A Survey of Interactive Verifiable Computing

Utilizing Low-degree Polynomials

Angold Wang[a]

Hong Kong Baptist University, Department of Computer Science, Kowloon Tong, Hong Kong

Abstract. This survey provides a comprehensive examination of verifiable computing, tracing its evolution from foundational complexity theory to modern zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARKs). We explore key developments in interactive proof systems, knowledge complexity, and the application of low-degree polynomials in error detection and verification protocols. The survey delves into essential mathematical frameworks such as the Cook-Levin Theorem, the sum-check protocol, and the GKR protocol, highlighting their roles in enhancing verification efficiency and soundness. By systematically addressing the limitations of traditional NP-based proof systems and then introducing advanced interactive proof mechanisms to overcome them, this work offers an accessible step-by-step introduction for newcomers while providing detailed mathematical analyses for researchers. Ultimately, we synthesize these concepts to elucidate the GKR protocol, which serves as a foundation for contemporary verifiable computing models. This survey not only reviews the historical and theoretical advancements in verifiable computing over the past three decades but also lays the groundwork for understanding recent innovations in the field.

Keywords: Zero-Knowledge · Interactive Proof Systems · Verifiable Computing · Sum-Check Protocol · GKR Protocol · Complexity Theory

1 Introduction

Verifiable computing has been a pivotal area of research since the latter part of the 20th century, rooted in the study of computational complexity [Coo71, Lev73]. Early investigations focused on understanding the inherent difficulty of various computational problems and developing abstractions that illustrate the hardness of these problems and the methods to solve them efficiently [GJ79]. As computing power surged, the focus shifted towards more robust computing models that enable resource-constrained clients to delegate complex computations to powerful servers. Crucially, these models ensure that the server can provide a proof of correctness for the delegated tasks, embodying the essence of verifiable computing [GMR85].

In recent years, the field has witnessed significant advancements [GKR08, BCG+14], particularly in enhancing the efficiency of verifiable computing protocols on both the client and server sides [PHGR13, Gro16, BBB+18, GWC19], these advancements often involve sophisticated mathematical concepts from cryptography. This survey aims to chart the historical development of verifiable computing, starting with fundamental concepts and models, identifying their limitations, and progressing towards more powerful and efficient protocols. To ensure accessibility, each concept introduced is accompanied by one or two minimal examples with detailed descriptions. Additionally, every claim and analysis is

E-mail: f3459026@comp.hkbu.edu.hk (Angold Wang)
[a]Hong Kong Baptist University

supported by rigorous mathematical proofs, with prerequisite mathematical foundations provided to aid newcomers. The survey is structured as follows:

Section 2: Complexity Theory Foundations We begin by exploring the foundational concepts of complexity theory developed in the 1980s, including complexity classes such as P, NP, and NP-completeness. Through the Cook-Levin Theorem [Coo71, Lev73] and the transformation of NP problems like HAMPATH and SAT, we demonstrate the reducibility of NP problems to a unified framework, setting the stage for further developments.

Section 3: Interactive Proof Systems (IPS) Recognizing the limitations of NP-based proof systems in verifying certain problems effectively, we delve into interactive proof systems. IPS extend NP by incorporating randomness and interaction, enhancing verification efficiency at the cost of allowing a small probability of error. To address the practical challenges inherent in traditional IPS, we explore the concept of knowledge complexity, as originally introduced in [GMR85], this approach enables a P[1] with bounded computational resources to convincingly demonstrate possession of secret knowledge to V, facilitating high-probability verification by quantifying V's ability to distinguish between a simulator and a genuine P, knowledge complexity elucidates how zero-knowledge protocols maintain the secrecy of underlying information while still proving its validity.

Section 4: The Power of Low-Degree Polynomials This section explores essential mathematical theories and cryptographic protocols based on Thaler's seminar PPT [Tha19], including the Schwartz-Zippel lemma and Reed-Solomon error correction. These underpin the efficacy of low-degree polynomials in error detection and interactive proof systems.

Section 5: The Sum-Check Protocol We provide a detailed, step-by-step introduction to the sum-check protocol [LFKN90], proving its soundness and analyzing its runtime characteristics. While theoretically applicable to all NP problems via SAT reduction, we highlight its limitation in requiring superpolynomial time for general-purpose computations by an honest P.

Section 6: The GKR Protocol To overcome the limitations of the sum-check protocol, we introduce the GKR protocol [GKR08], a sophisticated general-purpose interactive proof system developed in the 2010s. We demonstrate how the sum-check protocol integrates into the GKR framework to achieve efficient and sound verification of computations in polynomial time for both P and V. This protocol forms the foundation for the most recent verifiable computing models, including ZK-SNARKs.

In conclusion, this survey not only reviews the historical and theoretical advancements in verifiable computing over the past thirty years but also provides an accessible foundation for understanding significant recent developments. By assembling the various components of verifiable computing, we offer a cohesive understanding of the field's progression towards efficient and secure computational verification protocols.

[1]In this survey, we use P and V to represent the prover and the verifier in the interactive proof model.

2 Preliminaries: Complexity Theory

Complexity theory is a cornerstone of theoretical computer science, focusing on classifying computational problems based on the resources required to solve them, such as time and space. This field seeks to understand the fundamental limits of what can be efficiently computed and to categorize problems according to their inherent difficulty. This section provides a survey of key concepts in complexity theory, drawing upon the foundational texts by Michael Sipser [Sip97], M. R. Garey and D. S. Johnson [GJ79].

2.1 P (Polynomial Time)

P (Polynomial time decidable languages) is the class of languages that is decidable in polynomial time ($\mathcal{O}(n^k)$) on a deterministic single-tape Turing machine. Formally, P is defined as:

$$P = \bigcup_{k \in \mathbb{N}} \text{Time}(n^k)$$

where k is a constant. This definition encompasses all decision problems deemed tractable, meaning their solutions can be efficiently computed as the input size increases.

2.2 NP (Nondeterministic Polynomial time)

In P, we can avoid brute-force search and solve the problem using efficient algorithms. However, attempts to avoid brute force in certain other problems, including many interesting and useful ones, haven't been successful, and polynomial time algorithms that solve them aren't known to exist.

Language View: [Sipd]

- **P** is the class of language which membership can be **decided** (solved) quickly.
- **NP** is The class of language which membership can be **verified** quickly.

Turing Machine View:

- **P** is the set of problems **solvable** in polynomial time by a **deterministic TM**
- **NP** is the set of problems **verifiable** in polynomial time by a **deterministic TM** and **solvable** in polynomial time by a **non-deterministic TM**.

2.2.1 NTM Decider

A Non-deterministic Turing Machine (NTM) decider is guaranteed to halt on all inputs. [Sip97] An NTM decider is designed so that every possible computation branch halts, either by accepting or rejecting the input. This means that for any input the machine is given, it will always come to a conclusion (halt) within a finite number of steps. There are no branches where the machine runs forever without deciding the outcome.

2.2.2 Solving NP

The non-deterministic nature of NP gives us an abstraction to imagine a machine (NTM) that could guess a solution "in parallel" and verify it quickly. If we had such a machine, it would allow us to "solve" NP problems quickly by magically finding the right solution path. However, for real-world deterministic machines, we still don't have efficient algorithms to solve many NP problems.

2.2.3 Example: Hamiltonian Path Problem

If a directed or undirected graph, G, contains a **Hamiltonian path**, a path that visits every vertex in the graph exactly once. The HAMPATH problem has a feature called polynomial verifiability that is important to understand its complexity. Verifying the existence of a Hamiltonian path may be much easier than determining its existence.

Theorem: HAMPATH belongs to NP: On input $\langle G, s, t \rangle$ (Say G has m nodes): we non-deterministically write a sequence $v_1, v_2, ..., v_m$ of m nodes, and only accept iff: a. $v_1 = s$ b. $v_m = t$ and c. each (v_i, v_{i+1}) is an edge and no v_i repeats.

We do not know whether the Co-HAMPATH is in NP: The reason is we do not know whether or not we can give a short certificate for a graph not have a Hamiltonian path [Sipd].

- **If P equaled NP:** Then we can test in polynomial time whether a graph has a Hamiltonian path by directly solving the problem, which yields a short certificate.
- **If P not equal to NP:** then co-HAMPATH is not an NP problem, since it is not easily verified.

2.3 NP-Completeness

The Relationship between P and NP indicates that whether all problems can be solved in polynomial time, typically, without **searching**. **NP-completeness** is a cornerstone concept in computational complexity theory, providing a framework for understanding the inherent difficulty of computational problems. Building upon the detailed exposition presented in Michael Sipser's Lecture Notes [Sipb], this section delves into the foundational definitions, key theorems, and proof techniques that characterize NP-complete problems.

2.3.1 P $\equiv$ NP

You can always eliminate searching. If these classes were equal, any polynomially verifiable problem would be polynomially decidable.

2.3.2 P $\neq$ NP

There were cases where you need to search. [Sip97] *"Most researchers believe that the two classes are not equal because people have invested enormous effort to find polynomial time algorithms for certain problems in NP, without success. Researchers also have tried proving that the classes are unequal, but that would entail showing that no fast algorithm exists to replace brute-force search."*

Defn: B is NP-complete if :

1. B is a member of NP
2. For all A in NP, A $\leq p$ B

Every language in NP has the polynomial time reduced to a complete language of NP, which means if B is NP-complete and B is in P then P = NP. One important advance on the P versus NP question came in the early 1970s with the work of Stephen Cook and Leonid Levin.[Coo71, Lev73] which shows that the Boolean Satisfactory Problem (SAT) is NP-complete.

NP-completeness is a very important complexity property of any question:

1. Showing NP-complete is strong evidence of computational intractability (hard).
2. Gives a good candidate for proving P $\neq$ NP.

