want to avoid this leakage by having the prover commit to the messages.

For the protocols we’re interested in, there are two main types of checks the verifier can do; consistency checks, and value checks. In a consistency check, the verifier is making sure one message is equal to another message. In consistency checks, the verifier doesn’t necessarily care what the messages are. These often come up during intermediate stages of protocols. In a value check, the verifier has a known value, v , and needs to check that this is equal to a message from the prover. Examples of the power of linear commitments in the space bounded setting will be seen as we go on. Fundamentally, a lot of the tools we use in SIPs are linear combinations, such as fingerprinting and LDEs.

In the next section, we build a spatially hiding linear commitment scheme to be able to resolve both of these checks. We will be able to solve linear consistency checks.

If the verifier has the commitment to two different messages, m_1 and m_2 , they can check that the difference between the two commitments is 0 by getting the prover to decommit to $m_2 - m_1$. If they are equal, no information should be leaked; the verifier will only know they were equal. In our protocols, we need to ensure that the consistency checks the verifier does involve only linear combinations of messages from the prover. This is the case in the highly powerful sum-check protocol, which we explore in Section 5.7.

For value checks, the verifier often wants to check that the prover’s message is consistent with a random secret message. The prover will send a series of messages, and the verifier will compute a function of these messages to compare against the secret message, with equality only occurring if the prover is honest. If the function is a linear combination, the verifier will be able to produce the commitment to the secret message, and then can request the prover decommits to the secret message. In order to maintain zero knowledge, we have the verifier also send the expected message to the prover, so as to ensure the prover isn’t being encouraged to decommit to a message previously unknown to the verifier.

Soundness in these protocols comes from the fact the prover doesn’t know the verifier’s secret check. This is maintained in this new setting. The prover commits to its messages, and the verifier stores a secret commitment to the linear combination of messages. The verifier reveals the secret combination so the prover can decommit to the message the verifier has. The binding property of the decommitment scheme stops the prover from maliciously changing the commitment here.

5.4 Preliminaries for Spatial Zero Knowledge

In order to construct our spatially hiding commitment scheme, and build our spatial zero knowledge protocols, we will go through several useful preliminary results. We begin with several more well-known results within interactive proofs, giving us the necessary background to build our protocols. These are in addition to our results in Section 2.5. We then show how these can be used to give intuition behind our commitment protocol.

5.4.1 Polynomial Evaluation Protocol (PEP) [Chakrabarti et al., 2013]

The polynomial evaluation protocol is an interactive proof protocol that allows a verifier with a single random evaluation of a degree- p (for a multivariate polynomial, we say the degree is the sum of the individual maximum degrees of each variable) polynomial $f : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$, with help from a prover with knowledge of f in its entirety, to evaluate f at any other point. For soundness, the protocol requires the random evaluation of the polynomial, $f(r)$, to be unknown to the prover.

First, we'll discuss how polynomials will be sent from the prover to the verifier. If we consider a set $F = \{(x, y) : x \in \mathbb{F}_q^d, y \in \mathbb{F}_q\}$, we can construct the LDE of F to get a polynomial $\text{LDE}(F, \cdot) : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$. This polynomial will be at most degree $|F|$, and $\forall (x, y) \in F, \text{LDE}(F, x) = y$.

If the prover needs to communicate a degree- p polynomial to the verifier, it can send a set of $p + 1$ evaluations of the polynomial. The verifier can use LDEs to evaluate the polynomial at any point. If we had a degree- p polynomial $f : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$, with the verifier holding a secret evaluation $f(r)$ for $r \in_R \mathbb{F}_q^d$, the prover could help the verifier compute $f(x)$ by sending an entire description of f as a set $F = \{(x, f(x)) : x \in \mathbb{F}_q^d\}$. This will be a huge set, $|F| = p^d$, but the verifier will be able to compute $\text{LDE}(F, r) = f(r)$ to check that the polynomial agrees with the random known evaluation, and be able to compute $\text{LDE}(F, x) = f(x)$.

PEP allows us to reduce the communication from $O(p^d \log(q))$ to $O(p \log(q))$. The protocol is detailed in Algorithm 14.

Algorithm 14: POLYNOMIAL EVALUATION PROTOCOL

Input : $f(r)$ for a degree- p polynomial $f : \mathbb{F}_q^d \rightarrow \mathbb{F}$, with $r \in_R \mathbb{F}_q^d$, unknown to the prover, and $x \in \mathbb{F}_q^d$.

Output: $f(x)$

Verifier

Compute the line $\ell : \mathbb{F}_q^d \rightarrow \mathbb{F}_q^d$ satisfying

$$\ell(a) = r \quad \ell(b) = x$$

with $a, b \in_R \mathbb{F}_q^d$, then send ℓ to the prover.

Prover

Sends the *restriction* of f to ℓ :

$$f|_{\ell}(t) = f(\ell(t))$$

as a set $f^{\{\ell\}} = \{(f|_{\ell}(t), t) : t \in [p]\}$.

Verifier

Check $\text{LDE}(f^{\{\ell\}}, a) = f(r)$.

Compute $f(x) = \text{LDE}(f^{\{\ell\}}, b)$.

Theorem 16. *Given a degree- p polynomial $f : \mathbb{F}_q^d \rightarrow \mathbb{F}$, and a verifier with space $O(d \log(q))$ and access to $f(r)$ for $r \in_R \mathbb{F}_q^d$, Algorithm 14 is a valid streaming interactive proof with communication $O(\Delta \log(q))$ and soundness error $\frac{p}{q}$ that allows the verifier to learn $f(x)$ for $x \in \mathbb{F}_q^d$.*

Proof. To show Algorithm 14 is a valid streaming interactive proof, we need to prove

soundness and completeness. Completeness comes from the previously discussed properties of LDEs. If the prover did indeed honestly send the set $f^{\{\ell\}}$, then it will be the case that $\text{LDE}(f^{\{\ell\}}, t) = f|_{\ell}(t)$. The verifier's checks will then correctly succeed with $f(r)$ and uncover $f(x)$.

In order to show soundness, we need to show that if a dishonest prover sent an incorrect $f^{\{\ell\}}$. Say the prover sends the set G , which defines $g(t) = \text{LDE}(G, t)$. Both $f|_{\ell}(t)$ and $g(t)$ will have degree at most p . We also know that $g(t) \neq f|_{\ell}(t)$ for at least one $t \in \mathbb{F}_q$. Due to fundamental theorem of algebra, it is fact the case that $f|_{\ell}$ and g can only agree at p points, the roots of the non-zero polynomial $f|_{\ell}(t) - g(t) = 0$. The verifier will perform the check $g(b) = f(r)$. This will only pass if *by chance* b is a root of $f|_{\ell}(t) - g(t)$. This will happen with probability $\frac{p}{q}$, which we can make arbitrarily small with a sufficiently large field. \square

5.4.2 SIPs for INDEX

The algorithm described for PEP, in conjunction with the streaming nature of LDEs, provides a simple and efficient protocol for INDEX. This algorithm, introduced by Chakrabarti et al. [2013], gives a $(\log(n) \log(q), \log(n) \log(q))$ protocol for INDEX on a stream $A \in \mathbb{F}^n$. The protocol is described by Algorithm 15. The soundness and correctness of the algorithm

Algorithm 15: INDEX

Input : $A \in \mathbb{F}_q^n$, followed by $j \in [n]$

Output: A_j

Verifier

| Streams in A and computes $\text{LDE}(A, r)$ for $r \in_R \mathbb{F}^{\log(n)}$.

j arrives

Run

| POLYNOMIAL EVALUATION PROTOCOL($\text{LDE}(A, r), j$).

follow immediately from Theorem 1. The protocol, however, is not zero knowledge. The verifier receives $\text{LDE}(A, \cdot)_{\ell}$, which it would not be able to construct prior to learning j . The vital thing here is that the verifier only uses this line at two points, to check $\text{LDE}(A, r)$ and to learn $\text{LDE}(A, j)$. In the following sections we will try and construct commitment schemes that let the prover only reveal information about these two points, without sacrificing soundness.

