Computational Analysis of Plausibly Post-Quantum-Secure Recursive Arguments of Knowledge

Dustin Ray[1][0009−0002−1782−5773]

University Of Washington

Abstract. With the recent standardization of post-quantum cryptographic algorithms, research efforts have largely remained centered on public key exchange and encryption schemes. Argument systems, which allow a party to efficiently argue the correctness of a computation, have received comparatively little attention regarding their quantum-resilient design. These computational integrity frameworks often rely on cryptographic assumptions, such as pairings or group operations, which are vulnerable to quantum attacks. In this work, we present a fully implemented post-quantum secure argument system that compresses unbounded computation into a constant-sized space. We present a fully implemented prover which can argue the truth of any size computation, and verifier which can verify correctness in constant time. This work shows an extension of utility for computational integrity statements into the quantum domain. We provide real-world performance metrics demonstrating that post-quantum secure argument systems not only exist but can outperform classical systems in both efficiency and scalability, making such systems an attractive choice for practical deployment.

Keywords: Post-Quantum Cryptography, Recursive Arguments of Knowledge, Zero-Knowledge Proofs, Reed-Solomon Error-Correcting Codes

1 Introduction

In 1995, Peter Shor introduced what is now known as Shor's algorithm [1], demonstrating that under certain conditions, powerful adversaries can efficiently factor large primes and solve discrete logarithms. The wide-reaching implication of this discovery led to a period of rapid development of stronger primitives which could resist such adversaries.

Since Shor's pivotal discovery, a wide range of post-quantum cryptographic constructions, such as lattice, code, or isogeny-based constructions have been proposed[2], implemented, broken[3], fixed[4], standardized[5], and deployed. Most of these efforts have focused on replacing traditional cryptographic tools like public-key encryption, key exchange schemes, and digital signatures with quantum-resilient alternatives.

Argument systems, including zero-knowledge proofs and computational integrity frameworks, have seen relatively little attention in the post-quantum

space. Many existing argument systems[6][7][10] rely on pairings and group operations, which are based on hard problems[11] vulnerable to quantum attacks under Shor's algorithm.

Quantum-resilient argument systems are known and have been deployed in practice [12][13], showing promising performance and security asymptotics. In this work, we present an implementation of a post-quantum secure argument system that achieves constant-sized arguments, constant-time verification, and updateable argument states, all while offering state-of-the-art performance for both the prover and verifier. Through this evaluation, we demonstrate the protocol's readiness for real-world deployment and provide optimizations for practical use in environments requiring efficient, secure verification.

We emphasize that our aim is to provide a practical implementation that can be constructed using existing, well-established cryptographic primitives and frameworks. Specifically, we do not propose a *quantum argument system* based on quantum algorithms or quantum primitives. Instead, our work focuses on security against a theoretical quantum-equipped adversary, whose existence and capability to break current systems is plausible but not guaranteed. While there is a growing body of research into quantum argument systems that leverage quantum algorithms and technologies, these systems are beyond the scope of our work. Our focus remains on argument systems that operate in *probabilistic polynomial time* (PPT), designed to run on modern, classical hardware, while providing post-quantum security properties.

1.1 Relevant Concepts and Nomenclature

This work draws from an active field of cryptography focused on the study and application of argument systems, particularly those that are probabilistically-checkable and non-interactive. The celebrated results of [14] introduced an argument system in which a prover (P) convinces a verifier (V) that a computation was performed correctly, with high probability. Rather than re-running the entire computation, the verifier only needs to check a logarithmic-sized, non-deterministic sample of the prover's argument. With overwhelming probability, this sampling is sufficient to ensure the correctness of the result.

In the literature on zero-knowledge (ZK) proofs, a computational integrity statement is often referred to as a "proof," and the party generating it is the "prover." However, when perfect soundness is not required, these statements are more accurately referred to as *arguments*, as they do not prove the truth of a statement with certainty, but instead argue its correctness with high probability. The term "prover" refers to the party generating the computational integrity statement. While the prover possesses a complete proof of the computation's correctness, the system optimizes asymptotic complexity by transforming this proof into a more efficient argument. The term "prover" thus reflects the role of the party producing these arguments, which are more practical for large-scale computations. Throughout this work, we use the terms "zero-knowledge proof," (ZKP) "computational integrity statement," (CI statement) and "argument" interchangeably.

SNARKs (Succinct Non-Interactive Arguments of Knowledge) are a class of argument systems widely used for their efficiency, particularly in applications requiring succinctness and non-interactivity. SNARKs often rely on cryptographic primitives such as pairing-based group operations, elliptic curve assumptions, or trusted setups that may be vulnerable to quantum attacks[10]. For instance, the reliance on discrete logarithm or pairing-based cryptographic assumptions, both of which are broken by Shor's algorithm, suggests that these SNARKs may be insecure against a quantum adversary. When we refer to SNARKs in this paper, we are generally referring to argument systems that lack quantum resistance, rely on group operations, require a trusted setup phase, or exhibit any combination of these features.