Michael Sipser in 2020: [Sipc] *"Back 20 years ago, I was working very hard to show the composite number problem is not in P. And then, turns out, composite was in P* (proved by [AKS04]). *So it was the wrong to pick the composite number problem, but what NP-complete is guarantees is that: If you work on a problem, which is NP-complete, you can't pick the wrong problem, because if any problem is in* NP *and not in P, an NP-complete problem is going to be an example of that. Because if the NP-complete problems in P, everything in* NP *is in P."*

2.4 The 3SAT Problem

2.4.1 Conjunctive Normal Form (CNF)

A boolean formula ϕ is in Conjunctive Normal Form (CNF) if it has the form:

$$\phi = (x \vee \neg y \vee z) \wedge (\neg x \vee \neg s \vee z \vee u) \wedge ... \wedge (\neg z \vee \neg u)$$

- **Literal:** a variable $\neg x$ or a negated variable
- **Clause:** an OR of the literals.
- **CNF:** an AND of the clauses.
- **3CNF:** a CNF with exactly 3 literals in each clause.

2.4.2 SAT

Boolean satisfiability problem (SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values (TRUE or FALSE) in such a way that the formula evaluates to TRUE.

2.4.3 3SAT is the satisfatory problem restricted to 3CNF formulas

$$3SAT = \{\phi \mid \phi \text{ is a satisfiable 3CNF formular}\}$$

2.4.4 Theorem: 3SAT $\leq p$ K-CLIQUE

We will show that we can reduce 3SAT to K-CLIQUE in polynomial time by building a model on 3SAT. And hence to show that K-CLIQUE problem is also NP-complete.

The K-Clique Problem A k-clique in a graph is a subset of k nodes all directly connected by edges, the input of k-clique problem is an undirected graph and k. The output is a clique (closed) with k vertices, if one exists.

$$\text{K-CLIQUE} = \{\langle G, k\rangle \mid \text{ graph G contains a k-clique}\}$$

The K-Clique Problem is in NP: You can easily verify that a graph has a k-clique by exhibiting the clique.

Theorem 1 (**3SAT $\leq p$ K-CLIQUE** [Sipb])**.** *Given polynomial-time reduction f that maps ϕ to $\langle G, k\rangle$ where ϕ is satisfiable if and only if G has a k-clique. Given the structure of a CNF, a satisfying assignment to a CNF formula has ≥ 1 true literal in each clause.*

Definition 1 (G)**.** G: Assume each literal in the formula is a node in G, where:

- The forbidden edges:
  1. No edges within a clause.
  2. No edges that go between inconsistent labels (a and $\neg a$)
- G has all non-forbidden edges.
  - k is the number of clauses
  - Other than those forbidden edges, all other edges are connected.

Claim: ϕ **is satisfiable iff** G **has k-clique**. We will show that we can reduce 3SAT to K-CLIQUE in polynomial time by **constructing a model based on 3SAT**.

$$\phi = (a \vee b \vee \neg c) \wedge (\neg a \vee b \vee d) \wedge (a \vee c \vee \neg e) \wedge ... \wedge (\neg x \vee y \vee \neg z)$$

Proof. **If ϕ is satisfiable $\implies$ G has a k-clique.** [Sipb]We begin by taking any satisfying assignment to ϕ, pick 1 true literal in each clause. assuming that 3SAT is solvable. Then the corresponding nodes in G are a k-clique:

1. **There are at least k nodes:** Assign any node to 1 means a valid node G.
2. **No forbidden edges among them**: based on Definition 1, those nodes on different clauses have edges connected, and since all the labels came from the same assignment. (a is true then $\neg a$ is false, we cannot pick the inconsistent nodes in different clauses)

□

Proof. **If G has a k-clique $\implies$ it will make ϕ satisfiable.** [Sipb]Taking any k-clique in G. It must have 1 node in each clause, because when we construct 3SAT from given G, nodes cannot appear in a clique together, since there are k clauses, each clause must have exactly one node to form a k-clique graph. □

Setting each corresponding literal TRUE gives a satisfying assignment to ϕ., the reduction f is computable in polynomial time, which suggests that: **If k-clique can be solved in polynomial time, then 3SAT can be solved in polynomial time.** Conversely, a polynomial-time solution to 3SAT implies that all NP problems, including K-clique, are in P.

2.5 The Cook-Levin Theorem

Once we have one NP-complete problem, we may obtain others by polynomial time reduction from it, as we've seen in K-CLIQUE. However, establishing the first NP-complete problem is more difficult. Now we do so by proving that SAT is NP-complete. In 1971, Stephen Cook states that the **Boolean satisfiability problem is NP-complete** [Coo71]. That means any problem in NP can be reduced in polynomial time by a deterministic Turing Machine to the boolean satisfiability problem (SAT).

2.5.1 SAT is in NP

A nondeterministic polynomial time machine can guess an assignment to a given formula ϕ and accept if the assignment satisfies ϕ.

2.5.2 For each A in NP, we have A $\leq p$ SAT:

(Any language in NP is polynomial time reducible to SAT).

Proof Idea: [Sip97]Let N be a nondeterministic Turing machine that decides A in n^k time for some constant k. *We are trying to proof that for any w belongs to any* NP *problems, there is a polynomial time reduction procedure that can transform that w to $\phi(SAT)$*

Key to the Proof: For any w belongs to any NP problems, it can be determined in polynomial time by a nondeterministic Turing machine N, say the running time is n^k. Then we can construct a Tableau for N is an $n^k \times n^k$ table whose rows are the configurations of a branch of the computation of N on input w. Which represents the computation steps/history of that branch of NTM (N). Based on this Tableau, by carefully define each part of *Phi*:

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

We can show that the construction time is is $O(n^{2k})$, the size of ϕ is polynomial in n. Therefore we may easily construct a reduction that produces ϕ in polynomial time from the input w of any NP problem.

3 Interactive Proof Systems

3.1 The Limitation of NP Proof Systems

Let's recall Stephen Cook and Leonid Levin's influential model definition of NP: [Coo71] The NP proof-system consists of two communicating deterministic Turing machines A and B: respectively, the P and the V. Where the P is **exponential-time**, the V is **polynomial-time**. They read a common input and interact in a very elementary way. On input a string x belonging to an NP language L, A computes a string y (whose length is bounded by a polynomial in the length of x) and writes y on a special tape that B can read. B then checks that $f_l(y) = x$ (where f_l is a polynomial-time computable function) and, if so, it halts and accepts.

3.1.1 $f_l(y) = x$

We can understand $f_l(y) = x$ in this way: "The output (certificate) y belongs to the input x, where $f_l()$ is a function that can check that y in poly-time."

3.1.2 Formalization vs. Intuition

Sometimes formalization cannot entirely capture the inituitive notions. In the context of **theorem-proving**: NP captures a specific form of proving a theorem, where a proof can be "written down" and verified without interacting with P [GMR85, Section 3] The certificates, which is like a formal written proof, and V just passively checks it. This is like reading a proof in a book. Once you have the book, there is no back-and-forth to clarify or ask questions about the proof.

3.1.3 Example: Co-HAMPTATH Problem

As we mentioned in the previous section, we do not know whether the complement of HAMPATH (Co-HAMPATH) is in NP:

- y is easy to be verified: **HAMPATH**

  For the Hamiltonian Path (HAMPATH) problem, given a solution (i.e., a path), it's easy for a V to check it in polynomial time.

  P can just present the path (certificates, or y), and V checks whether it's a valid Hamiltonian path (i.e., visits each vertex exactly once and satisfies the graph's edges).

- y is hard to be verified: **Co-HAMPATH**

  The Co-HAMPATH problem asks whether a graph does not have a Hamiltonian path. Here, proving the non-existence of something becomes far more complex.

  If you ask a P to convince you that no Hamiltonian path exists, the proof isn't as simple as just pointing to something (like a path). Instead, you'd need to somehow verify all possible paths don't work, which could take **exponential time**.

3.1.4 Limitation of the NP Proof-System

In the NP model, some problems in NP (like HAMPATH) are easily verifiable, but NP does not capture the complexity of some other problems (often their complements i.e., Co-NP problems). That's why problems like Co-HAMPATH are much harder to verify using the static NP model: **In our example, there isno easy way for a P to present a simple "proof" that no Hamiltonian path exists, and for V to check it efficiently.**

3.2 The Interactive Proof Systems

In 1985, Goldwasser et al. [GMR85] introduced an interactive proof-systems to capture a more general way of communicating a proof.

Much like in computation, BPP [Sip97, Section 10.2.1: The class BPP, pp. 336–339] (Bounded-error Probabilistic Polynomial time) algorithms provide a probabilistic analog to P to enhance efficiency while accepting a small chance of error for that speed. In verification, **IP (Interactive Proof) systems provide a way to define a probabilistic analog of the class NP**. IP includes problems not known to be in NP, demonstrating greater verification power due to *randomness* and *interaction.*

3.2.1 Interactive Pairs of Turing Machines

- P **(Prover)**

  An entity with unlimited computational power, aiming to convince V the truth of a statement.

- V **(Verifier)**

  A probabilistic polynomial-time Turing machine (with a random tape) that interacts with P to verify the statement's validity.

3.2.2 Interactions

The interaction consists of multiple rounds where P and V exchange messages. V uses randomness to generate challenges, and P responds accordingly. The key properties of such systems are:

- **Completeness:** If the statement is true, an honest P can convince V with high probability.
- **Soundness:** If the statement is false, no cheating P can convince V except a small probability.