5.5 A Spatially Hiding Commitment Scheme

Our goal in this section is to construct a spatially hiding and statistically binding commitment scheme. Recalling our definitions from Section 5.2, this means we're looking to construct a commitment scheme allowing a spatially unbounded sender to commit to a message m in such a way that a receiver with bounded space can't learn anything about m until decommitment.

5.5.1 The Average Case Hardness of INDEX

We will now show how we build up the first spatially hiding commitment scheme, utilizing the average case hardness of INDEX. The intuition is that a streaming algorithm needs linear space to confidently solve INDEX. The following result confirms this.

Lemma 7. [Rao and Yehudayoff, 2020] *Given a random binary string, x , of length κ , followed by an index $j \in_R [\kappa]$, an algorithm with space s will correctly return x_j with probability at most*

$$\frac{1}{2} + \sqrt{\frac{s \ln(2)}{2\kappa}}.$$

This statement from Rao and Yehudayoff [2020] tells us that for INDEX with a binary string, a streaming algorithms' advantage on knowing x_j is barely more than guessing, for a large κ . For our scheme, our plan will be send a big stream, and hide the message at a random index. We want to prove that the verifier, a space $O(s)$ streaming algorithm can't learn any useful information about the message with high probability.

Lemma 8. *For a verifier with space $O(s)$ who receives $\rho \in \mathbb{F}_q^\kappa$ followed by $j \in [\kappa]$ such that, for $m \in \mathbb{F}_q$,*

$$\rho_i = \begin{cases} m & \text{if } i = j \\ r_i & \text{for } r_i \in_R \mathbb{F}_q, i \neq j \end{cases}$$

the probability the verifier will be able to determine if

$$m \in \mathcal{X} = \{x \in \mathbb{F}_q : x \text{ satisfies some property}\}$$

correctly is at most

$$\mathbb{P}(x \in \mathcal{X} \mid x \in_R \mathbb{F}) + \sqrt{\frac{s \ln 2}{2\kappa}}.$$

Proof. From Corollary 6.15 from Communication Complexity and Applications (Rao and Yehudayoff [2020]), we know that if we have independent random variables, $P = \{P_1, \dots, P_\kappa\}$ and $S(P)$ jointly distributed, then in the average case over a uniformly random $I \in [\kappa]$,

$S(\rho) \in S(\mathbb{P})$, and $\rho_i \in \mathbb{P}_i$,

$$\begin{aligned} \mathbb{P}(P_i = \rho_i) &\stackrel{\epsilon}{\approx} \mathbb{P}\left((P_i = \rho_i | I = i) \text{ and } (S(\mathbb{P}) = S(\rho)) \text{ and } (P_j = \rho_j \ \forall j < i)\right) \\ &= \mathbb{P}(P_i = \rho_i | S(\mathbb{P}) = S(\rho)). \end{aligned}$$

Here $\stackrel{\epsilon}{\approx}$ means statistically close with error $\epsilon \leq \sqrt{\frac{H(S(\rho)) \ln(2)}{2\kappa}}$. Here, $H(S(\rho))$ is the entropy of the message $S(\rho) \in S(\mathbb{P})$. Given that the verifier has space s , the verifier can store $\frac{s}{\log(q)}$ elements from \mathbb{F}_q . The entropy of this message is still simply s , as

$$\begin{aligned} H(S(\rho)) &= - \sum_{S(\rho) \in \mathbb{F}_q^{\frac{s}{\log(q)}}} \mathbb{P}(S(\mathbb{P}) = S(\rho)) \log \left(\frac{1}{\mathbb{P}(S(\mathbb{P}) = S(\rho))} \right) \\ &= q^{\frac{s}{\log(q)}} \left(\frac{1}{q^{\frac{s}{\log(q)}}} \log \left(q^{\frac{s}{\log(q)}} \right) \right) = s \end{aligned}$$

Note that the reason we have ‘ $\forall j < i$ ’ is that the algorithm will have seen the preceding ρ_j prior to storing information about ρ_i (however, if we expand this to $\forall j \neq i$, this would only decrease the probabilities, as further conditioning reduces the entropy).

As $\mathbb{P}(P_i = \rho_i) = \frac{1}{q}$, we know that

$$\begin{aligned} \mathbb{E}(\text{Verifier returns } m \text{ successfully}) &= \mathbb{P}(P_i = m | S(\mathbb{P}) = S(\rho)) \\ &\leq \frac{1}{q} + \sqrt{\frac{s \ln(2)}{2\kappa}}. \end{aligned}$$

Relating this back to our setting, this corresponds to the P_i represent x_i , and $S(\mathbb{P})$ is the verifier’s summary. If we precondition one of the P_i as m for a random $i = j \in_R [\kappa]$ we get the setting in the Lemma for a message m . The above tells us that $\mathbb{P}(x_i = m) \stackrel{\epsilon}{\approx} \mathbb{P}(x_i = m | S() = S(x))$. In other words, the probability that $x_i = m$ is almost exactly the probability that $x_i = m$ given the verifier’s summary is $S(x)$, i.e. the conditioning the verifier’s summary gives barely alters the probability that any one uniformly random element is some specific value.

Now consider some binary property that $m \in \mathbb{F}_q$ could possess. This could be that its odd/even parity, the (modulo 2) sum of its digits, or indeed membership in given set. Let \mathcal{X} be the set of elements in \mathbb{F}_q satisfying this property. We want to find the probability the receiver could determine if $m \in \mathcal{X}$ after engaging in the protocol. Let $V(m)$ be the output of the receiver after engaging in a commitment protocol to m .

$$V(m) = \begin{cases} 1 & \text{The receiver believes } m \in \mathcal{X} \\ 0 & \text{The receiver believes } m \notin \mathcal{X} \end{cases}$$

We want to show that the output of the receiver after engaging in the protocol will not be significantly larger than the probability a random element of \mathbb{F}_q is in \mathcal{X} . Formally, we want to show, for a negligible $\epsilon > 0$, we have:

$$|\Pr(V(m) = 1) - \Pr(x \in \mathcal{X} | x \in_R \mathbb{F})| \leq \epsilon$$

Our previous results in this section considered $\mathcal{X} = \{x \in \mathbb{F} : x = m\}$, i.e., just seeing if the receiver can correctly guess m . Now instead of considering a stream $\rho \in \mathbb{F}_q^\kappa$, we consider dealing with the stream $\rho|_{\mathcal{X}} \in \mathbb{F}_2^\kappa$. This will be the binary string where $(\rho|_{\mathcal{X}})_i = 1$ iff $\rho_i \in \mathcal{X}$. Here, we implicitly assume that the receiver has oracle access to \mathcal{X} to determine membership; if not, this would only make the problem harder. We are now asking if the receiver can solve INDEX on this stream. It is quite immediate that this inherits the same hardness from Lemma 7. \square

5.5.2 The Commitment Scheme

Lemma 8 is what we need to finalise our commitment scheme. Algorithm 16 details the COMMIT stage, and Algorithm 17 covers the DECOMMIT stage. Loosely speaking, the protocol has the sender send a stream ρ with the message hidden at a random index k , which is sent after ρ . The receiver keeps a summary of ρ which it can use to recover the message at ρ_k . In our schemes, the summary is a low degree extension of ρ evaluated at a random point, as the decommitment requires a very small message which can be constructed by a simulator with the same space as the receiver. However, if the zero knowledge aspect is no longer required, a simpler protocol could simply have the sender resend ρ , which the receiver could compare with the previous LDE it kept, and recover ρ_k . The task now is to

Algorithm 16: COMMIT

Input : $m \in \mathbb{F}_q$ with a space $O(s)$ receiver and security parameter κ

Output: Receiver holding $K(m, k)$, a commitment to m hidden at index k

Sender

Sends $\rho \in \mathbb{F}_q^\kappa$ followed by $k \in_R [\kappa]$ with $\rho_i = \begin{cases} r_i \in_R \mathbb{F}_q & i \neq k \\ m & i = k \end{cases}$

Receiver

Stores $K(m, k) = \{\text{LDE}(\rho, r), r, k\}$ for $r \in_R \mathbb{F}_q^{\log(\kappa)}$.