STARKs (Succinct Transparent Arguments of Knowledge)[13] form the backbone of this research effort. In general, these systems do not rely on assumptions that are currently known to be susceptible in the presence of quantum adversaries. We elaborate in detail the specific inner workings of STARKs in further sections. Similar to the terminology surrounding SNARKs, when we refer to a STARK, we are referring to an argument system which is quantum-resilient and has no requirement for trusted setups.

Throughout this work, when we refer to a cryptographic scheme or system as "quantum secure" or "post-quantum secure," we are specifically referring to what is more accurately termed "plausibly post-quantum secure." This terminology acknowledges that while the cryptographic assumptions underlying these systems are believed to resist quantum attacks, there is currently no concrete proof of absolute security against a quantum adversary. As with most cryptographic systems, security is based on the hardness of specific mathematical problems, which remain assumptions until rigorously proven or disproven.

1.2 Organization

The remainder of this paper is organized as follows:

- *Section 2* provides the broader context and challenges in quantum-resilient cryptography, introducing key concepts like zero-knowledge proofs and computational integrity, and explaining the gap this work addresses.
- *Section 3* reviews relevant work in cryptographic argument systems, analyzing existing methods and identifying research gaps that this paper seeks to fill.
- *Section 4* defines the asymptotic qualities of the protocol we wish to implement. This section forms the main foundation of this research effort in that it explicitly constructs the theoretical backbone of the construction. Finally, this section formally prescribes our construction with the features commonly employed in argument systems, such as completeness, soundness. etc.
- *Section 5* presents the mathematical foundations of the protocol, including polynomial commitment schemes, Fast Reed-Solomon IOP (FRI), Merkle trees, Reed-Solomon error-correcting codes, and their relevance to quantum security.

- *Section 6* details the design and implementation of the proposed argument system, covering the framework, tools, and technical decisions made to build a secure and efficient protocol. It also discusses the system architecture and the choice of an ASIC-based design over other frameworks.
- *Section 7* evaluates the performance of the implemented system, including benchmarking results, resource consumption, and scalability tests. Key performance metrics such as constant-sized proof generation and runtime efficiency are presented.
- *Section 8* compares the proposed system with other state-of-the-art cryptographic argument systems, drawing both quantitative and qualitative comparisons to demonstrate the advantages of the proposed scheme, particularly in terms of post-quantum security and asymptotic performance.
- *Section 9* summarizes the main contributions of this work and discusses potential future directions. This section outlines how the proposed argument system can be extended or adapted in future research.

2 Motivation

Argument systems, including zero-knowledge proofs and computational integrity statements, allow one party to convince another with high probability that a computation was carried out correctly and faithfully, without needing to rerun the entire computation. These systems have become crucial in scenarios where computation is delegated to untrusted parties, ensuring trust without direct oversight or interaction.

These cryptographic constructions have seen widespread adoption in recent years[7], particularly in cloud computing, blockchain applications, and distributed systems. In these settings, computations are often outsourced to potentially untrusted parties, and ensuring the integrity of the results is paramount. For example, in cloud computing, users must be able to trust that outsourced computations were performed correctly without needing to replicate the work themselves. Similarly, in blockchain applications, computational integrity underpins the trustless nature of decentralized systems, where participants must verify state transitions without direct access to the underlying data.

Further examples relevant to our design include computations with a potentially unbounded number of steps or invocations. For instance, consider a machine-learning model that must update itself every time new data is added to the training set. The cost of retraining the model from scratch only to include the effect of the new data element in the model weights would be prohibitively expensive. Our argument system implementation is particularly well-suited for recursive and updateable computations, where starting the algorithm from the first step would represent a prohibitive amount of work to be done.

Most argument systems require a commitment phase, during which the proving party performs an operation in addition to the computation to be verified. This commitment operation acts as a "seal" that cannot be altered without causing the argument to fail verification with high probability. For large computations that need to update in response to new input, we leverage recursion to

simulate an *updateable* commitment. This allows the proving party to efficiently update the existing argument from the previous state in response to a changing input space, rather than recompute the entire argument from the beginning. Addressing the unbounded nature of certain computations places our approach among the most versatile class of such functions in terms of scalability, and assessing the quantum resilience of such systems is identified as a gap in existing research.

The implementation we present is ideally suited for computations that are continually updated. Formally, our system targets general recursive, Turing-computable functions in $\mathcal{P}$. We demonstrate a quantum-resilient argument system that effectively secures computations in this class, while achieving optimal asymptotic performance. The system produces constant-sized arguments, constant-time verification, and updateable argument states, making it both efficient and scalable for real-world applications.