3.2.3 Example: Quadratic Nonresidue Problem

An integer a is a quadratic residue modulo n if there exists an integer x such that:

$$x^2 \equiv a \pmod{n}$$

An integer a is a quadratic nonresidue modulo n if no such integer x exists. Suppose A (Prover) claim that a is a **quadratic nonresidue**, and the B (Verifier) wants to check that using an **Interactive Proof System**.

An Interactive Proof System to the QAP can be: B begins by choosing m random numbers $\{r_1, r_2, ..., r_m\}$. For each i, $1 \leq i \leq m$, he flips a coin:

- If it comes up heads he forms $t = r_i^2 \pmod{m}$.
- If it comes up tails he forms $t = a \times ri^2 \pmod{m}$.

Then B sends $t_1, t_2, ..., t_m$ to A, who having unrestricted computing power, finds which of t_i are quadratic residues, and uses this information this information to tell B the results of his last m coin tosses. If this information is correct, B accepts.

Why this Will Work?

- If a is really a quadratic nonresidue:

  According to the property of quadratic nonresidue:

  - $t = a \times r_i^2 \pmod{m}$ is a quadratic non-residue.
  - $t = r_i^2 \pmod{m}$ is a quadratic residue.

  P can distinguish which side of the coin by looking whether t is a quadratic nonresidue or residue.

- If a is a quadratic residue (P is lying):

  Then both $t = a \times r_i^2 \pmod{m}$ and $x = r_i^2 \pmod{m}$ are quadratic residues.

  Which means t_i are just random quadratic residues, all t_i looks the "same" for P to guess the coin side, P will respond correctly in the last part of the computation with probability $1/2^m$.

3.3 The Power of Interactive Proof Systems

As mentioned in Section 3.1.4, While traditional NP proof systems are powerful for verifying the existence of solutions, they encounter significant limitations when it comes to proving the non-existence of solutions. Specifically, NP proof systems are inherently designed to provide short certificates for *yes*-instances of decision problems. However, for *no*-instances, such as those in the class coNP, no analogous short certificates are known. A quintessential example is the Co-Hamiltonian Path problem (Co-HAMPATH), where we seek to verify that a given graph does not contain a Hamiltonian path. Currently, it remains unknown whether Co-HAMPATH resides in NP , primarily because we lack efficient methods to certify the absence of a Hamiltonian path.

Theorem 2 ([Sha92])**.** $\mathsf{coNP} \subseteq \mathsf{IP}$

This theorem, established by Shamir in 1990, reveals a profound capability of interactive proof systems: they can handle the complements of NP problems efficiently. In other words, for every problem in coNP, there exists an interactive proof system where a P can convince a V of the truth of a statement without V needing to check an exhaustive list of possibilities. This inclusion signifies that interactive proof systems transcend the limitations of traditional NP proof systems by enabling the verification of *no*-instances through interactive protocols.

The detailed proof of this theorem leverages the *sum-check protocol*, a pivotal technique in interactive proofs that facilitates the verification of complex statements through a series of interactive rounds between P and V. For an in-depth exploration of the sum-check protocol and its role in proving that $\mathsf{coNP} \subseteq \mathsf{IP}$, we refer the reader to Section 5.7.

In Practice: By assuming P has unlimited computing power, The theoretical model introduced in Interactive Proof Systems can describe many languages that cannot be captured using NP model. But back to practice, seems that this model will only work in theory until those kinds of unlimited computing machine comes in real life. (Can determine NP problems in polynomial Time). (i.e., Can quickly determine NP problems like whether t is a quadratic nonresidue or residue) Is that true?

3.4 Secret Knowledge

It is true that having unlimited computing machine is infeasible in current practice. However, by assuming P runs in polynomial time with some "secret knowledge" that can help it communicating with V efficiently. It can convince V that P has that "secret knowledge" without revealing it.

3.4.1 Eyewitness & Police Officer

Let us try to illustrate the above ideas using an informal example: Assume that a crime x has happened, B is a police officer and A is the only eyewitness. A is greedy in telling B that to tell him about what happened in x, \$100,000 must be transferred to his bank account first. For B, it is important to verify whether A has that "secret knowledge" – details of crime x before making transfer. And for obvious reasons, A cannot just prove that he has that "secret knowledge" by telling it directly to the police officer B. By using interactive proof systems, A can convince B that he has that "secret knowledge" x without revealing it.

3.4.2 Quadratic Nonresidue Problem

In the interactive proof system describe in Quadratic Nonresidue Problem, the key challenge for P (A) is to determine whether each number t_i sent by V (B) is a quadratic residue or a quadratic nonresidue modulo m. This determination is crucial because it allows P to infer the results of B's coin tosses and respond correctly. In this case, the "secret knowledge" for efficient computation on A is the **prime factorization of the modulus** m[2]. If P knows the prime factors of m, they can efficiently compute the *Legendre*[3] or *Jacobi*[4] symbols to

[2]Note: In practice, the modulus m is often chosen such that its factorization is hard to obtain (e.g., a product of two large primes), ensuring that without the secret knowledge, determining quadratic residuosity remains difficult.

[3]Legendre, A. M. (1798). Essai sur la théorie des nombres. Paris. p. 186.

[4]Jacobi, C. G. J. (1837). "Über die Kreisteilung und ihre Anwendung auf die Zahlentheorie". Bericht Ak. Wiss: 127–136.

determine quadratic residuosity. This "secret knowledge" enables polynomial P (A) to interact with B that are otherwise computationally infeasible.

("secret knowledge" + poly-time machine $\equiv$ unlimited computing power)

3.5 The Knowledge Complexity

3.5.1 The Knowledge Computable from a Communication

Which communications (interactions) convey knowledge? [GMR85, Section 3.2] Informally, those that transmit the output of an unfeasible computation. How to ensure that V gains no secret knowledge beyond the validity of the statement being proven?

3.5.2 Knowledge Complexity

Simulator: [GMR85] introduce the idea of **simulator**: An algorithm that can generate transcripts of the interaction without access to P's secret information.

Quantifying Knowledge By linking the **chance** that V is able to distinguish the simulator, we quantify the knowledge complexity of proofs. If the simulator can effectively replicate the interaction, making it indistinguishable to V, **the knowledge complexity** is considered zero.

This formalization allows us to assess and prove the zero-knowledge property of certain interactive proofs by showing that sometimes V cannot gain knowledge because whatever it sees could be simulated without P's help.

The Simulator acts as an algorithm that can generate transcripts of the interaction between P (A) and V (B) without **knowing P's secret knowledge**. By producing transcripts that are indistinguishable from those of a real interaction, the simulator demonstrates that V gains no additional knowledge from the interaction. Therefore, if such a simulator exists, we say that the proof is zero-knowledge because any information V receives could have been simulated without P's secret.

3.5.3 Zero Knowledge Interactive Proof System for the QRP

[GMR85, Section 4.2] introduces a zero knowledge IP system for the quadratic residue problem by carefully designing the protocol and demonstrating the existence of a poly-time simulator. The difficulties of the proof is that M must compute the coin tosses correctly as a real P (A) with secret knowledge does. Since the simulator M simulates both sides of the interaction, it both can know/control the randomness of the coin.

4 The Power of Low-Degree Polynomials

This section of the survey builds upon the concepts presented by Justin Thaler [Tha19] during the Proofs, Consensus, and Decentralizing Society Boot Camp in 2019. Thaler's insightful discussion on the power of low-degree polynomials in verifiable computing serves as the backbone for the detailed explanations and proofs provided herein.

4.1 Example: Equality Testing

Two parties (i.e., Alice and Bob) each have an equal-length binary string:

$$a = (a_1, a_2, ..., a_n) \in \{0,1\}^n \mid b = (b_1, b_2, ..., b_n) \in \{0,1\}^n.$$

They want to collaborate with each other (No malicious user) to determine whether $a \equiv b$, while exchanging as few bits as possible.

4.1.1 A trivial solution

Alice sends a to Bob, who checks whether $a \equiv b$. The communication cost is n, which is optimal amongst deterministic protocols.

4.1.2 A logarithmic cost randomized solution

According to [Tha22, Section 2.3], let F be any finite field with $|F| \geq n^2$, then we interpret each a_i, b_i as elements of F: Let $p(x) = \sum_{i=1}^{n} a_i \times x^i$ and $q(x) = \sum_{i=1}^{n} b_i \times x^i$

1. Alice picks a random r in F and sends $(r, p(r))$ to Bob.
2. Bob calculates $q(r)$, outputs EQUAL iff. $p(r) \equiv q(r)$, otherwise output NOT-EQUAL.
   - **Total Communication Cost: $\mathcal{O}(\log n)$ bits**
     Since there are at least total n^2 elements in F, to represent each of the elements, we need $\log(|F|) = \log(n^2) = 2\log(n) = \mathcal{O}(\log n)$ bits.
3. If $a \equiv b$: Then Bob outputs EQUAL with probability 1
4. If $a \neq b$: Then Bob outputs NOT-EQUAL with probability at least $(1 - \frac{1}{n})$ over the choice of r in F.

A detailed proof of this statement will be given in the section 4.2.

4.2 Low-degree Polynomials

4.2.1 Field

Field arises from the need for a structured and versatile system in mathematics and science to perform algebraic operations in a consistent and predictable way.

A field is a set equipped with two operations, addition and multiplication, along with their respective inverses, subtraction and division (except division by zero). Operations in a field follow specific rules, such as commutativity, associativity, and distributivity, ensuring that the result of any operation between elements of the field remains within the field itself (closure). Here are some non-field examples:

- The set of 2x3 matrices cannot perform multiplication.
- The set of 2x2 matrices, multiplication between any two elements is not commutative.

- Some of the elements in set Z/6Z (integers modulo 6) do not have multiplicative inverses.