Algorithm 17: DECOMMIT

Input : Receiver with $K(m, k)$, a commitment to m

Output: Receiver with m

Run

| POLYNOMIAL EVALUATION PROTOCOL($\text{LDE}(\rho, r), k$)

prove the following theorem.

Theorem 17. *Algorithm 16 and 17 form a commitment scheme for a message $m \in \mathbb{F}_q$ with some security parameter κ for a verifier of space $O(s)$ that is $(s, \sqrt{\frac{s \ln(2)}{2\kappa}})$ -spatially hiding and statistically binding with probability at most $\frac{\log(\kappa)}{q}$.*

Proof. We need to prove two conditions:

Spatially Hiding Given two commitments for two different messages, a $O(s)$ receiver can not distinguish the two messages with probability $\sqrt{\frac{s \ln(2)}{2\kappa}}$.

Statistically Binding The probability the sender is able to successfully decommit to a message different to the committed message is at most $\frac{\log(\kappa)}{q}$.

For statistically binding we rely on soundness of the polynomial evaluation protocol. The sender has sent a stream ρ followed by an index j . The receiver has kept $\text{LDE}(\rho, r)$ and will be looking to evaluate $\text{LDE}(\rho, j)$. We know from Theorem 16 that the PEP is sound, hence the sender engaging in PEP can only convince the verifier of an incorrect $\text{LDE}(\rho, j)$ with probability $\frac{\log(\kappa)}{q}$.

For spatially hiding, we can use Lemma 8, which tells us that the probability the verifier can learn some property of ρ after seeing the stream and k is only boosted by $\sqrt{\frac{s \ln(2)}{2\kappa}}$. \square

Furthermore, for use in our protocols, we want to show that given knowledge of m , a simulator can simulate this protocol with space $O(s)$. The prover in the commitment scheme requires space $O(\kappa)$, as it needs to store all of ρ . The simulator however knows what line the verifier will query, hence is only required to store $\text{LDE}(\rho, \cdot)|_\ell$, which will require space $O(\log(\kappa))$.

Lemma 9. *The verifier's view in the above commitment scheme for a verifier of space $O(s)$ can be simulated by a simulator with space $O(s)$, with $\log(\kappa) < s$.*

Proof. Here we want to show that for our commitment scheme, the simulator with white box access to the verifier and knowledge of m can produce a transcript of the commitment protocol in space $O(s)$. The simulator can do this as follows:

1. Produce $\rho \in \mathbb{F}^\kappa$ with $\rho_i = \begin{cases} r_i \in_R \mathbb{F}_q & i \neq k \\ m & i = k \end{cases}$

Simultaneously, knowing the $r \in_R \mathbb{F}_q^{\log(\kappa)}$ that the verifier will pick, compute ℓ , the line passing through r and κ when κ is interpreted as an element of $\mathbb{F}_2^{\log_2(\kappa)}$.

2. When the verifier requests $\text{LDE}(\rho, \cdot)|_\ell$ in the PEP decommit stage, send the stored LDE restricted to the line.

This will be done in space $O(\log(\kappa))$, and as $\log(\kappa) < s$ for the verifier to be able to compute $\text{LDE}(\rho, r)$. We know the simulator can store $\text{LDE}(\rho)|_\ell$ in space $O(\log(\kappa))$ as the simulator creates ρ , because it knows ℓ , and so just stores $\log(\kappa)$ random evaluations of $\text{LDE}(\rho)|_\ell$, each of which can be computed in space $O(\log(\kappa))$. \square

5.5.3 Warm-up: Zero Knowledge with a $O(\sqrt{n})$ Verifier

Before we go onto more efficient zero knowledge protocols, we will build a protocol for INDEX with a $O(\sqrt{n})$ space verifier that uses the commitment scheme we just made.

As a brief overview, the protocol works by transforming the vector $A \in \mathbb{F}_q^n$ into a $\sqrt{n} \times \sqrt{n}$ grid in the canonical fashion, with the ultimate aim to find the element at index (j_1, j_2) . The verifier will store a hash of each column, the inner product of the column with a random vector $r \in \mathbb{F}_q^{\sqrt{n}}$, but will only need to use the column corresponding to j_2 .

At the point of receiving (j_1, j_2) , the verifier has $C_{j_2} = \sum_{i=1}^{\sqrt{n}} A_{ij_2} r_i$. The prover sends its claimed $\hat{A}_{j_1 j_2}$, and the verifier reveals $\sqrt{n} - 1$ of its random points; all of r except r_{j_1} . The crux of the protocol now is that the prover can compute

$$m = \sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i$$

and $C_{j_2} - m r_{j_1} = A_{j_1 j_2}$. The prover commits to m , and then the verifier sends $C_{j_2} - \hat{A}_{j_1 j_2} r_{j_1}$, which should equal m . This interaction tells the prover that the verifier does indeed have C_{j_2} for the random vector r , so by decommitting m , the prover isn't leaking information. The protocol is formally defined by Algorithm 18.

Theorem 18. *Algorithm 18 is a spatial zero knowledge protocol for INDEX on $A \in \mathbb{F}^n$ with a verifier of space $O(\sqrt{n} \log(q))$. The communication cost of the protocol is $O(\kappa \log(q))$.*

Proof. We need to prove this protocol satisfies three definitions: Correctness, soundness and spatial zero knowledge. For simplicity, we make the assumption that $A_{j_1 j_2} \neq 0$. If $A_{j_1 j_2} = 0$ the secret value r_i becomes useless, so if the prover sends $\hat{A}_{j_1 j_2} = 0$ the verifier

Algorithm 18: \sqrt{n} ZK-INDEX

Input : $A \in \mathbb{F}_q^n$, followed by $(j_1, j_2) \in [\sqrt{n}] \times [\sqrt{n}]$

Output: $A_{j_1\sqrt{n}+j_2}$

Verifier

| Set $C_j = 0 \forall j \in [\sqrt{n}]$ and choose $r_1, \dots, r_{\sqrt{n}} \in_R \mathbb{F}_q$.

| **For** A_{ij}

| | Set $C_j += A_{ij}r_i$

(j_1, j_2) arrives

Verifier

| Sends r_k for $k \in [\sqrt{n}] \setminus j_1$.

Prover

| Sends $\hat{A}_{j_1j_2}$

Run

| COMMIT $\left(\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i \right)$ hidden at index k in a random string $\rho \in \mathbb{F}_q^\kappa$.

Verifier

| Computes $C_{j_2} - \hat{A}_{j_1j_2} r_{j_1}$ and sends to the prover.

Run

| DECOMMIT $\left(\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i, \rho, k \right)$.

Verifier

| Checks $\frac{C_{j_2} - \rho_k}{r_{j_1}} = \hat{A}_{j_1j_2}$.

can send some $a \in_R \mathbb{F}_q \setminus \{0\}$ and the protocol runs for $A + a$, meaning we're now working with $C_{j_2} + a \sum_{i=1}^{\sqrt{n}} r_i$.