2.1 Contributions

In this work, we make the following contributions:

- *Empirical analysis of post-quantum security*: We provide a detailed empirical analysis of the post-quantum security of various components of our argument system, and conduct a survey of commonly deployed argument systems and their potential quantum resilience.
- *A proof circuit that supports unbounded recursive computations*: Our circuit is capable of verifying itself any number of times, and thus produces a constant-sized proof, regardless of the computation's depth.
- *Efficient performance benchmarking*: We provide a detailed evaluation of the system's runtime for each layer of the circuit, demonstrating its scalability and efficiency.
- *Flexible Design*: Our circuitry is designed to be computation agnostic. It has input and output gates that can be connected to a circuit representation of any computation. We find that this greatly simplifies the construction of recursive computation.
- *Constant-sized circuit*: We analyze the number of gates in the circuit, showing that the circuit size remains constant as the number of computation invocations grows.
- *Asymptotic performance*: We highlight how this scheme exhibits some of the best-performing asymptotic characteristics in the current landscape of post-quantum argument systems.

2.2 Existing Limitations

While several cryptographic argument systems have been proposed and theoretically analyzed, significant limitations exist in both their practical implementations and their readiness for deployment in post-quantum secure environments. These limitations can be broadly categorized as follows:

- *Quantum Vulnerability*: Many of the widely used and deployed argument systems rely on cryptographic assumptions that are vulnerable to quantum attacks, such as pairings or group operations[15]. With quantum computing advancing, these systems are at risk of being compromised, highlighting a need for protocols that offer post-quantum security. Our implementation considers quantum resilience at every step of the argument, relying only on assumptions conjectured to be quantum-resilient. We discuss the precise threats that these constructions face from a quantum adversary in further sections.
- *Lack of Practical Implementations*: Although post-quantum secure argument systems have been proposed theoretically, few have been implemented and rigorously tested in real-world environments. This gap creates uncertainty about their performance, scalability, and practicality. Existing recursive systems like zk-SNARKs have received significant attention, but post-quantum alternatives remain underexplored in practical contexts. Our work addresses this by implementing and empirically evaluating a fully functional post-quantum secure argument system, providing insights into its performance and scalability in realistic scenarios.
- *High Computational Burden*: Some protocols[6] place a heavy computational load on the prover or verifier, requiring substantial resources to generate or verify proofs, respectively. This limits their usability in resource-constrained environments such as embedded systems or devices with limited processing power. Our implementation reduces the computational burden on the prover and verifier by optimizing the proof generation and verification process, making it feasible for use in resource-constrained environments. We also reduce the burden on circuit design by abstracting away the recursive circuitry.
- *Proof Size and Verification Complexity*: Some argument systems produce large proofs [6], which are costly to store and transmit, especially in bandwidth-constrained environments. Additionally, verification complexity can scale with the size of the computation[17], limiting the feasibility of using these systems in real-time applications or large-scale deployments. Even widely-used systems such as Groth16[8][10], known for its succinct proofs, still require public parameters and auxiliary data of substantial size, often measured in gigabytes (GB). Our implementation requires no such parameters, and generates short proofs relative to computation size.

These limitations emphasize the need for cryptographic argument systems that are both post-quantum secure and efficient in practice. In particular, the lack of empirical performance evaluations for post-quantum secure protocols leaves a gap in the understanding of their real-world applicability and scalability.

3 Literature Review

3.1 Key Previous Works

[12] **Ligero** The Ligero protocol introduces lightweight sublinear arguments that do not require a trusted setup. It is designed to balance between efficiency and

scalability, addressing the need for transparent cryptographic proofs. This approach allows for efficient verification while maintaining security, making it applicable in various zero-knowledge proof systems. The design leverages the advantages of both succinctness and transparency, catering to the growing demand for post-quantum secure cryptographic protocols.

Ligero is one of the first argument systems to use error correcting codes in its commitment scheme. This advancement leads directly to argument systems and multi-party computation protocols which operate without opaque setup phases and weaker cryptographic assumptions.

Our work leverages a commitment scheme known as FRI, which was directly inspired from the advancements in Ligero. FRI is formally defined in further sections.

[17][18] **Halo2** Halo2 is among the first and widest used recursive arguments, originally deployed as a privacy and scaling solution targeting blockchains. It is one of the first examples of recursive proof composition leveraging cycles of elliptic curves, in which the scalar group of one curve is the finite field of another, and visa versa. This recursive relationship facilitates short proofs and eliminates the need for a trusted setup phase.