  A number a in Z/6Z has a multiplicative inverse if there exists a number b in Z/6Z such that $a \times b \equiv 1 \pmod 6$. For example, 2, 3, 4 do not have a multiplicative inverse, in fact, integers mod p (Z/pZ) is a field when p is a prime number.

4.2.2 Reed-Solomon Error Correction

Since there are total n bits of a and b, the lowest-degree polynomials that for us can ensure the uniqueness of representation of each a_i and b_i is n. Which means each bit a_i (and correspondingly b_i) is uniquely represented as the coefficient of a distinct term in the polynomial and affects the polynomial differently. And that is why we need to define $p(x)$ and $q(x)$ in the following way for error detection in equality testing:

$$p(x) = \sum_{i=1}^{n} a_i \times x^i$$

$$q(x) = \sum_{i=1}^{n} b_i \times x^i$$

Theorem 3. *Any non-zero polynomial $d(x)$ of degree n has at most n roots*

Proof. Assume the polynomial has more than n roots. Let $r_1, r_2, ..., r_{n+1}$ be distinct elements of the field F, such that $d(r_i) = 0$ for each $i = 1, 2, ..., n+1$. Then the polynomial d(x) can be written as:

$$d(x) = (x - r_1)(x - r_2)...(x - r_{n+1})q(x)$$

where $q(x)$ is some polynomial of degree $m \geq 0$ and $(x - r_i)$ are factors corresponding to the roots. Then the product of $(x - r_1)(x - r_2)...(x - r_{n+1})$ is a polynomial of degree $n + 1$. This assumption leads to a contradiction. Hence the polynomial $d(x)$ can have at most n distinct roots. $\square$

Theorem 4. *In section 4.1, if $a \neq b$, then the probability of Bob is wrong is at most $\frac{1}{n}$*

Proof. In equality testing, if $p(x) \neq q(x)$, then the chance that Alice picks a random r in F such that $d(r) = p(r) - q(r) \equiv 0$ is at most $(\frac{n}{|F|} \leq \frac{n}{n^2} \leq \frac{1}{n})$. The reason is that a n-degree polynomial $d(r) = p(r) - q(r)$ has at most n roots, a randomly picked value r in F of size n^2 will only let $d(r)$ equal to zero with a probability $(\frac{n}{n^2} = \frac{1}{n})$. $\square$

4.2.3 Polynomials are Constrained by Their Degree

As we can see that, A polynomial $d(x)$ of degree n over a large field F is uniquely determined by its values on $n + 1$ distinct points. (one extra constant coefficient with no variable)

If $p(x)$ is not equal to $q(x)$, then they can only agree on at most n points ($d(x) = p(x) - q(x) = 0$), meaning they **differ at most everywhere** on the field. (Any two polynomials of degree n can agree in at most n places, unless they agree everywhere.) This strong divergence is very useful and powerful for error detection.

The Power of Low-degree Polynomials In practice, we aim to keep the degree of a polynomial as low as possible while ensuring that each term uniquely affects the polynomial. A polynomial of degree n has $n + 1$ terms, each with a distinct power of x and a unique coefficient, which ensures that every term influences the polynomial differently. Low-degree polynomials are powerful because if two polynomials differ, they will diverge over many points when evaluated multiple times with efficient evaluation. This makes discrepancies clear across a larger number of evaluations. In cryptography, this property allows for efficient detection of errors or differences, especially when random evaluations are used.

4.3 Example: Freivalds's algorithm for Verifying Matrix Products

Input are two $(n \times n)$ matrices A, B. The goal is to verify the correctness of $A \cdot B$. The time complexity of matrix multiplication is $\mathcal{O}(n^3)$, this is because each element in the resulting matrix $A \cdot B$ is computed by taking the dot product of a row of A and a column of B. For each of n^2 elements in the resulting matrix, you perform n multiplications and additions, leading to a total of $\mathcal{O}(n^3)$ operations. The best bound of matrix multiplication algorithm for now is $\mathcal{O}(n^{2.371552})$ [WXXZ24].
If a P claims the answer of $A \cdot B$ is a matrix C? Can V verify it in $\mathcal{O}(n^2)$ time?

$\mathcal{O}(n^2)$ **Protocol:[Fre77]**

1. V picks a random r in F and lets $x = (r, r^2, ..., r^n)$.
2. V computes $C \cdot x$ and $A \cdot (B \cdot x)$, accepting if and only if they are equal.

Runtime Analysis: V's runtime dominated by computing 3 matrix-vector products, each of which takes O(n^2) time.

- $C \cdot x$ is one matrix $(n \times n)$ times a vector $(n \times 1)$, the time complexity is $\mathcal{O}(n^2)$.

  Because each row of B is multiplied by the vector x, requiring n multiplications and n additions per row, and there are n rows.

- $(A \cdot B) \cdot x = A \cdot (B \cdot x)$ takes two matrix-vector multiplications.

  Matrix multiplication is associative, $B \cdot x$ takes $\mathcal{O}(n^2)$ first, and produces a $(n \times 1)$ matrix M, then $A \cdot M$ will also take $\mathcal{O}(n^2)$.

Correctness Analysis:

- If $C \equiv A \cdot B$:

  Then V accepts with probability 1

- If $C \neq A \cdot B$:

  The V rejects with high probability at least $(1 - \frac{1}{n})$.

Simplified Proof. Recall that $x = (r, r^2, ..., r^n)$. So each matrix-vector multiplication is the polynomials we've seen in section 4.2.2 Reed-Solomon Error Correction at r of the i-th row of C. So if one row of C does not equal the corresponding row of $A \cdot B$, the fingerprints for that row will differ with probability at least $(1 - \frac{1}{n})$, causing V to reject w.h.p. □

4.4 Function Extensions

4.4.1 Schwartz-Zippel Lemma

The Schwartz-Zippel Lemma[Sch80] is an extension of univariate error-detection to multivariate polynomials. If p and q are distinct l-variate polynomials of **total degree** at most d. Then the same kind of statement holds. If we evaluate them at randomly chosen inputs, they agree at the probability at most $\frac{d}{|F|}$.

Total Degree: The total degree of a polynomial is the maximal of the sums of all the powers of the variables in one single monomial.
For example: $\deg(x^2yz^4 - 3y + 4xe^5 - xy^3z^2) = 7$ (first monomial).

4.4.2 Extensions

An extension polynomial bridges the gap between a function defined on a discrete set of points and a function defined over a continuous (or larger discrete) domain.

A l-variate polynomial g over F is said to extend f if and only if g agrees at all of the input where f is defined. For example: We are given a function f that maps l-bits binary strings to a field F. This means f is defined on all possible combinations of l bits ($\{0,1\}^l$). But it is only defined on a finite set of points (the binary strings), which usually cannot form a field, **where we can leverage algegratic tools of polynomials**.

A function g is said to extend f if:

- For all $x \in \{0,1\}^l$, $f(x) = g(x)$.
- g is defined on a larger field

Let's say $l = 1$, then f is defined on input set $\{0,1\}$, where $f(0) = 2$, $f(1) = 3$. If we want to extend f to field R (real numbers), then our objective is to find a polynomial $g(x)$ in R such that g agrees with f on all inputs where f is initially defined. (i.e., $g(0) = 2$, and $g(1) = 3$). We can find $g(x) = x + 2$ defined for all x in R, not just $\{0,1\}$.

By representing f as a polynomial g, **we can apply the rich toolbox of algebraic methods and theorems available for polynomials**. For example, Schwartz-Zippel Lemma is more powerful when there is a low-degree extension represents that function.

4.4.3 Constructing Low-Degree Extensions

In this section, we present a general way to construct low-degree extensions.

Notations: There is a vector $w = (w_0, w_1, ..., w_{k-1})$ in F^k. $W : H^m \to F$: We define a function $W : H^m \to F$ such that $W(z) = w_{a(z)}$ if $a(z) \leq k-1$, and $W(z) = 0$ otherwise.

- W acts as a way to represent the vector w as a function over H^m that for indices corresponding to elements of w, $W(z)$ returns the corresponding w_i, otherwise 0.
- $a(z) : H^m \to F$ is the lexicographic order of z, which means **transforming an m-element vector to an index in the original w**.

Low-Degree Extension $\widetilde{W} : F^m \to F$: $\widetilde{W}$ is an extension of W input from H^m to F^m, such that $\widetilde{W}$ is a polynomial of degree **at most** $|H| - 1$ in each variable, which enables efficient computation and has nice algebraic properties.
The degree of W is at most |H|-1 can be understood in the following cases:

Univariate Case: $W(x) : H \to F$:

- We have $n = |H|$ distinct points in the subset H.
- For each $h \in |H|$, we want $\widetilde{W}(x) : F \to F)$ to satisfy $\widetilde{W}(h) \equiv W(h)$. **A univariate polynomial of degree at most $(|H| - 1)$ can be uniquely determined by its values on those n points**.

Multivariate Case: $W(x) : H^m \to F^m$

- Similarly, we have a m-dimensional grid H^m, where $H \subseteq F$ and $|H| = n$.
- A polynomial $\widetilde{W}(x_1, x_2, ..., x_m)$ (in m variables) that:
    1. Agrees with W on every point in H^m (i.e., $\widetilde{W}(h) = W(h)$ for all $h \in H^m$.
    2. Has degree at most $|H| - 1$ in each variable. (i.e., for each variable x_i, the highest exponent of x_i in $\widetilde{W}$ is $\leq |H| - 1$. In other words, just like in the univariate case (where we need $deg(\widetilde{W}) \leq |H| - 1$ to interpolate $|H|$ points), here each variable is similarly bounded by $|H| - 1$. This ensures $\widetilde{W}$ can uniquely "pass through" all the points specified by W on H^m.

The low-degree extension is the **simplest** polynomial that fits all the given points in H^m, The size of H determines how "complex" the polynomial needs to be (i.e., degree) in order to pass through all those points without ambiguity.