We begin with correctness. The verifier has stored $\sum_{i=1}^n A_{ij} r_i$ for each $j \in [\sqrt{n}]$, when the verifier sees (j_1, j_2) it can ignore all but $C_{j_2} = \sum_{i=1}^n A_{ij_2} r_i$. This is effectively a fingerprint of the column of A containing the index of A that the verifier needs. In order to uncover $A_{j_1 j_2}$ the verifier needs to know

$$\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i.$$

The prover can provide this, the verifier reveals $r_1, \dots, r_{j_1-1}, r_{j_1+1}, \dots, r_{\sqrt{n}}$, and the prover can send $\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i$. In the above protocol, this is done through our commitment scheme.

The prover sends $\hat{A}_{j_1 j_2}$, its claimed value of $A_{j_1 j_2}$, followed by a claimed commitment to $\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i$. If the prover is honest, when it decommits the verifier can check

$$A_{j_1 j_2} r_{j_1} + \sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i = C_{j_2}$$

and this will always pass.

Next we will prove the soundness property. Imagine the prover sends $\hat{A}_{j_1 j_2} \neq A_{j_1 j_2}$, and the verifier computes

$$C_{j_2} - \hat{A}_{j_1, j_2} r_{j_1}.$$

The prover has to send the commitment to $\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i$. This needs to agree with C_{j_2} , which the prover doesn't know. The soundness security comes from the prover not knowing C_{j_2} or r_{j_1} . When committing to $\sum_{\substack{i=1 \\ i \neq j_1}}^m A_{ij_2} r_i$, in order to convince the verifier of an incorrect $\hat{A}_{j_1 j_2}$ the prover would need to know what the verifier computed with $C_{j_2} - \hat{A}_{j_1 j_2} r_{j_1}$. If we're using a statistical binding commitment scheme (Theorem 17) then the prover will not be able to change the message committed to with high probability. This means that when the verifier reveals $C_{j_2} - \hat{A}_{j_1, j_2} r_{j_1}$, the prover knows exactly what the verifier wants to see decommitted, it can't change anything.

The final property to prove is spatial zero knowledge. This comes from the spatially hiding property (Theorem 17) and simulatability (Lemma 9) of the commitment scheme. \square

The protocol described by Algorithm 18 is the first spatial zero knowledge streaming interactive proof. The fundamental crux was the use of INDEX to 'hide' the prover

message, then have the verifier reveal its secret (which is usually required to remain secret for soundness) to convince the prover the sent message doesn't tell the verifier something it shouldn't.

This is the same idea as a commitment in a zero knowledge protocol for a non-streaming interactive proof. The prover commits to a proof, and the verifier can query the commitment at various locations in such a way that it can be sure the proof is correct, but learning no problem specific information, besides what it already secretly knew.

Algorithm 18 required a niche set-up to have the verifier know exactly what the prover will send, and this is not easily achievable in an efficient interactive proof, such as Algorithm 15. Protocols such as these require the verifier to receive several messages in order to build up a value to compare to its initial summary of the input, and hence check the protocol was done correctly.

5.5.4 Making the Scheme Linear

As it stands, this commitment scheme has several useful properties. If the receiver has $\text{Commit}(m, k)$ then due to the linearity of LDEs, it can compute $\alpha \text{Commit}(m, k) = \text{Commit}(\alpha m, k)$ for $\alpha \in \mathbb{F}_q$. We'd like to extend this homogeneity to linearity, i.e. we want the receiver to be able to construct $\text{Commit}(\alpha m_1 + \beta m_2, k)$ for $\alpha, \beta \in \mathbb{F}_q$ using $\text{Commit}(m_1, k)$ and $\text{Commit}(m_2, k)$.

At the moment the reason that this isn't possible is quite simple, the commitment for m_1 will hide it within $\rho^{(1)} \in \mathbb{F}_q^\kappa$, with $\rho_k^{(1)} = m_1$ for some $k \in [\kappa]$, and m_2 is hidden in some $\rho^{(2)} \in \mathbb{F}_q^\kappa$ where $\rho_l^{(2)} = m_2$ for some $l \in [\kappa]$. If $k = l$, we can achieve this linear property, as $(\rho^{(1)} + \rho^{(2)})_k = m_1 + m_2$. The linearity of LDEs tells us that $\text{LDE}(\rho^{(1)}, \cdot) + \text{LDE}(\rho^{(2)}, \cdot) = \text{LDE}(\rho^{(1)} + \rho^{(2)}, \cdot)$, so the instantiation of PEP remains consistent.

However, it's not as simple as hiding all the data at the same index. These commitment schemes are useful when we don't want to decommit to m_1 or m_2 , but want to decommit to $m = m_1 + m_2$. We want to make sure no information about m_1 or m_2 is leaked. This is a problem regarding *simulatability*. For the prior scheme, the simulator knew m , the final message, and there weren't intermediate messages. For linear schemes where the simulator doesn't know m_1 or m_2 , we need more. In this case the simulator's sent ρ won't necessarily be realistic, it might not contain m_1 or m_2 at all.

Say the sender hid m_1 and m_2 in two strings $\rho^{(1)}$ and $\rho^{(2)}$ respectively, at index k , with $m = m_1 + m_2$. The simulator needs to be able to recreate this without knowing m_1 or m_2 . The simulator can commit to two fake messages m_1^* and m_2^* with $m = m_1^* + m_2^*$, but this doesn't solve our problem. In order to be simulatable, the transcript should be indistinguishable from a real transcript. A distinguisher here would be able to notice that $\rho^{(1)}$ from the simulator is entirely random, whilst $\rho^{(1)}$ from the real sender wouldn't be.

The solution to this problem is to add extra randomness to the real sender's $\rho^{(1)}$ and $\rho^{(2)}$.

In Lemma 9 we had the simulator producing ρ by picking

$$\rho_i = \begin{cases} r_i \in_R \mathbb{F}_q & i \neq k \\ m & i = k \end{cases}$$

This is possible because the simulator knows m . When the simulator doesn't know m , the problem here is that if we had a distinguisher as in Definition 15, it could feasibly pick up on the difference between ρ_i and ρ_i^* sent by this simulator with

$$\rho_i^* = r_i \in_R \mathbb{F}_q \forall i$$

We want to allow our simulator to generate a collection of messages that can predominantly be randomly generated, but has sufficient consistency within the messages that the distinguisher can't differentiate.

Our method instead is to split each unknown message m_j into m_j^ρ and m_j^μ with $m_j = m_j^\rho + m_j^\mu$. The idea here is that now m_j^ρ and m_j^μ will be hidden in the random strings $\rho^{(j)}$ and $\mu^{(j)}$ at different indices.

$$\rho_i^{(j)} = \begin{cases} r_i \in_R \mathbb{F}_q & i \neq k_\rho \\ m_j^\rho & i = k_\rho \end{cases}$$

$$\mu_i^{(j)} = \begin{cases} r'_i \in_R \mathbb{F}_q & i \neq k_\mu \\ m_j^\mu & i = k_\mu. \end{cases}$$

This allows the simulator to produce these messages at random. The prover will choose $m_j^\rho \in_R \mathbb{F}$ and set $m_j^\mu = m_j - m_j^\rho$. This means that taken independently $\rho_i^{(j)}$ and $\mu_i^{(j)}$ will appear uniformly random. The fact the distinguisher doesn't know the indices k_ρ and k_μ means that the distinguisher can't recover either m_j^ρ or m_j^μ , and so without these, the messages would appear uniformly random whether they're from the prover or the simulator.