[19] **Plonk** "Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge" is a SNARK invented to mitigate a limitation of traditional trusted setups or structured reference strings (SRS). Traditionally, the SRS is limited to a single computation description. Should a distinct or differing computation need to be argued, the setup phase must be repeated each time a change is made. Plonk resolves this redundancy by introducing the notion of a "Universal Setup", which can be easily updated for new computations as needed, and with little overhead.

A further key development from Plonk is a new format in which circuits are described. Known as "plonk-ish" arithmetization, this wiring syntax closely resembles that of physical hardware, which significantly improves over previous methods the experience of describing the computation to be proven. Our research makes use of plonkish arithmetization in the design and construction of our circuit.

[20] **Plonky2** Plonky2 is an argument construction framework which supports plonkish-arithmetization paired to the FRI commitment scheme. This argument is a STARK which exhibits several quantum-resilient properties of specific interest to our work. Plonky2 further supports recursive argument and circuit design which makes it an effective platform for the design of our circuit.

[21] **Non-recursive SHA-256 Hashing with Plonky2:** This work shows an example of describing a hash function (SHA256) as gates and wires in the plonky2 framework. It implements non-recursive SHA-256 hashing with Plonky2. It provides a basis for the hash-chain benchmarking computation that we encode into our circuit. We leverage implementations of the Poseidon hash function due to its compact description in circuitry.

[23] **Plonky2 Test Code:** This implementation in Plonky2 showcases a two-step, non-recursive hash chain. It is part of the recursion module, which demon-

strates basic hashing functionality without recursion. This approach highlights a simpler use case within the Plonky2 framework for testing purposes, contrasting with more complex recursive proof structures like those using Poseidon or SHA-256. The code provides a foundation for testing and benchmarking hash functions in cryptographic proof systems.

Our approach formalizes this example in recursive circuitry. We provide a modular design with respect to the construction of various components of the hash-chain circuit. We achieve a lower proof size in 2 step recursion compared to 2-steps without recursion due to a reduction in circuit size, and achieve constant size proofs for subsequent invocations of the circuit.

[16] **The Tip5 Hash Function for Recursive STARKs:** The paper introduces Tip5, a new arithmetization-oriented hash function designed for recursive STARK verification. It combines low-degree power maps and lookup tables optimized for field elements. The function is tailored specifically for applications involving the recursive verification of STARKs, which impose stringent constraints on the hash function's design. The authors provide an in-depth analysis of the arithmetization properties of Tip5 to meet these constraints.

The construction of recursive and STARK-friendly hash functions has been a key step in optimization of program logic specifically suited for the format required for recursive argument systems. Tip5 has been key in this respect in that it has informed the design of provable hash functions, such as the Poseidon hash, which is used throughout this work.

3.2 Gaps in Current Research

While significant advances have been made in the development of zero-knowledge proof systems and argument systems, several key areas remain underexplored. This work aims to highlight some of these gaps and address them where possible.

1. **Concrete Implementations with a Focus on Quantum Security**: Many argument systems today are designed based on classical security assumptions, leaving a gap in practical, concrete implementations that focus specifically on quantum resilience. While academic progress has been made, there is a lack of concrete, working systems that can be demonstrated to resist quantum attacks, particularly with respect to key components such as hash functions, commitment schemes, and polynomial evaluations. This paper contributes to bridging this gap by exploring the necessary adaptations for quantum security.
2. **Formal Definitions of Quantum-Resilience**: Although various components of argument systems—such as commitment schemes, polynomial evaluations, and zero-knowledge protocols—are widely used, their quantum-resilient counterparts have not been formally defined in a comprehensive manner. A more rigorous and formal definition of these components, along with an explanation of why they are secure against quantum adversaries, is essential for constructing robust post-quantum cryptographic systems. This research works to establish formal definitions behind cryptographic assumptions involved in a quantum-resilient argument.

3. **ZKVMs and ASICs**: A comprehensive study comparing the performance and efficiency of Zero-Knowledge Virtual Machines (ZKVMs) and circuit-level, ASIC-style argument systems is currently lacking. While ZKVMs offer flexibility and broader applicability, ASIC-style implementations tend to be highly optimized for specific tasks. A survey of the components involved in the construction of ZKVMs and their quantum-resilience is lacking. This paper chooses a quantum-safe ASIC-style design and considers the characteristics of a quantum-safe ZKVM approach.
4. **Disparity in Quantum-Secure Recursive Schemes**: Recursive proof systems, which allow a proof to attest to the correctness of previous proofs, are an emerging area of interest due to their ability to efficiently handle complex, iterative computations. However, there is a notable disparity in the availability of recursive schemes that are quantum-secure. While several classical recursive schemes exist, their adaptation to withstand quantum adversaries remains an underexplored area. This work seeks to highlight this disparity and suggest potential paths forward by focusing on the quantum resilience of recursive components in argument systems.