Here is the full definition of $\widetilde{W} : F^m \to F$:

$$\widetilde{W}(t_1, ..., t_m) = \sum_{i=0}^{k-1} \widetilde{B_i}(t_1, ..., t_m) \cdot w_i.$$

Indicator Function: $\widetilde{B_i} : F^m \to F$ The polynomials $\widetilde{B_i}$ act as an indicator functions on H^m, on that field, $\widetilde{B_i}(z) = 1$ if and only if $i \equiv a(z) = a(t_1, ..., t_m)$. Otherwise $\widetilde{B_i}(z) = 0$.

Outside H^m (in F^m/H^m, $\widetilde{B_i}$ takes on values determined by its polynomial extension, which means when input is outside H^m, $\widetilde{W}$ can have sum of multiple terms.

Sum Selection: $\widetilde{W} : F^m \to F$ $\widetilde{W}$ is a low-degree polynomial, summing all possible k indexes i from 0 to $k - 1$, in the field of H^m. According to the definition of $\widetilde{B_i}$ there will be only one "selected" corresponding w_i which is equal to $\widetilde{W}$.

Lagrange Basis Polynomial: $\widetilde{B}(z, p) : F^m \to F$ Also, we can express $\widetilde{W}(t_1, ..., t_m)$ as:

$$\widetilde{W}(z) = \sum_{p \in H^m} \widetilde{B}(z, p) \cdot W(p)$$

Which constructs the polynomial $\widetilde{W}$ by summing the contributions from all basis polynomials $\widetilde{B}(z, p)$, each weighted by $W(p)$.

- $\widetilde{B}(z, z) = 1$, $\widetilde{B}(z, p) = 0$ for all $z \in H^m$ where $z \neq p$
- When z outside H^m, $\widetilde{B}(z, p)$ can be other values

This means, when z is in F^m/H^m, each $\widetilde{B}(z, p)$ is a polynomial in z and can be evaluated at any z in F^m. And To compute $\widetilde{W}(z)$, which means by summing over all $W(p) \in H^m$ that has valid $\widetilde{B}(z, p)$. Since W is only defined in the field H^m. And the low-degree polynomial property still holds. By enlarging the input field (F^m) while keeping the degree of $\widetilde{W}$ low, this way of constructing LDE $\widetilde{W}$ helps us effectively using the power of randomness to detect the potential error.

4.4.4 Multilinear Extensions

A multilinear extension of a function $f : \{0,1\}^n \to F$ (where F is a finite field) is a polynomial $\bar{f} : F^n - > F$ that agrees with f on $\{0,1\}^n$ and is **multilinear**, which means each variable x_i in $\bar{f}$ has a degree at most 1, which make them highly effective to evaluate. (a univariate f by make other variables constants will become a linear function).

And since multilinear polynomials have minimal degree, the error detection probability is maximized for a given field size.

5 The Sum-Check Protocol

Suppose given a l-variate polynomial g defined over a finite field F. The purpose of the **sum-check protocol** [LFKN90] is to compute the sum:

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(b_1, b_2, ..., b_l)$$

In applications, this sum will often be over a large number of terms, so V may not have the resources to compute the sum without help. Instead, she uses the **sum-check protocol** to force P to compute the sum for her. V wants to verify that the sum is correctly computed by P, where g is a known multivariate polynomial over finite field F.

5.1 Initialization

P claims that the total sum equals a specific value H_0.

5.2 First Round

Protocol 1 First Round of the Sum-check Protocol

1. P sends a univariate polynomial $s_1(x_1)$ to V, which is claimed to equal:

$$s_1(x_1) := \sum_{b_2 \in \{0,1\}} \sum_{b_3 \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(x_1, b_2, ..., b_l)$$

2. V calculates $s_1(0) + s_1(1)$ and checks whether that value is equal to H_0.

   Since s_1 is a univariate polynomial, V can compute $s_1(0) + s_1(1)$ directly (not using structure of H_0) in relatively short amount of time.

3. V picks a random element r_1 from F and sends it to P.

4. V sets $H_1 := s_1(r_1)$ for use in the next iteration.

$$H_1 = s_1(r_1) := \sum_{b_2 \in \{0,1\}} \sum_{b_3 \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(r_1, b_2, ..., b_l)$$

5.3 Iterative Rounds ($i = 2$ to l)

For each round i (i from 2 to l):

Protocol 2 Round i to l of the Sum-check Protocol

1. P sends a univariate polynomial $s_i(x_i)$ to V, claimed to equal:

$$s_i(x_i) := \sum_{b_{i+1}\in\{0,1\}} \dots \sum_{b_l\in\{0,1\}} g(r_1, \dots, r_{i-1}, x_i, b_{i+1}, \dots, b_l)$$

   Which represents the partial sum over variables b_{i+1} to b_l, with b_1 to b_{i-1} fixed to random values chosen by V in previous rounds.

2. V calculates $s_i(0) + s_i(1)$ and checks whether that value is equal to H_{i-1}. H_{i-1} is the sum in the previous iteration: $s_{i-1}(r_{i-1})$:

$$H_{i-1} := s_{i-1}(r_{i-1}) := \sum_{b_i\in\{0,1\}} \dots \sum_{b_l\in\{0,1\}} g(r_1, \dots, r_{i-1}, b_i, \dots, b_l)$$

3. V picks a random element r_i from F and sends it to P.

4. V sets $H_i = s_i(r_i)$ for use in the next iteration.

$$H_i = s_i(r_i) := \sum_{b_{i+1}\in\{0,1\}} \dots \sum_{b_l\in\{0,1\}} g(r_1, \dots, r_i, b_{i+1}, \dots, b_l)$$

5.4 Final Check

In the final iteration:

$$H_l = s_l(r_l) := g(r_1, r_2, \dots, r_l)$$

All b_i in the original equation of $g(b_1, \dots, b_l)$ has been fixed to the random number r_i chosen by V in previous l rounds. V then checks whether $s_l(r_l)$ is equal to $g(r_1, r_2, \dots, r_l)$ **by calculating it** herself. **If $s_l(r_l)$ is equal to $g(r_1, \dots, r_l)$, V accepts**.

5.5 Soundness of the Sum-Check Protocol

Theorem 5. *The Sum-check Protocol is Sound and Complete.*

Completeness holds by design: If P sends the prescribed messages, then all of V's check will pass. Let's prove the **soundness** of the protocol:

Proof. The Sum-Check Protocol is **Sound**:
The Section 5.4 Final Check is the most crucial part, and plays a key role in understanding this protocol. In the last step, V directly computes $g(r_1, r_2, \dots, r_l)$ and $s_l(r_l)$ to check whether they are equal, note that this is the only time for V to actually compute l-variate polynomial g by herself.

This direct comparison is critical because it anchors the entire verification process to the actual function g. According to Schwartz-Zippel Lemma, in the final check, if $s_l \neq g$, then the probability $g(r_1, \dots, r_l)$ equal to $s_l(r_l)$ is less than $d/|F|$. (d is the Total Degree of polynomial g and s_l).

Backward Reasoning: By working backwards from the last iteration, We can understand the correctness of each step based on the validity of the final check:

In the last step, if the equality of $g(r_1, ..., r_l)$ and $s_l(r_l)$ holds, **with high probability, $s_l(x_l)$ must be the correct polynomial of $g(r_1, ..., x_l)$**, because a dishonest P would need to guess r_l to fake $s_l(x_l)$ in order to pass the test, and the chance of success is less than $d/|F|$, where d is the totdal degree of polynomial g over field F.
If V can confirm $s_l(x_l)$ is correctly formed w.h.p in Final Check where:

$$s_l(x_l) := g(r_1, ..., r_{l-1}, x_l)$$

Then in iteration $(l-1)$, $s_{l-1}(r_{l-1})$ can be written as:

$$\begin{aligned} s_{l-1}(r_{l-1}) &= \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{l-2}, r_{l-1}, b_l) \\ &= g(r_1, ..., r_{l-1}, 0) + g(r_1, ..., r_{l-1}, 1) \\ &= s_l(1) + s_l(2) \end{aligned}$$

Thus, $s_{l-1}(x_{l-1})$ must be correctly formed w.h.p based on correct $s_l(x_l)$.

Inductive Case: If $s_i(x_i)$ is correctly formed in round i For any round i, if we can make sure $s_i(x_i)$ is correctly formed w.h.p that:

$$s_i(x_i) = \sum_{b_{i+1} \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, x_i, b_{i+1}, ..., b_l)$$

Backwards to its previous round:

$$\begin{aligned} s_{i-1}(r_{i-1}) &= \sum_{b_i \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, b_i, ..., b_l) \\ &= \sum_{b_{i+1} \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, 0, b_{i+1}, ..., b_l) \\ &+ \sum_{b_{i+1} \in \{0,1\}} ... \sum_{b_l \in \{0,1\}} g(r_1, ..., r_{i-1}, 1, b_{i+1}, ..., b_l) \\ &= s_i(1) + s_i(2) \end{aligned}$$

Then, with high probability, $s_{i-1}(r_{i-1})$ should be correct, which indicate $s_{i-1}(x_{i-1})$ should also be correctly formed. **By induction, in the first round, since $s_2(x_2)$ is correctly formed, then w.h.p $s_1(x_1) = s_2(0) + s_2(1)$ should be formed correctly. And thus w.h.p, $H_0 = s_1(0) + s_1(1)$ should be the correct answer.** So far, we've proved the soundness of the protocol. □

Summary: We've proved the soundness of the sum-check protocol by showing that: According to Schwartz-Zippel Lemma, with high probability a dishonest P cannot initially fake a incorrect H_0 that won't break the consistent in every round of the protocol and causes a correct $s_l(r_l)$ equals to $g(r_1, ..., r_l)$ in the final round.