This will be the basis of our commitment scheme. For a set of messages $\{m_1, \dots, m_n\}$ the prover can commit to each of these pairs as $\{m_1^\rho, \dots, m_n^\rho\}$ and $\{m_1^\mu, \dots, m_n^\mu\}$, hiding the ρ messages at the same index k_ρ and the μ messages at k_μ . This means the verifier will end the commitment stage with the linear commitments for the two 'shares' of $\{m_1, \dots, m_n\}$, $\{K(m_1^\rho, k_\rho), \dots, K(m_n^\rho, k_\rho)\}$ and $\{K(m_1^\mu, k_\mu), \dots, K(m_n^\mu, k_\mu)\}$. It is now possible for the verifier and prover to engage in a decommitment protocol for any linear combination, $M = \sum_{i=1}^n \alpha_i m_i$, of the messages. This is done by having the verifier com-

pute the commitments

$$K(M_\rho, k_\rho) = \sum_{i=1}^n \alpha_i K(m_i^\rho, k_\rho) \quad K(M_\mu, k_\mu) = \sum_{i=1}^n \alpha_i K(m_i^\mu, k_\mu),$$

because now the prover can engage in decommitment for each of these and the verifier can learn M_ρ and M_μ and then add them to get M . The exact commitment scheme is detailed in Algorithm 19 and Algorithm 20.

Algorithm 19: LINEAR COMMIT

Input : $m_1, \dots, m_n \in \mathbb{F}_q$ with a space $O(s)$ Receiver, and $M = \sum_{i=1}^n \alpha_i m_i$ for $\alpha_i \in \mathbb{F}$, and security parameter κ .

Output: Receiver holding $K(M_\rho, k_\rho)$ and $K(M_\mu, k_\mu)$, a commitment to M_ρ and M_μ such that $M = M_\rho + M_\mu$, and $K(k_\rho, k_1)$ and $K(k_\mu, k_2)$.

Run

| COMMIT(k_ρ)
| COMMIT(k_μ)

For m_i Run

| COMMIT(m_i^ρ) using index k_ρ .
| COMMIT(m_i^μ) using index k_μ .

Receiver

| Add $K(m_i^\rho, k_\rho)$ to the linear combination $K(M_\rho, k_\rho)$.
| Add $K(m_i^\mu, k_\mu)$ to the linear combination $K(M_\mu, k_\mu)$.

Algorithm 20: LINEAR DECOMMIT

Input : Receiver holding $K(M_\rho, k_\rho)$ and $K(M_\mu, k_\mu)$, a commitment to M_ρ and M_μ such that $M = M_\rho + M_\mu$, and $K(k_\rho, k_1)$ and $K(k_\mu, k_2)$.

Output: Receiver with M

Receiver

| Sends $\alpha_1, \dots, \alpha_n$ and to the prover.

Run

| DECOMMIT($K(k_\rho, k_1)$)
| DECOMMIT($K(k_\mu, k_2)$)
| DECOMMIT($K(M_\rho, k_\rho)$)
| DECOMMIT($K(M_\mu, k_\mu)$)

Receiver

| Computes $M = M_\rho + M_\mu$.

Theorem 19. *Algorithm 19 and 20 form a linear spatially hiding and statistically binding commitment scheme with a $O(s)$ space verifier and allowing a prover to commit to $m_1, \dots, m_n \in \mathbb{F}$ and decommit to $M = \sum_{i=1}^n \alpha_i m_i$ for $\alpha_i \in \mathbb{F}$. The security parameter κ provides a spatial hiding bound of $O\left(\sqrt{\frac{s \ln 2}{2\kappa}}\right)$.*

Proof. We need to prove three properties: spatial hiding, statistically binding and linearity.

Spatial Hiding We want to show that the $O(s)$ receiver will fail to uncover the message with high probability. This comes from the spatially hiding property of Commit, as k_ρ and k_μ will be hidden with probability $O\left(\sqrt{\frac{s \ln 2}{2\kappa}}\right)$. This means the receiver won't be able to uncover m_i^ρ or m_i^μ for any $i \in [n]$. Furthermore, the receiver will get no advantage from repeatedly seeing m_i^ρ and m_i^μ for each i as k_ρ and k_μ are both sent only once, and each message is uniformly randomly distributed without knowing the corresponding other value.

Statistically Binding Again, this property comes largely from the binding property of Commit. The sender can't change the committed message due to the fact it can't change any of the intermediate messages due to the soundness of PEP and INDEX.

Linear As each of the m_i^ρ and m_i^μ are hidden at the same respective index, and we are storing LDEs of each sent $\rho^{(i)}$ and $\mu^{(i)}$, the linearity of LDEs tells us that we can do scalar multiplication and add two commitments to get a new commitment of the correct linear combination of the prior messages. \square

This tells us we do indeed have a linear commitment scheme for receivers of size $O(s)$ and can adjust the size of κ to give us the desired security guarantee. We do not yet have everything we need for this to be useful in our zero knowledge protocols. We still want to show that for a space $O(s)$ simulator with knowledge of M and the linear combination, the simulator can generate the transcript with high probability. This will tell us that our linear commitment scheme is zero knowledge, and the receiver never learns $\{m_1, \dots, m_n\}$.

Theorem 20. *Algorithm 19 and 20 as in Theorem 19 are simulatable by a $O(s)$ simulator with $\log(\kappa) < s$ with knowledge of M and α_i .*

Proof. As in Lemma 9, we know Commit and Deccommit are both simulatable when the message is known, with the simulator using space $O(\log(\kappa))$. We want to show now that when the message is unknown, as long as the simulator runs the protocol with something that produces a correct M , the distinguisher has no significant distinguishing advantage.

If we have $\{\alpha_1, \dots, \alpha_n\}$ and $\{m_1, \dots, m_n\}$ with $M = \sum_{i=1}^n \alpha_i m_i$, we will have the simulator using $\{m_1^*, \dots, m_{n-1}^*\} \in_R \mathbb{F}_q^{n-1}$ with

$$m_n^* = \frac{M - \sum_{i=1}^{n-1} \alpha_i m_i^*}{\alpha_n}$$

The simulator will split each m_i^* into $m_i^{*\rho}$ and $m_i^{*\mu}$, and due to the spatial hiding property of the commitment scheme, without learning k_ρ or k_μ there is no way to confidently recover these intermediate messages.

The simulator will be able to do this in the space required, as each m_i^* can be treated one at a time, with a single running sum of $\sum_{j=1}^i \alpha_j m_j^*$ being kept. As mentioned in Lemma 9, the simulator requires space $O(\log(\kappa))$ for the LDE evaluations of the random strings, and hence we need $\log(\kappa) < s$. \square

5.6 Efficient Spatial Zero Knowledge INDEX

Before we introduce the zero knowledge sum-check protocol, we will build our understanding with a relatively simple explanation as to how we transform a pre-existing protocol for INDEX into a zero knowledge protocol using linear commitments. This protocol is exponentially more space efficient than Algorithm 18.

The following protocol for index is an adaptation of Algorithm 15. We have a verifier with space $O(\log(n))$, receiving a vector of length n , $A \in \mathbb{F}_q^n$, followed by an index j , with the aim to return A_j . This protocol, as it stands, is in no way zero knowledge. The verifier learns the LDE of A restricted to a line of its choosing during the instantiation of PEP.

Our linear commitments allow for this to be avoided. The way we will use it will be creating a spatial zero knowledge version of PEP.

Theorem 21. *Algorithm 21 is a zero knowledge protocol for PEP for a verifier with space $O(s)$ and security parameter κ with $\log(\kappa) < s$ and zero knowledge guarantee of $O\left(\sqrt{\frac{s \ln 2}{2\kappa}}\right)$.*

Proof. We need to show that a simulator with knowledge of $f(x)$ can simulate the below protocol. The crucial similarity with Theorem 20 is that the messages $\{(f|_\ell(t), t) : t \in [p]\}$ are equivalent to $\{m_1, \dots, m_n\}$. We are in exactly the same setting as Theorem 19. The simulator can generate the messages in the same way, the hiding property of the protocol means that with high probability the verifier wouldn't be able to see that the values sent were wrong, and the LDE evaluations will only work for the specific points the verifier knows and needs.