4 Protocol Definition

4.1 Recursive Hash Chains

The central benchmarking computation used in this work is a recursive hash chain with n steps. At each step $i \in \{1, 2, \ldots, n\}$, we compute a hash function H such that:

$$h_i = H(h_{i-1}),$$

where h_0 is the initial input. At each step, the previous proof π_{i-1} is verified by computing:

$$V(\pi_{i-1}) \implies \text{Evaluation of } h_{i-1},$$

and a new proof π_i is generated for h_i. This result, h_i, along with its corresponding proof π_i, is then passed to the next recursive layer, continuing until h_n and π_n are produced, completing the recursion.

Recursive STARK-Friendly Hash Description This subsection describes the 4-tuple of functions used in our recursive hash chain design.

$$\begin{aligned}
\mathcal{B}_{(\text{f, c, d})} &\to \mathcal{T}_{\text{Proof}} \\
\mathcal{V}(\mathcal{P}_{\text{proof}}, \mathcal{C}_{\text{cyclic}}) &\to \mathcal{R}(\mathcal{E}_{\text{Accept?}}) \\
\lambda_{\text{cyclic}}(\mathcal{C}_{\text{cond}}, \iota_{\text{inner}}, \mathcal{P}_{\text{proof}}, \mathcal{V}_{\text{data}}, \mathcal{C}_{\text{cyclic}}) &\to \mathcal{R}(\mathcal{P}, \mathcal{E}_{\text{Accept?}}) \\
\mathcal{E}_{\text{RecLayer}}(\mathbb{B}_{\text{cond}}, \iota_{\text{inner}}, \mathcal{D}_{\text{common}}, \mathcal{C}_{\text{cyclic}}, \mathcal{V}_{\text{data}}, s) &\to \mathcal{R}_{\text{ProofCircuit}}
\end{aligned}$$

The functions are as follows:

- $\mathcal{B}_{(\text{f, c, d})} \rightarrow \mathcal{T}_{\text{Proof}}$: Builds a hash chain circuit over a specified number of steps and returns a target wiring for the proof result.
- $\mathcal{V}(\mathcal{P}_{\text{proof}}, \mathcal{C}_{\text{cyclic}}) \rightarrow \mathcal{R}(\mathcal{E}_{\text{Accept?}})$: A constant-time, constant-sized verification algorithm for a single proof aggregate representing the argument for the entire hash chain.
- $\lambda_{\text{cyclic}}(\mathcal{C}_{\text{cond}}, \iota_{\text{inner}}, \mathcal{P}_{\text{proof}}, \mathcal{V}_{\text{data}}, \mathcal{C}_{\text{cyclic}}) \rightarrow \mathcal{R}(\mathcal{P}, \mathcal{E}_{\text{Accept?}})$: Verifies the aggregated cyclic proof from the previous steps, proves the current layer, and updates the aggregate proof.
- $\mathcal{E}_{\text{RecLayer}}(\mathbb{B}_{\text{cond}}, \iota_{\text{inner}}, \mathcal{D}_{\text{common}}, \mathcal{C}_{\text{cyclic}}, \mathcal{V}_{\text{data}}, s) \rightarrow \mathcal{R}_{\text{ProofCircuit}}$: Evaluates the gates in a recursive layer and produces proof and circuit results.

The parameters are defined as follows:

- $\mathcal{P}_{\text{proof}}$: Proof input.
- $\mathcal{C}_{\text{cyclic}}$: Cyclic circuit data.
- $\mathcal{E}_{\text{Accept?}}$: Decision to accept or reject the proof.
- $\mathcal{C}_{\text{cond}}$: Condition (boolean target).
- ι_{inner}: Inner cyclic proof with public inputs, the aggregate proof so far.
- $\mathcal{V}_{\text{data}}$: Verifier data target.
- $\mathbb{B}_{\text{cond}}$: Check for the recursive case.
- $\mathcal{D}_{\text{common}}$: Common data for configuration of layers.
- s: Number of steps in recursion.
- $\mathcal{R}_{\text{ProofCircuit}}$: Proof and circuit result.
- $\mathcal{R}(\mathcal{P}, \mathcal{E}_{\text{Accept?}})$: A pair containing the proof of the current layer and the decision to accept or reject the proof.

The notion of a hash chain extends past academic curiosity. Modern blockchain networks are themselves a form of hash chain, with each new block leveraging the hash of the previous block to create a chain of valid hashes. Verifying the correctness of such chains typically involves reconstructing the chain from scratch. Recursive hash chain arguments can be verified at the cost of a verifying the current state of the chain, which is constant relative to chain size.