5.6 Analyzing the Sum-Check Protocol

5.6.1 A Scenario where V cannot Compute $g(r_1, ..., rl)$

The soundness is relying on the final check, by validating the final step, V effectively validates all prior steps due to their interdependence in the chain of validations. For example, here we show a specific cheating strategy when V cannot compute $g(r_1, ..., r_l)$ by herself:

Imagine a dishonest P consistently uses different function $j(x)$ instead of the correct function $g(x)$ throughout the protocol. In this scenario, P computes all the partial sums and polynomials correctly with respect to $j(x)$, ensuring consistency at each step, P only deviates from the correct computation at the final check when V computes $g(r_1, ..., r_l)$.

During the protocol, V only computes $g(x)$ in the final check, the only point of detection is the final check, and the probability of detection is $\frac{d}{|F|}$. But if there is no final check, which means the protocol ends at round l, **V cannot have any guess of what function P used to compute H_0.**

5.6.2 Probability of Successful Cheating by P in Sum-check Protocol

In this part, we are going to quantify the probability for a dishonest P can successfully convince V for the wrong computation H_0.

Claim. *For sum-check protocol on polynomial $g : F^l \to F$ with total degree d, if P is cheating at any round, the upper bound for V to accept is: $ld/|F|$.*

Proof. For any round i, let's assume what a dishonest P sends to V ($s_i(x_i)$) is formed incorrectly (i.e., $s_i(0) + s_i(1) \neq s_{i-1}(r_{i-1})$, deviation at round i), then.

In round i, $s_i(r_i)$ is set to H_i by V (in round l, H_i becomes to $g(r_1, ..., r_l)$). **The probability for P to provide s_i to satisfy polynomial $H_i - s_i(r_i) = 0$ with randomly selected r_i by V from field F is** $\frac{d}{|F|}$, where d is the total degree of polynomial g and s, according to Schwartz-Zippel Lemma.

If $s_i(r_i) \neq H_i$, P is left to prove a false claim in the recursive call: P must construct $s_{i+1}(x_{i+1})$ such that $s_{i+1}(0) + s_{i+1}(1) = s_i(r_i)$. This means $s_{i+1}(r_{i+1})$ must deviate from true H_{i+1}, leading to a new error polynomial in subsequent rounds.

Thus, we can get the the cumulative probability of acceptance when P deviates at round i and V does not detect in rounds $i + 1$ to l and end up accepting: $\frac{(l-i)d}{|F|}$.

This summation approach accounts for the fact that each round provides an independent opportunity for V to catch cheating, effectively adding an "**error margin**" with each additional round. Thus, more rounds increase the total soundness error linearly, rather than multiplicatively, then we have:

$$\begin{aligned} P[A] &\leq P[D_i] + P[D_{i+1,l}] \\ &\leq \frac{d}{|F|} + \frac{(l-i)d}{|F|} \\ &= \frac{(l-i+1)d}{|F|} \end{aligned}$$

Where A denote "V accepts", D_i denote "V does not detect at i", and $D_{i+1,l}$ denote "V does not detect in $\{i+1, \ldots, l\}$".

The possibility of acceptance when P deviates at round i is $(l - i + 1)d/|F|$. The worse-case scenario is that if P deviates in the first iteration ($i = 1$) (Error Margin), the total probability of acceptance is:

$$\frac{d}{|F|} + \frac{(l-1)d}{|F|} = \frac{ld}{|F|}$$

□

5.6.3 Example: $l = 2$

Let's consider a minimum sum-check protocol with $l = 2$ by working backward from the last iteration to understand the soundness:

1. Final Check:

   (a) V computes $g(r_1, r_2)$

   (b) Comparison with P's $s_2(r_2)$
   If the equality holds, with high probability, $s_2(x_2)$ must be correct polynomial $g(r_1, x_2)$, because a dishonest P would need to guess r_2 to fake $s_2(x_2)$.

2. Round 2: Since $s_2(x_2)$ that P sends to V is confirmed to be correct, the sum: $s_2(0) + s_2(1) = g(r_1, 0) + g(r_1, 1) = H_1$ must be satisfied. This confirms that H_1 is correctly computed based on $s_2(x_2)$.

3. Round 1: At the end of this round, P sets $H_1 = s_1(r_1)$. Since H_1 is now confirmed to be $g(r_1, 0) + g(r_1, 1)$, it implies $s_1(r_1) = g(r_1, 0) + g(r_1, 1)$, thus $s_1(x_1)$ is correct polynomial $\sum_{b_2 \in \{0,1}\}g(x_1, b_2)$ w.h.p, any deviation would be detected with high probability due to the random r_1.

4. Initialization: V check $s_1(0) + s_1(1) = H_0$ Since $s_1(x_1)$ is correct w.h.p, then H_0 should be correctly formed based on $s_1(x_1)$.

V only needs to perform a few evaluations of univariate-polynomials and checks (except the final check g), making the protocol practical even for large computations.

5.7 coNP $\in$ IP

Having introduced the sum-check protocol, we now demonstrate how it empowers interactive proof systems to verify statements in coNP efficiently. Recall from Section 2.5 the *Cook–Levin Theorem*, which establishes that the SAT problem is NP -complete. Consequently, any problem in NP can be transformed into an instance of SAT in polynomial time. A canonical example is the HAMPATH problem, which asks whether a given directed graph G contains a Hamiltonian path.

In this subsection, we turn our attention to *complements* of NP problems, such as Co-Hamiltonian Path (Co-HAMPATH), which asks whether a directed graph *does not* contain a Hamiltonian path. Traditional NP proof systems do not straightforwardly provide a "short certificate" for the non-existence of certain structures (e.g., no Hamiltonian path). Therefore, their capability to verify statements in coNP is limited.

By leveraging the sum-check protocol, we can Specifically, we reduce any coNP problem, such as Co-HAMPATH, to a corresponding #SAT instance. For instance, proving that #SAT$(\phi) = 0$ (i.e., ϕ has no satisfying assignments) corresponds to showing there is no Hamiltonian path in G. The key insight is that verifying a #SAT claim via the sum-check protocol can be done in *polynomial time* with the help of an interactive *P–V* framework.

Therefore, by composing these steps—polynomial-time reduction to #SAT plus an efficient interactive verification protocol (sum-check)—we conclude that every problem in coNP admits an interactive proof system.

Theorem 6 ([Sha92])**.** *$\#SAT \subseteq$ IP.*

Let #SAT be the number of satisfying assignments of Boolean formula ϕ.

$$\#\text{SAT} = \{\langle \phi, k \rangle \mid k = \#\phi\}$$

Proof. #SAT $\subseteq$ IP [Sip97][Chapter 10].
[Sipa]Assume ϕ has m variables $(x_1, ..., x_m)$, let $\phi(a_1, ...a_i)$ be ϕ with $x_1 = a_1, \dots, x_i = a_i$ for $a_1, ..., a_i \in \{0,1\}$. Then we call $a_1, ..., a_i$ presets, the remaining $x_{i+1}, ..., x_m$ stay as unset variables. Then equivalently:

$$\#\phi(a_1, ..., a_i) = \sum_{a_{i+1},...,a_m \in \{0,1\}} \phi(a_1, ..., a_m)$$

$$\#\phi(a_1, ..., a_i) = \#\phi(a_1, ..., a_i, 0) + \#\phi(a_1, ..., a_i, 1)$$

We can apply the sum-check protocol on $\#\phi$ if ϕ is a low-degree polynomial. By going gate-by-gate through ϕ, we can replace each gate with the gate's arithmetization:

$$\text{NOT}(x) \to 1 - x$$

$$\text{AND}(x, y) \to x \times y$$

$$\text{OR}(x, y) \to x + y - x \times y$$

To complete the proof of the theorem, we need only show that V operates in probabilistic polynomial time, let's use $g(x_1, ..., x_m) : \{0,1\}^m \to \{0,1\}$ to denote the low-degree polynomial of $\#\phi$. For $0 \leq i \leq m$ and for $a_1, ..., a_i \in \{0,1\}$, let

$$g(a_1, ..., a_i) = \sum_{a_{i+1},...,a_m \in \{0,1\}} \phi(a_1, ..., a_m)$$

For Boolean formula ϕ with S gates, where each gate can affect the overall degree at most 2 ($\text{AND}(x, y) = x \times y$). In the worst-case scenario (where there are all AND gates in ϕ), the boolean formula ϕ, which is tree-like with each gate having a single output path and no shared sub-expressions, the degree of the polynomial representing ϕ can be up to 2^D, where D is the depth of the formula. For a balanced formula, the depth D is in $\mathcal{O}(\log S)$, therefore, the degree of g is at most $2^{\log S} = \mathcal{O}(S)$. So for V, in each round:

Communication Cost: Since polynomial $g : \{0,1\}^m \to \{0,1\}$ can be represented by its coefficients, requiring at most $\mathcal{O}(S)$ elements (the degree of g), P sends a polynomial s of degree equal to g to V, which is in size $\mathcal{O}(S)$, and V sends one field element r in each round. The total communication cost is $\mathcal{O}(m \cdot S)$.

V Time Complexity: It takes $\mathcal{O}(S)$ time for V to process each of m messages (s in degree $\mathcal{O}(S)$) of P, and $\mathcal{O}(S)$ time to evaluate $g(r)$. The total cost is $\mathcal{O}(m \cdot S)$. □

In other words, the non-existence of certain objects (e.g., a Hamiltonian path) can be certified interactively without requiring an exhaustive search over all possibilities by V using the interactive sum-check protocol.