Algorithm 21: ZK-POLYNOMIAL EVALUATION PROTOCOL

Input : $f(r)$ for a degree- p polynomial $f : \mathbb{F}_q^d \rightarrow \mathbb{F}$, with $r \in_R \mathbb{F}_q^d$, unknown to the prover, and $x \in \mathbb{F}_q^d$.

Output: $f(x)$

Prover

| Send $f(x)$ to the verifier.

Verifier

| Compute the line $\ell : \mathbb{F}_q^d \rightarrow \mathbb{F}_q^d$ satisfying

$$\ell(a) = r \qquad \ell(b) = x$$

| with $a, b \in_R \mathbb{F}_q$, then send ℓ to the prover.

Run

| LINEAR COMMIT $\left(\{(f|_{\ell}(t), t) : t \in [p]\} \right)$

Verifier

| Using the linear commitments, construct the commitments of

| LDE $(\{(f|_{\ell}(t), t) : t \in [p]\}, a)$ and LDE $(\{(f|_{\ell}(t), t) : t \in [p]\}, b)$,
| $K(f|_{\ell}(a))$ and $K(f|_{\ell}(b))$.

| Send $(f(r), r)$ to the prover.

Prover

| Check the sent $(f(r), r)$ is consistent with f , and on ℓ .

Run

| LINEAR DECOMMIT $\left(K(f|_{\ell}(a)) \right)$

| LINEAR DECOMMIT $\left(K(f|_{\ell}(b)) \right)$

Verifier

| Check $f(x) = f|_{\ell}(b)$ and $f(r) = f|_{\ell}(a)$.

Furthermore, Algorithm 21 is indeed a correct and sound algorithm. The verifier has $f(r)$ and x , and can create the line ℓ in space $O(\log(n))$. The prover sends a claimed $f(x)$ to start with, and then commits to $\{(f|_{\ell}(t), t) : t \in [p]\}$. This allows the verifier to construct the LDEs at the necessary points without learning any of the intermediate values. Before the prover decommits, the verifier proves it knew the point $f(r)$ on the line beforehand, hence showing that it has won't learn any additional information from the decommitments. The prover and verifier engage in the decommitment protocol so the verifier learns $f|_{\ell}(a) \stackrel{?}{=} f(r)$ and $f|_{\ell}(b) \stackrel{?}{=} f(x)$. The latter check uses the prover's sent $f(x)$, which was sent prior to learning r , so prior to any time the prover could have known the linear combination the verifier would be comparing it against. The binding property of the commitment scheme and soundness of PEP prevent the verifier from altering the decommitment to $f|_{\ell}(b)$, as changing this value would change the value of $f|_{\ell}(a)$, which the prover doesn't know whilst committing. \square

This ZK-PEP allows us to simply swap out the PEP in Algorithm 15 with the above protocol. This change does multiply the communication of the protocol by the security parameter κ , but the verifier can continue to work in space $O(\log(n))$. The algorithm is detailed below.

Algorithm 22: $O(\log(n))$ -ZK INDEX

Input : $A \in \mathbb{F}_q^n$, followed by $j \in [n]$
Output: A_j

Verifier
| Streams in A and computes $\text{LDE}(A, r)$ for $r \in_R \mathbb{F}^{\log(n)}$.

j arrives

Run
| ZK-POLYNOMIAL EVALUATION PROTOCOL($\text{LDE}(A, r), j$).

Theorem 22. *Algorithm 22 is a spatial zero knowledge protocol for the index problem on the string $A \in \mathbb{F}_q^n$ with the verifier having $O(\log(n))$ space, and with $O(\kappa \log(n))$ communication. The probability of information being leaked is at most $\sqrt{\frac{\log(n) \log(2)}{2\kappa}}$.*

Proof. Theorem 21 shows that the ZK-PEP used is zero knowledge, and the above extension where the verifier views A and forms $\text{LDE}(A, r)$ doesn't do anything to break this. The simulator would work in exactly the same way, and any summary the verifier kept, so long as it didn't break the commitment scheme, wouldn't allow the verifier to learn extra information as the commitment scheme is hiding and so the verifier can only learn the values decommitted to, which will be $\text{LDE}(A, r)$ and A_j . The probability the verifier learns information is the probability the verifier breaks the commitment scheme, which Theorem 19 tells us happens with probability at most $\sqrt{\frac{\log(n) \log(2)}{2\kappa}}$. \square

5.7 Spatial Zero Knowledge Sum-Check

The power of linear commitments have been shown in the previous two sections. In Section 5.6, the ability for the prover to commit to messages in such a way that the verifier can achieve a commitment to a *secret* linear combination of said messages allowed us to quite simply make the INDEX protocol of Cormode et al. [2011] achieve spatial zero knowledge.

We now show that we can use linear commitments to produce a spatial zero knowledge protocol for sum-check. We want to produce a protocol that will allow the verifier, given a single evaluation of a polynomial $g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ of degree λ , to compute the sum

of this function over a subset of points. The usual sum-check protocol leaks information of partial sums of the function, however we have the prover simply commit to these values.

Algorithm 23 shows the non-zero knowledge sum check protocol with the checks delayed to the last round. In the standard sum check protocol, the check for $\Sigma(g_j)$ will take place on the j th round. We delay the checks to the end to ultimately simplify the transition to the ZK version.

We see the intermediate functions g_j are leaking the partial sums, which the verifier couldn't compute in space $O(\log(n))$. The equality checks the verifier does at the end is where we will take advantage of linear commitments. In zero knowledge, the verifier will have a commitment to $g_{j-1}(r_{j-1})$ and $\Sigma(g_j)$, and will simply check that the difference of these two commitments is zero, which will prove the consistency between the prover's messages without leaking the partial sums.

Our commitment protocol will involve the prover sending linear commitments to $\{(g_j(x), x) : x \in [\lambda]\}$, and then the verifier can use these to construct $g_j(r_j)$ and $\Sigma(g_j)$, as

$$\begin{aligned} \text{LINEAR COMMIT}(g_j(r_j)) &= \text{LDE} \left(\text{LINEAR COMMIT}(\{(g_j(x), x) : x \in [\lambda]\}), r_j \right) \\ \text{LINEAR COMMIT}(\Sigma(g_j)) &= \sum_{k_j=0}^{l-1} \text{LDE} \left(\text{LINEAR COMMIT}(\{(g_j(x), x) : x \in [\lambda]\}), k_j \right) \end{aligned}$$

The verifier can do these whilst the commitments to g_j arrive, and can then store the $O(d)$ commitments until the end, when the prover can decommit to each one at the necessary points, and the verifier can do the checks. The reason we move the checks to the end is that if the checks were done part way through, this would require decommits, and this would leak the k_ρ and k_μ used before we're finished needing them. There is another issue regarding simulation. The simulator needs to store the decommitment lines for each commitment, which would require space $O(\lambda d)$. The verifier in our protocol has space $O(d)$ so we need to reduce the number of decommitments down to a constant.

Fortunately, this is possible with relative ease. The verifier wants to check that $\Sigma(g_{j+1}) - g_j(r_j) = 0$ for $j \in [d-2]$. This is equivalent to checking that the vector of length $d-2$ of $\Sigma(g_{j+1}) - g_j(r_j)$ has a fingerprint equal to 0. The verifier picks a random $x \in_R \mathbb{F}_q$ and instead of forming the commitment of $\Sigma(g_j)$ and $g_j(r_j)$, it adds them to a running fingerprint, $f_x^{\text{vec}}(g_r)$ and $f_x^{\text{vec}}(\Sigma)$. The prover, not knowing the secret x used in the fingerprint until after everything is committed will only be able to break soundness here with probability $O\left(\frac{d}{q}\right)$. The simulator will know x , and can therefore generate the lines used for decommitment in space $O(d)$. Algorithm 24 shows the zero knowledge sum check protocol.