Lastly, we choose this computation as a benchmark of our circuit because hashes invoke a complex set of machine instructions and are commonly used in many areas of computing to gain insight into performance metrics.

With a computation defined as a tuple of recursive algorithms, we can begin paramterizing an argument system with desired characteristics, and then proceed to survey existing frameworks and protocols to find a practical design that suits our requirements. Naturally, we begin by considering the asymptotic qualities (and their dependents) of a desired argument system. In probabilistic polynomial time, we strive for the following properties for a robust and efficient argument system under random oracle assumptions: [24]

Completeness: True statements can always be proven by a prover and will always be accepted by a verifier. Formally: For every instance-witness pair (x, w) in a relation R, $\Pr\left[V^p\left(x, P^p(x, w)\right) = 1\right] = 1$ for probability p taken over $\mathcal{P}$ and any randomness from $\mathcal{P}$ or $\mathcal{V}$. For any PPT adversary $\mathcal{A}$ and a tuple of recursive algorithms:

$$\Pr\left[\begin{array}{l}(x,w)\in R,\\ V^p\left(x,P^p(x,w)\right)=1\end{array}\middle|\begin{array}{l}\mathcal{T}_{\text{Proof}}\leftarrow\mathcal{A}(\mathcal{B}_{(\text{f, c, d})}),\\ 1\leftarrow\lambda_{\text{cyclic}}(\mathbb{B},\mathcal{P},\mathcal{P}_{\text{proof}},\mathcal{V}_{\text{data}},\mathcal{C}_{\text{cyclic}}),\\ 1\leftarrow\mathcal{E}_{\text{RecLayer}}(\mathbb{B},\mathcal{P},\mathcal{D}_{\text{common}},\mathcal{C}_{\text{cyclic}},\mathcal{V}_{\text{data}},s)\end{array}\right]=1$$

Soundness: A prover should not be able to deceive a verifier into accepting a false statement as true, except with negligible probability. Formally: For every instance x not in the language of R and every malicious prover $\widetilde{P}$ submitting at most a polynomial number of queries to a random oracle, $Pr\left[V^p\left(x,\widetilde{P}^p\right)=1\right]$ is negligible in the security parameter. [24] shows how a quantifiable lower-bound of security can be derived from the soundness parameter, which confers a configurable level of $n-$bit security directly from the security of the choice of underlying hash function. For a tuple of recursive algorithms, any constant $n\in\mathbb{N}$, and for any PPT adversaries P^*, there exists a PPT extractor $\mathcal{E}$ such that, for all randomness ρ:

$$\Pr\left[\begin{array}{l}z_n\neq z,\\ \mathcal{V}(\mathcal{P}_{\text{proof}},\mathcal{C}_{\text{cyclic}})=1\end{array}\middle|\begin{array}{l}\mathcal{T}_{\text{Proof}}\leftarrow\mathcal{E}(\mathcal{B}_{(\text{f, c, d})}),\\ 1\leftarrow\lambda_{\text{cyclic}}(\mathbb{B},\iota,{\mathcal{P}^*}_{\text{proof}},\mathcal{V}_{\text{data}},\mathcal{C}_{\text{cyclic}}),\\ 1\leftarrow\mathcal{E}_{\text{RecLayer}}(\mathbb{B},\iota,\mathcal{D}_{\text{common}},\mathcal{C}_{\text{cyclic}},\mathcal{V}_{\text{data}},s)\end{array}\right]\leq\text{neg}(\lambda)$$

Succinctness: This work demonstrates the existence of a post-quantum secure argument system that meets or exceeds the asymptotic properties of current computational integrity frameworks. We extend the concept of succinctness to ensure that the size of the argument remains logarithmic in relation to the size of a single computation invocation. By constructing a recursive system that can verify its own argument, we achieve an argument size and verification time that are logarithmic for a single invocation, but remain constant across an unbounded number of iterations.

Updatability: Building on the previous requirement, a key property of the argument system in use is its ability to prove the $n+1$ iteration at the cost of a single invocation. This ensures that the system does not recompute the entire chain of computations when a new step is introduced. Instead, it can start from the verification of the first n steps, and then efficiently update the argument to reflect the next step. Updateable arguments are essential for applications that handle an unbounded number of sequential computations. In such cases, polynomial commitment schemes that are *additively homomorphic* allow commitments to be updated incrementally without recomputing proofs. We can define this notion formally:

Definition 1. *(Recursive Commitment Scheme with Incremental Proof Verification) A recursive commitment scheme is a cryptographic protocol between two parties, the sender S and the receiver R, that allows the verification of earlier steps in a computation or proof to be included in future steps, without the need to recompute the entire process.*

The scheme consists of the following components:

1. **Commit Phase**: The sender S chooses a value $v \in \mathbb{Z}_N$ and computes a commitment c to v using a probabilistic polynomial-time algorithm $\text{Com}(v, r)$, where $r \in \mathbb{Z}_N$ is randomness. The commitment $c = \text{Com}(v, r)$ is sent to the receiver R.
2. **Proof Generation**: For each step i, the sender S can generate a proof π_i for the committed value v_i. The proof includes the verification of any prior proofs $\pi_{i-1}, \ldots, \pi_1$, without the need to recompute all previous proofs from scratch.
3. **Recursive Proof Verification**: The verification process at step i not only verifies the current proof π_i, but also ensures that the commitment c_i is consistent with the commitments $c_{i-1}, \ldots, c_1$ from earlier steps. This is done by incorporating a recursive verification function $\text{Ver}(v_i, \pi_i)$, which checks the validity of π_i given c_i and the proofs from previous steps.

The scheme satisfies the following properties:

- **Recursive Proof Composition**: The scheme allows the proof at step i to efficiently incorporate all prior proofs $\pi_{i-1}, \ldots, \pi_1$ into a single proof π_i, without needing to recompute each step individually. Formally, for any step i, the recursive proof generation is defined as:

$$\pi_i = \text{Prove}(v_i, \pi_{i-1}, \ldots, \pi_1, r_i)$$

  where Prove is the recursive proof generation function, and r_i represents the randomness used at step i.
- **Recursive Verification**: Verification at step i ensures that the proof π_i and all prior steps $\pi_{i-1}, \ldots, \pi_1$ are valid. The verification function can be formalized as:

$$\text{Ver}(v_i, \pi_i) = 1 \iff \forall j \leq i,\ \text{Ver}(v_j, \pi_j) = 1$$

  indicating that the current proof π_i is valid only if all prior proofs $\pi_{i-1}, \ldots, \pi_1$ are also valid.
- **Simulating Updatability**: Although the scheme does not update commitments, it allows the sender S to start proof generation from any valid previous proof π_{i-1}, rather than having to recompute the entire proof chain from the beginning for all n steps. This recursive structure resembles updatability in that the sender can incorporate previously verified steps incrementally, allowing for efficient proof updates. However, the commitments themselves do not change across steps.
- **Binding**: Once S commits to a value v_i and generates a valid proof π_i, it is computationally infeasible to change v_i to another value v_i' without altering the corresponding commitment c_i and proof π_i.

Transparency: Let CRS denote a *Common Reference String*, which is a trusted setup phase featured in many commonly deployed protocols. The purpose of a CRS is to generate shared public parameters, typically instantiated as elements

from a cryptographic group G, such that these parameters are used to ensure the succinctness of argument systems, i.e., producing arguments of size $O(1)$, often as short as two group elements.

However, such a setup often relies on pairing-based cryptographic primitives, which are vulnerable to quantum adversaries. Specifically, let $\mathrm{e} : G_1 \times G_2 \to G_T$ be a bilinear pairing operation over the groups G_1, G_2, and G_T, and let $\mathrm{CRS} = (\alpha G_1, \alpha G_2)$ be the parameters derived from a secret value α. If a quantum adversary can efficiently solve the discrete logarithm problem (DLP) over G_1, G_2, then the soundness of the system is compromised, and the adversary is able to produce false proofs.

To mitigate this vulnerability, STARKs in general do not require any form of trusted setup or (CRS). By removing the reliance on a CRS, our protocol no longer requires the distribution of pairing-based group elements, which inherently enhances the *transparency* of the system. As a result, the absence of a CRS ensures that our system is more resistant to quantum adversaries, as there are no classical group elements, such as G_1 and G_2, upon which quantum attacks, like Shor's algorithm, can be executed. Additionally, this approach eliminates the need to store the CRS, which can often require gigabytes of space even for relatively small computations [7]. In traditional schemes that rely on a CRS, if the computation changes or is updated, the public parameters must be recomputed, which is computationally expensive and inflexible. Our implementation avoids this costly step at the expense of slightly larger argument sizes, offering a more flexible and practical solution.

Perfect Zero-Knowledge: In traditional SNARK setups, perfect zero-knowledge ensures that the proof reveals nothing about the input except the correctness of the result. This property, known as *hiding* or *blinding*, is added trivially in an argument system by perturbing the computation record with random data prior to commitment. If this perturbation is carried out by means of a secure and correctly instantiated pseudorandom function, then blinded commitment data is considered to remain secure against a quantum adversary. We can formally define zero-knowledge as the following:

Definition 2. *Let (P, V) be an interactive proof system for an NP-language L, and let R_L be the associated NP-relation. We say that (P, V) is black-box computational zero knowledge if there exists a probabilistic-polynomial time oracle machine S such that for every non-uniform probabilistic-polynomial time algorithm V^* it holds that:*

$$\{output_{V^*}(P(x,w), V^*(x,z))\}_{(x,w)\in R_L, z\in\{0,1\}^*} \equiv \{S^*(x,z,r')(x)\}_{x\in L, z, r'\in\{0,1\}^*}$$

where $\equiv$ denotes computational indistinguishability.