5.8 Limitation of the Sum-Check Protocol

However, while V's runtimes scales linearly with the circuit size, the practical limitation lies in the **P's runtime**. In each of the m rounds, P must compute the univariate polynomial g_i by summing over all 2^{m-i} possible assignments of the remaining $m - i$ variables, since g has degree S, P's runtime complexity becomes $\mathcal{O}(Sm \cdot 2^m)$. This exponential time requirement makes the sum-check protocol impractical for even relatively simple problems, as P cannot feasibly perform the necessary computations.

6 General Purpose Interactive Proof Protocol

6.1 Introduction to the GKR Protocol

6.1.1 Recall: The Notion of Interactive Proofs in 1980s

Recall Interactive Proof Systems in 80s[GMR85], when the IP model first came out, it was only a theoretical model, which means no one cares about the runtime of the all-powerful P. At that time, the idea[Kal] was people want to see how expressive, which computation can P prove to a polynomial time V by using interactive proofs.

6.1.2 Delegating Computation: Interactive Proofs for Muggles

In the [GKR08] paper, the author introduced a protocol that can be used to **effectively** for both P and V to prove/verify the correctness of **general purpose computation**. More formally: For any question/language computable by a boolean circuit C with depth d and input length n, the protocol can ensure:

- The costs to V grow linearly with the depth d and input size n of the circuit, and only logarithmically with size S of the circuit.
- **P's running time is polynomial in the input size n.**

6.1.3 Blueprint of the Protocol

Layered Circuit: The protocol divides circuit C into d phases, since for each phase i, if its gates value is wrong, then some gates' value in phase $(i+1)$ must be wrong. More specifically, some gates that are connected to the error gates in layer $i+1$ must be incorrect.

Local Correctness: We run a local **sum-check protocol** at each phase/layer to ensure **local correctness** by defining a function $V_i : F^{s_i} \to F$ (where $s_i := logSi$, the bits to represent # gates in layer i), for any gate g_i in that layer i, running $V_i(g_i)$ will give us the corresponding gate value of g_i. By running d-subprotocols of each layer, where each protocol show connections between layer i and its layer before $(i+1)$. (More specifically, if V_i is not correct, then w.h.p V_{i+1} is not correct). **Eventually, it will be reduced to a claim about the input values (layer d), which are known to V.**

6.2 Notations of the GKR Protocol

Assume without loss of generality, circuit C has d layers, which means that each gate belongs to a layer, and each gate in layer i is connected by neighbors (determined) only in layer $i+1$. In a nutshell, the goal of the GKR protocol introduces a layered approach to circuit verification, significantly reducing P's runtime to polynomial time.
Notations: [Kal24]Convert C to a layered arithmetic circuit with fan-in 2 with layer d, and only consists of gates of the form **ADD** and **MULT**. fan-in 2 means each gates in the i-th layer takes inputs from two gates in the (i+1)-th layer. We denote the number of gates in layer i as S_i, and let s_i to be the number of input elements of layer i. ($s_i = \log S_i$) We define function $V_i(z_i)$ at each layer i to return the value of that gate with index z_i[5]:

$$V_i : H^{s_i} \to F \textbf{ where } H = \{0, 1\} \textbf{ and } s_i = \log S_i$$

[5]In practice, we often choose $m = \frac{\log S_i}{\log logS_i}$ and $H = \{0, 1, ..., \log S - 1\}$ to ensure F^{s_i} is not super-polynomial in size, for simplicity, here we use binary value as input.

We designate V_0 as corresponding to the output of the circuit, and V_d as corresponding to the input layer. To define V_i for $1 \leq i \leq d$, recall the function W from Section 4.4.3. Consider layer i of the circuit C as a vector of S_i gates:

$$g = (g_1, g_2, \ldots, g_{S_i}),$$

where each g_j represents the value of the j-th gate in that layer.
Define the function $V_i : H^{s_i} \to F$ as follows:

$$V_i(z) = \begin{cases} g_{a(z)} & \text{if } a(z) \text{ is a valid gate in layer } i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that for every $p \in H^{s_i}$, we can express $V_i(p)$ in the circuit as:

$$V_i(p) = \sum_{w_1, w_2 \in H^{s_{i+1}}} (add_i(p, w_1, w_2) \times V_{i+1}(w_1) + V_{i+1}(w_2)) + (mult_i(p, w_1, w_2) V_{i+1}(w_1) \times V_{i+1}(w_2))$$

where $add_i(mult_i)$ takes one gate label $p \in H^{s_i}$ of layer i and two gate labels w_1, $w_2 \in H^{s_{i+1}}$ in layer $i+1$, and outputs 1 if and only if gate p is an addition (multiplication) gate that takes the output of gate w_1, w_2 as input, and otherwise 0.

Protocol 3 Constructing Multilinear Extensions of V_i at Layer i

In the i-th phase ($1 \leq i \leq d$): P runs a local protocol with V to argue for the correctness of V_i. To do this, a local **sum-check protocol** of layer i will be applied, however, recall that the sum-check protocol only works if the expression inside the sum is a low-degree (multi-variate) polynomial, so let's try to convert $V_i(p)$ to corresponding **multilinear extensions** by leveraging the Lagrange basis polynomial introduced in section 4.4.3, for $z \in F^{s_i}$:

$$\widetilde{V_i}(z) = \sum_{p \in H^{s_i}} \widetilde{B}(z, p) \times \widetilde{V_i}'(p)$$

Where $\widetilde{V_i}'(z)$: $F^{s_0} \to F$ refers to the LDE of $V_i(p)$:

$$\widetilde{V_i}'(z) = \sum_{w_1, w_2 \in H^{s_{i+1}}} (\widetilde{add_i}(z, w_1, w_2) \times (\widetilde{V_{i+1}}(w_1) + \widetilde{V_{i+1}}(w_2))) + (\widetilde{mult_i}(z, w_1, w_2) \times \widetilde{V_{i+1}}(w_1) \times \widetilde{V_{i+1}}(w_2))$$

Let $\widetilde{add_i}, \widetilde{mult_i} : F^{s_i + 2s_{i+1}} \to F$ be the LDEs of add_i and $mult_i$, respectively. By replacing $\widetilde{V_i}'(p)$ to its definition, we can get for every z_i in F^{s_i}, let $f_{i,z_i} : H^{s_i + 2s_{i+1}} \to F$ where:

$$f_{i,z_i}(p, w_1, w_2) = \widetilde{B}(z_i, p) \times [(\widetilde{add_i}(p, w_1, w_2) \times (\widetilde{V_{i+1}}(w_1) + \widetilde{V_{i+1}}(w_2))) + (\widetilde{mult_i}(p, w_1, w_2) \times \widetilde{V_{i+1}}(w_1) \times \widetilde{V_{i+1}}(w_2))]$$

Then $\widetilde{V_i}(z_i) : F^{s_0} \to F$ can be expressed as:

$$\widetilde{V_i}(z_i) = \sum_{p \in H^{s_i}} \sum_{w_1, w_2 \in H^{s_{i+1}}} f_{i,z_i}(p, w_1, w_2)$$

6.3 GKR Protocol at Layer 0

To begin with, let's describe the local sum-check protocol at layer 0 (the output layer) to show the idea of the GKR protocol. Recall:

- **Layer 0 (Output Layer):** Contains a single gate z_0.
- **Layer 1:** Contains gates w_j in $H^{s_{i+1}}$, each associated with their LDEs $\widetilde{V_{i+1}}(w_j)$.

In layer 0, P and V's task is to reduce the claim of $\widetilde{V_0}(z_0) = r_0$ to the claim of $\widetilde{V_1}(z_1) = r_1$, where $z_1 \in F^{s_1}$ is a random value selected by V in layer 1.

Protocol 4 Sum-check Protocol at Layer 0

1. **Initial Claim:** P claims that $\widetilde{V_0}(z_0) = r_0$, according to the definition of $\widetilde{V_0}$, r_0 is the output of the circuit C.
2. **Round 1:** P computes a univariate polynomial $g_1(w_1) : F^{s_1} \to F$ defined as:

$$g_1(w_1) = \sum_{w_2 \in H^{s_1}} f_{0,z_0}(z_0, w_1, w_2)$$

   Which encapsulates the partial sum over w_2 for each fixed w_1 in layer 1.

   P then sends g_1 to V, V needs to ensure that the total sum over all w_1 equals r_0, this is done by verifying that:

$$\sum_{w_1 \in H^{s_1}} g_1(w_1) \equiv r_0$$

   V then selects a random value $z_1 \in F^{s_1}$ and sends to P.
3. **Final Round:** P computes a univariate polynomial $g_2(w_2) : F^{s_1} \to F$ defined as:

$$g_2(w_2) = f_{0,z_0}(z_0, z_1, w_2)$$

   P then sends g_1 to V, V then picks a random value $z_2 \in F^{s_1}$ and verify the correctness of g_2 on her own: Let's recall the definition of $f_{i,z_i} : F^{s_i + 2s_{i+1}} \to F$ then $f_{0,z_0}(z_0, z_1, z_2)$ can be defined as:

$$\widetilde{B}(z_0, z_0) \times [(\widetilde{add_i}(z_0, z_1, z_2) \times (\widetilde{V_{i+1}}(z_1) + \widetilde{V_{i+1}}(z_2))) + (\widetilde{mult_i}(z_0, z_1, z_2) \times \widetilde{V_{i+1}}(z_1) \times \widetilde{V_{i+1}}(z_2))]$$

   V can compute on her own $\widetilde{add_i}$ and $\widetilde{mult_i}$ on $(z_0, z_1, z_2) \in F^{s_0+2s_1}$ (as well as other $(p_0, w_1, w_2) \in H^{s_0+2s_1}$ to check the correctness of the circuit). So as $\widetilde{B}$.