Theorem 23. *Algorithm 24 is a zero knowledge sum check protocol for a $O(d)$ verifier to*

Algorithm 23: SUM CHECK (delayed check)

Input : Verifier with $g(r)$ for $g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ of degree λ and $r \in_R \mathbb{F}_q^d$
(unknown to the prover).

Output: $G = \sum_{i \in [l]^d} g(i)$

Prover

Computes $g_0 : \mathbb{F}_q \rightarrow \mathbb{F}_q$ with $g_0(x) = \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(x, k_1, \dots, k_{d-1})$.

Sends g_0 as $\{(g_0(x), x) : x \in [\lambda]\}$.

Verifier

Sums $G = \sum_{k_0=0}^{l-1} g_0(k_0)$.

Computes $g_0(r_0)$.

Sends r_0 .

For $j = 1$ **to** $d - 2$

Prover

Sends $g_j : \mathbb{F}_q \rightarrow \mathbb{F}_q$ with

$$g_j(x) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1}).$$

Verifier

Computes $\Sigma(g_j) = \sum_{k_j=0}^{l-1} g_j(k_j)$.

Computes $g_j(r_j)$.

Sends r_j .

Prover

Sends $g_{d-1} : \mathbb{F}_q \rightarrow \mathbb{F}_q$ where $g_{d-1}(x) = g(r_0, \dots, r_{d-2}, x)$.

Verifier

Computes $\Sigma(g_{d-1}) = \sum_{k_{d-1}=0}^{l-1} g_{d-1}(k_{d-1})$.

For $j = 0$ **to** $d - 2$

| Check $g_j(r_j) = \Sigma(g_{j+1})$.

Checks $g_{d-1}(r_{d-1}) = g(r)$.

Algorithm 24: ZK-SUM CHECK

Input : Verifier with $g(r)$ for $g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ of degree λ and $r \in_R \mathbb{F}_q^d$
(unknown to the prover).

Output: $G = \sum_{i \in [l]^d} g(i)$

For $j = 0$ **to** $d - 1$

Prover

Computes $g_j : \mathbb{F}_q \rightarrow \mathbb{F}_q$ with

$$g_j(x) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1}).$$

Run

| LINEAR COMMIT($\{(g_j(x), x) : x \in [\lambda]\}$)

Verifier

Computes $K(\Sigma(g_j))$.

Computes $K(g_j(r_j))$.

If $1 \leq j$

| $f_x^{\text{vec}}(g_r) += K(g_j(r_j))x^j$.

If $j \leq d - 2$

| $f_x^{\text{vec}}(\Sigma) += K(\Sigma(g_j))x^{j+1}$.

Sends r_j .

Verifier

| Sends $g(r)$ and x .

Prover

| Checks $g(r) = g(r_0, \dots, r_{d-1})$.

Run

| LINEAR DECOMMIT($\Sigma(g_0)$)

| LINEAR DECOMMIT($f_x^{\text{vec}}(g_r) - f_x^{\text{vec}}(\Sigma)$).

| LINEAR DECOMMIT($g_{d-1}(r_{d-1})$)

Verifier

| $G = \text{LINEAR DECOMMIT}(\Sigma(g_0))$.

| Check LINEAR DECOMMIT($f_x^{\text{vec}}(g_r) - f_x^{\text{vec}}(\Sigma) = 0$).

| Check LINEAR DECOMMIT($g_{d-1}(r_{d-1}) = g(r)$).

verify the sum $G = \sum_{i \in [l]^d} g(i)$ for a $g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ of degree λ . It has total communication $O(\kappa \lambda d)$ and has a zero knowledge guarantee of $O\left(\sqrt{\frac{s \ln 2}{2\kappa}}\right)$ where $\kappa < d$.

Proof. First, we will handle soundness and correctness. Correctness comes from the correctness of the original sum-check protocol, and of the commitment protocol. The soundness here is slightly more complex, but as in previous uses of linear commitments, the fact the verifier reveals secrets isn't an issue as the commitment scheme is statistically binding.

The total communication comes from the λd commitments, which require the sending of $O(\kappa)$ elements, where κ is the security parameter. The verifier requires space $O(d)$ in order to store its random choices. However, the verifier is required to have space $O(d + \lambda)$ as this is the space the simulator requires. As discussed earlier, the simulator can generate the three decommitments as in Theorem 20, the messages will all be known linear combinations, with the result ' M ' known for each decommitment. The simulator will be using $O(d + \lambda)$. This shows the protocol is indeed a zero knowledge protocol. □

5.8 Future Work

One direction of research here could be on the possibility of *perfect* spatial zero knowledge. At various stages of the design for the protocols above, we would flip-flop between believing we had achieved a perfect ZK-SIP, to believing they couldn't exist. In order to define perfect spatial zero knowledge, it is likely that a much stronger definition would be required regarding the power of the distinguisher in the streaming setting. It remains to be seen what restrictions this might place on the power of perfect ZK-SIPs.

Another natural question that would arise with our results in ZK-SIPs would be to ask what is the limit? As mentioned at the end of Chapter 2, we define **SIP** to be the class of problems that can be verified by a logarithmic-round SIP with a polylog verifier and polylog communication. We discussed the results of Goldwasser et al. [2008] and Cormode et al. [2012], which tells us that log-space uniform **NC** is included in this class.

We conjecture that **ZK-SIP**, the class of problems that can be verified by a spatial zero knowledge logarithmic-round SIP with a polylog verifier and polylog communication, also includes log-space uniform **NC**. The sum-check result, with a polylog κ , strongly suggests this to be the case, with some work on linearization as in Cormode et al. [2012] in this setting required to complete this result. A fairly immediate result on classification of **ZK-SIP** is that we can at least say it does contain the class of problems that can be solved by a polylog-space stream verifier in non-zero knowledge, due to the INDEX result. A further, more interesting, extension is that **ZK-SIP** also extends the class of problems

that can be solved by an annotated data stream with a polylog-space stream verifier, and logarithmic communication. The sum-check result shows us we can solve F_2 with a **ZK-SIP**, but Chakrabarti et al. [2009] tells us that an (s, v) -annotated data stream requires $sv = \Omega(n)$.

Furthermore, we believe much work can be done on interesting classifications of SIPs, similar to the work on IPPs of Berman et al. [2018]. We do believe that **ZK-SIP** \neq **SIP**, and believe work done in zero knowledge, and specifically statistical zero knowledge, would give tools required to show that there are ZK-SIPs for **NP**, potentially even **NEXP**. Our reasoning here for **ZK-SIP** \neq **SIP** stems from the fact that so far we have ZK-SIPs for problems that can be solved with sum-check. Searching for an ‘interesting’ streaming interactive proof that can’t be solved using some form of sum-check is an open question, and a natural next step for investigation.

Chapter 6

Conclusion

6.1 Discussion

This thesis reviewed and developed streaming interactive proof protocols with a focus on efficiency and practicality. We built upon previous work and investigated further the practical side of streaming interactive proofs. It is our hope that this work and work in this field will begin to open up the options for real-world implementations of SIPs, for verification of cloud computing applications.

Chapters 3 and 4 aim to provide lightweight verification options with emphasis on making the work of the prover largely insignificant next to the cost of solving the problem. For a large cloud computing company who could be potentially reluctant to massively increase overheads by introducing a verification option, these improvements will hopefully encourage future implementation.

The advantage of increased efficiency for the prover is two-fold: whilst the cloud can use these improved protocols, for the problems we've dealt with, we can switch the roles, and imagine a cloud server monitoring the work of several smaller computers. The smaller computers act as the provers, and the cloud acts as a verifier, checking the work of the many provers in minimal space.

Chapter 5 looks in a somewhat different direction: we still try to make the protocols efficient for both parties, however we now attempt to add aspects of data privacy. There are two natural approaches to data privacy in this setting: The verifier can be the data owner, trying to analyse the data with the help of the cloud, without the cloud learning anything; or we can have the prover owning the data, trying to prove it satisfies some property to a verifier.