   The computational burden in this verificational task is computing $\widetilde{V_1}(z_1)$ and $\widetilde{V_1}(z_2)$, since they related to the value of gates in layer $2, 3, ...d$.

   So instead, in this protocol, P sends both these values $r_{1,1} = \widetilde{V_1}(z_1)$ and $r_{1,2} = \widetilde{V_1}(z_2)$ to V, and claim they are true. And then using the following interactive reduction Protocol 5 to "reduce" two claims into a single claim $r_1 = \widetilde{V_1}(z_1')$ used in layer 1.

Here we show the Protocol 5 which encodes two claims into a single polynomial, which efficiently reduces the verification burden across multiple layers. The reduction protocol combines the claims $r_{1,1} = \widetilde{V_1}(z_1)$ and $r_{1,2} = \widetilde{V_1}(z_2)$ into a single polynomial $f(r)$, verified by V through random challenge r, and in the next round P is left to prove the correctness of $f(r) = r_1 \equiv \widetilde{V_1}(\gamma(r))$, which is a single claim.

Protocol 5 Reduction Protocol at Layer 0 [GKR15]

1. **Initial Claims:** At the end of layer 0, P claims that:

$$r_{1,1} = \widetilde{V_1}(z_1) \text{ and } r_{1,2} = \widetilde{V_1}(z_2)$$

   Both P and V agree on two distinct fixed elements $t_1, t_2 \in F$.

2. **Define Linear Map:** V defines the unique linear polynomial $\gamma : F \to F^{s_1}$ such that:

$$\gamma(t_1) = z_1 \quad \text{and} \quad \gamma(t_2) = z_2$$

   This linear map effectively creates a line connecting z_1 and z_2 in the field F^{s_1}.

3. **P Sends f to V**: P constructs and sends univariate polynomial $f(t) : F \to F$ to V that passes two points $(t_1, r_{1,1})$ and $(t_2, r_{1,2})$ defined as:

$$f(t) = \widetilde{V_1}(\gamma(t)) = \widetilde{V_1}(z_1) \text{ at } t = t_1 \text{ and } \widetilde{V_1}(z_2) \text{ at } t = t_2$$

4. **V Checks f:** V checks that f indeed pass those two claimed points:

$$f(t_1) = r_{1,1} \quad \text{and} \quad f(t_2) = r_{1,2}$$

5. **Random Challenge by V:** If f is also in correct degree, V can argue the correctness of $f(t) = \widetilde{V_1}(\gamma(t))$ by choosing a random element $r \in F$ and sends it to P.

6. **Verification at Layer 1:** P and V sets:

$$z_1' := \gamma(r) \quad \text{and} \quad r_1 := f(r)$$

   and initiates a Sum-check protocol at layer 1 to verify the correctness of single claim:

$$r_1 = \widetilde{V_1}(z_1')$$

Soundness of the Reduction Protocol: In our example at layer 0, the protocol proceeds to layer 1 where a new Sum-check protocol is initiated to verify the correctness of $\widetilde{V_1}(z_1)$. And just like previous round, in layer 1, V examines the structure of the circuit to ensure that $\widetilde{V_1}(z_1)$ is correctly formed computed based on the circuit's specifications.

Let us say if P attempts to deceive by constructing an incorrectly $f'(t)$ that only satisfies $f'(t_1) = r_{1,1}$ and $f'(t_2) = r_{1,2}$, it is highly unlikely that $f'(t)$ will also satisfy $f'(r) = \widetilde{V_1}(\gamma(r))$ for the randomly chosen r. The reason is due to the random selection of r by V, which ensures that P cannot predict r in advance to tailor $f'(r)$ accordingly, and according to Schwartz-Zippel Lemma [Sch80], if $f'(r)$ does not correctly represent $\widetilde{V_1}(\gamma(r))$, the probability that it passes the check at a random r is negligible for large $|F|$. In other words, if the value r_1 that $r_1 = f(r)$ is consistent according to the sum-check protocol on $r_1 = \widetilde{V_1}(\gamma(r))$ at layer 1, since r is randomly selected by V, this means $f(t)$ is correctly formed with definition $(f(t) = \widetilde{V_1}(\gamma(t)))$. And since f passes two points $(t_1, r_{1,1})$ and $(t_2, r_{1,2})$, we can get that:

$$r_{1,1} = f(t_1) = \widetilde{V_1}(z_1) \text{ and } r_{1,2} = f(t_2) = \widetilde{V_1}(z_2)$$

are both correct with high probability.

6.4 GKR Protocol at Layer d and the Final Check

In layer d, which is very similar to previous phases. P wants to convince V that $r_d = \widetilde{V_d}(z'_d)$, and at the end of the protocol in this layer, P will send $g(z) : F^{s_d} \to F$ **which refers to the low-degree polynomial of the input**, and V can verify on her own. If all the input matches, this means function g is correctly formed, thus according to the Sum-check protocol $\widetilde{V_d}$ is also correctly formed, and in the previous layer, $\widetilde{V_{d-1}}$ is also valid etc. Thus, according to the Soundness of the Sum-Check Protocol, especially Backward Reasoning, we can get V_0 is correctly formed, which implies $C(x) = r_0$ should be correct w.h.p.

6.5 Analyzing P's Runtime Complexity

The traditional Sum-Check Protocol is a foundational component in interactive proof systems, enabling the verification of polynomial evaluations over exponentially large domains. Specifically, recall Section 5.8, in each of m rounds, P must compute a univariate polynomial $s_i(x_i)$ by summing over all 2^{m-i} possible assignments of the remaining $m-i$ variables. This results in a runtime complexity of $\mathcal{O}(Sm \cdot 2^m)$ for P, where S denotes the degree of the polynomial. Such exponential time requirements make the traditional Sum-Check Protocol impractical for large-scale computations.

In contrast, the **GKR Protocol** leverages the structured nature of layered arithmetic circuits to achieve improved efficiency. Each layer i of the circuit comprises S_i gates with a fixed fan-in of 2, performing either addition or multiplication. Recall Protocol 3, the gate function $\widetilde{V}_i(z_i)$ is defined as:

$$\widetilde{V}_i(z_i) = \sum_{p \in H^{s_i}} \sum_{w_1, w_2 \in H^{s_{i+1}}} f_{i,z_i}(p, w_1, w_2)$$

where H represents the set of possible gate indices, and f_{i,z_i} encapsulates the computation performed by each gate in layer i. the GKR Sum-Check Protocol aggregates contributions gate-by-gate within each layer, which means considering all gates in layer $i+1$, and this is feasible because the values of gates in layer $i+1$ are precomputed and stored. As a result, P's runtime at each layer i becomes $\text{poly}(S_i)$, where S_i is the number of gates in that layer. This polynomial-time complexity starkly contrasts with the exponential runtime of the traditional Sum-Check Protocol, underscoring the practicality of the GKR Protocol for delegating and verifying large computations. By capitalizing on the structural properties of layered arithmetic circuits, the GKR Protocol ensures that P can efficiently perform verification tasks, maintaining scalability even as the size of the computation grows.

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160:781–793, 2004. doi:10.4007/annals.2004.160.781.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018. doi:10.1109/SP.2018.00020.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014. doi:10.1109/SP.2014.36.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

[Fre77] Rusins Freivalds. Probabilistic Machines Can Use Less Running Time. *IFIP Congress 1977*, pages 839–842, 1977.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122. ACM, 2008. doi:10.1145/1374376.1374396.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. doi:10.1145/2699436.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985. doi:10.1145/22145.22178.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

[Kal] Yael Kalai. Gkr based zero-knowledge proofs. URL: https://www.youtube.com/watch?v=x8pUxFptfb0.

[Kal24] Yael Kalai. The gkr protocol, 2024. Lecture 12 Notes of Class MIT 6.5616; accessed 31-December-2024. URL: https://65610.csail.mit.edu/2024/lec/l12-gkr.pdf.

[Lev73] Leonid A. Levin. Some theorems on the algorithmic approach to probability theory and information theory. *CoRR*, abs/1009.5894, 1973. URL: http://arxiv.org/abs/1009.5894, arXiv:1009.5894.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 2–10 vol.1, 1990. doi:10.1109/FSCS.1990.89518.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013. doi:10.1109/SP.2013.47.

[Sch80] J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM Vol.27, No.4*, pages 707–717, 1980. doi:10.1145/322217.322225.

[Sha92] Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.

[Sipa] Michael Sipser. Interactive proof. MIT 6.840, Lecture 25, Interactive Proof, 2020; accessed 31-December-2024. URL: https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/resources/mit18_404f20_lec25/.

[Sipb] Michael Sipser. Np-completeness. MIT 6.840, Lecture 15, NP-Completeness, 2020; accessed 31-December-2024. URL: https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/0d814f5aba35f291f0253b7b34af2868_MIT18_404f20_lec15.pptx.

[Sipc] Michael Sipser. Np-completeness. URL: https://youtu.be/iZPzBHGDsWI.

[Sipd] Michael Sipser. P, np. MIT 6.840, Lecture 14, P, NP, 2020; accessed 31-December-2024. URL: https://ocw.mit.edu/courses/18-404j-theory-of-computation-fall-2020/resources/mit18_404f20_lec14/.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[Tha19] Justing Thaler. Interactive proofs, 2019. Presented at the Proofs, Consensus, and Decentralizing Society Boot Camp, Simons Institute for the Theory of Computing, August 2019; accessed 31-December-2024. URL: https://people.cs.georgetown.edu/jthaler/JTBootCamp.pdf.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, pages 117–660, 2022. doi:http://dx.doi.org/10.1561/3300000030.

[WXXZ24] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New Bounds for Matrix Multiplication: from Alpha to Omega. *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3895, 2024. doi:10.1137/1.9781611977912.134.