We don't examine the first problem, as it requires a form of encryption known as fully homomorphic encryption [Gentry, 2009; Gentry and Halevi, 2011]. The prover would

receive encrypted data, do the analysis on the encrypted data, and send the proof, which the verifier can decrypt and examine. These encrypted proofs would still fundamentally use the protocols from Chapter 3 and 4, and further improvements would be deeply set within the world of the fully homomorphic encryption research.

We study the prover owning the data in Chapter 5. The verifier sees a stream much larger than the space it owns, and can store some summaries of it. The prover then wishes to prove a property of the stream, without revealing any additional information about the stream. The investigation in the above chapter is largely theoretical, simply to prove the concept of a zero knowledge streaming interactive proof makes sense.

The work of Chapter 5 is currently unpublished, and follows an eighteen month exploration for a proof of concept for spatial zero knowledge. The work continues to lead towards a paper complete with the discussed classifications from Section 5.8.

6.2 Extensions and Future Work

The work on practical SIPs is being continued by several research groups [Thaler, 2013; Daruki et al., 2015; Chakrabarti and Ghosh, 2019]. Our work on practical efficient SIPs brings the ultimate implementation of verification to cloud computing operations closer and closer. The minimal interaction approach shows that further investigation on low interaction (but not constant) SIPs could well yield further efficient protocols.

The main area for future work, however, is by far Chapter 5, as mentioned in Section 5.8. The introduction of zero knowledge interactive proofs in the streaming setting should be of great interest to the theoretical community. Further research of the complexity classification of ZK-SIPs could produce novel and exciting results in the field of communication complexity, using previously unrelatable results from SIPs and zero knowledge.

6.3 Closing Remarks

The aim of this thesis was to examine the practicality and limitations of streaming interactive proofs. We have discussed and analysed in great detail what can be done with low-round SIPs, and how these can be applied for machine learning and data science. With the increased usage of the cloud and outsourced computation, one can hope techniques such as these will find their way into regular use.

Alongside this, the most exciting result of the thesis is the proof of concept of *zero knowledge* streaming interactive proofs. This result, following dozens of failed attempts, merges the fields of zero knowledge and streaming algorithms. Zero knowledge led to many remarkable results for IPs, and could well do the same for the world of SIPs and streaming.

We believe that thorough examination of the classifications of ZK-SIPs, using results from zero knowledge interactive proofs, one can obtain strong results in communication complexity. Our work in this area continues, and the potential for interesting and innovative results is well and truly on the horizon.

Bibliography

- Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, February 2009. doi: 10.1145/1490270.1490272. URL <http://doi.acm.org/10.1145/1490270.1490272>.
- Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of mathematics*, pages 781–793, 2004.
- Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.
- Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.
- Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.
- Galfridus Arturus. *Historia Regum Britanniae*. 1136.
- AWS. Aws agreement. 2020. URL <https://aws.amazon.com/agreement/>.
- László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429. ACM, 1985.
- László Babai and Shlomo Moran. Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2): 254–276, 1988.
- László Babai and Endre Szemerédi. On the complexity of matrix group problems i. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 229–240. IEEE, 1984.

- László Babai, Lance Fortnow, Leonid A Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 21–32, 1991.
- Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. Cryptology ePrint Archive, Report 2016/116, 2016. <https://eprint.iacr.org/2016/116>.
- Suman K Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- Itay Berman, Ron D Rothblum, and Vinod Vaikuntanathan. Zero-knowledge proofs of proximity. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.
- Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-shamir from simpler assumptions. Cryptology ePrint Archive, Report 2018/1004, 2018. <https://eprint.iacr.org/2018/1004>.
- Amit Chakrabarti and Prantar Ghosh. Streaming verification of graph computations via graph structure. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Annotations in data streams. *Automata, Languages and Programming*, pages 222–234, 2009.
- Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. On interactivity in arthur-merlin communication and stream computation. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 20, page 180, 2013.
- Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1786–1802. SIAM, 2020a.

- Amit Chakrabarti, Prantar Ghosh, and Justin Thaler. Streaming verification for graph problems: Optimal tradeoffs and nonlinear sketches. *Leibniz international proceedings in informatics*, 176, 2020b.
- Anne Condon. The complexity of space bounded interactive proof systems. In *Complexity Theory: Current Research*, pages 147–189, 1992.
- Anne Condon and Richard J Lipton. On the complexity of space bounded interactive proofs. In *FOCS*, volume 89, pages 462–467, 1989.
- Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proceedings of the VLDB Endowment*, 5(1):25–36, 2011.
- Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.
- Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.
- Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *School organized by the European Educational Forum*, pages 63–86. Springer, 1998.
- Samira Daruki, Justin Thaler, and Suresh Venkatasubramanian. Streaming verification in data analysis. In *International Symposium on Algorithms and Computation*, pages 715–726. Springer, 2015.
- Pierre A Devijver and Josef Kittler. *Pattern recognition: A statistical approach*. Prentice hall, 1982.
- Uriel Feige and Adi Shamir. Multi-oracle interactive protocols with space bounded verifiers. In *Proceedings. Structure in Complexity Theory Fourth Annual Conference*, pages 158–159. IEEE Computer Society, 1989.
- Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1991.
- Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

- Lance Fortnow and Michael Sipser. Are there interactive protocols for co-np languages? *Information Processing Letters*, 28(5):249–251, 1988.
- Rūsiņš Freivalds. Fast probabilistic algorithms. *Mathematical Foundations of Computer Science 1979*, pages 57–69, 1979.
- Martin Furer, Oded Goldreich, and Yishay Mansour. On completeness and soundness in interactive proof systems, 1989.
- Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.
- Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.
- Yael Gertner, Sampath Kannan, and Mahesh Viswanathan. Np and streaming verifiers, 2002.
- O Goldreich, S Micali, and A Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 174–187, 1986.
- S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, 1985.
- Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 1986.
- Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.
- Iftach Haitner. Foundation of cryptography course. <https://www.cs.tau.ac.il/~iftachh/Courses/FOC/Fall11/>, 2011.
- Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.
- Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

- The New York Times. New short cut found for long math proofs. 1992. URL <https://www.nytimes.com/1992/04/07/science/new-short-cut-found-for-long-math-proofs.html>.
- Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Secure commitment against a powerful adversary. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 437–448. Springer, 1992.
- Kun Il Park and Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer, 2018.
- Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Commun. ACM*, 59(2):103–112, January 2016. ISSN 0001-0782.
- M. Planitz. Inconsistent systems of linear equations. *The Mathematical Gazette*, 63(425): 181–185, 1979. ISSN 00255572. URL <http://0-www.jstor.org.pugwash.lib.warwick.ac.uk/stable/3617890>.
- Michael O Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.
- Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020. doi: 10.1017/9781108671644.
- César Sabater, Aurélien Bellet, and Jan Ramon. Distributed differentially private averaging with improved utility and robustness to malicious parties.
- J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. ISSN 0004-5411. doi: 10.1145/322217.322225. URL <https://doi.org/10.1145/322217.322225>.
- SETI@home. 1999. URL <https://setiathome.berkeley.edu/>.
- Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, volume 1, page 17, 2012a.
- Srinath TV Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium*, pages 253–268, 2012b.
- Adi Shamir. IP=PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

- Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, STOC '13*, page 565–574, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320290. doi: 10.1145/2488608.2488679. URL <https://doi.org/10.1145/2488608.2488679>.
- Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology—CRYPTO 2013*, pages 71–89. Springer, 2013.
- Justin Thaler. *Data Stream Verification*, pages 1–8. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-27848-8. doi: 10.1007/978-3-642-27848-8_798-1. URL https://doi.org/10.1007/978-3-642-27848-8_798-1.
- Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
- Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.
- Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 223–237. IEEE, 2013.
- Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213, 1979.