Our research does not incorporate perfect zero-knowledge, as our focus relies on analyzing the cryptographic *assumptions* in use of the argument. The blinding property of perfect zero-knowledge does not constitute an assumption and is trivially information-theoretic secure when truly random values are used to hide the computation record. When true randomness is used, the proof itself

becomes a one-time-pad and no adversary has any advantage in recovering the private input. In general, perfect zero-knowledge is added with little overhead and minimal impact on performance.

Non-Interactivity: We achieve non-interactivity in this construction by applying the Fiat-Shamir heuristic. The prover (P) and verifier (V) exchange a single message to instantiate the protocol, and the resulting argument can be verified by any party offline.

Definition 3. *(Non-Interactive Protocols via the Fiat-Shamir Heuristic) The Fiat-Shamir heuristic transforms an interactive proof system (P, V) for a language L under some NP-relation R_L into a non-interactive protocol using a random oracle $\mathcal{H}$.*

- **Proof Generation**: The prover P computes a commitment c based on the input x and private witness w, then queries the random oracle $\mathcal{H}$ with c to simulate the verifier's challenge. The oracle output $\mathcal{H}(c)$ serves as the challenge, and P computes the response r to generate the proof $\pi = (c, r)$.
- **Verification**: The verifier V, upon receiving the proof π, verifies it using the commitment c, challenge $\mathcal{H}(c)$, and response r.
- **Quantum Security Considerations**: In practice, a secure pseudorandom function (PRF) is used in place of the random oracle. To ensure post-quantum security, this PRF must be resistant to quantum adversaries, meaning that its outputs should remain computationally indistinguishable from random to any quantum attacker with oracle access.

5 Mathematical Foundations and Core Concepts

This section delves into the mathematical foundation of our chosen approach. We proceed to define and analyze each mathematical component of our implemented argument system, in order to provide a comprehensive review of the quantum resilience of our design.

5.1 Polynomial Commitment Schemes

A *polynomial commitment scheme* is a cryptographic protocol between a prover P and a verifier V, where the prover commits to a polynomial $P(x) \in \mathbb{F}[x]$ of degree d. The scheme allows the prover to later prove that the polynomial evaluates correctly at specific points, without requiring the verifier to fully evaluate or verify the polynomial at every point in the domain. This preserves computational efficiency for the verifier while still ensuring integrity.

However, while the verifier only checks a subset of evaluation points, the full polynomial (or significant parts of it) often needs to be transmitted from the prover to the verifier, leading to potentially larger proof sizes.

Definition 4. *Let $P(x) \in \mathbb{F}[x]$ be a polynomial of degree d, where $\mathbb{F}$ is a finite field. A polynomial commitment scheme consists of the following algorithms:*

1. **Commit**: The prover P commits to the polynomial $P(x)$ by choosing randomness $r \in \mathbb{F}$, and generating a commitment $c \in \mathbb{F}$, using a commitment function $\text{Com}(P, r)$, such that:

$$c = \text{Com}(P, r)$$

   The commitment c is sent to the verifier V.
2. **Prove**: Given a committed polynomial $P(x)$ and an evaluation point x_0, the prover P generates a proof π that demonstrates the polynomial evaluates to $v = P(x_0)$ at point x_0, using a proof generation function:

$$\pi = \text{Prove}(P, x_0, v, r)$$

   In this step, the prover may still transmit significant parts of the polynomial $P(x)$ or even the entire polynomial to allow the verifier to check the proof at selected evaluation points.
3. **Verify**: The verifier V checks the correctness of the proof π by running the verification algorithm $\text{Ver}(c, x_0, v, \pi)$, ensuring that the polynomial evaluates to the correct value at the selected points. The verifier does not need to evaluate $P(x)$ at every point but gains confidence by verifying the polynomial at a small sample of evaluation points:

$$\text{Ver}(c, x_0, v, \pi) = 1$$

Use in the Argument System: In our argument system, polynomial commitments are crucial for proving the correctness of polynomial evaluations efficiently. The verifier only needs to check the polynomial at specific evaluation points $x_0, x_1, \ldots, x_k \in \mathbb{F}$, which are chosen non-deterministically. The prover generates a proof for each evaluation, producing $\pi_0, \pi_1, \ldots, \pi_k$, which allows the verifier to check the integrity of the computation without revealing the entire polynomial at once.

However, despite the verifier checking the polynomial at a few points, the prover still typically transmits the entire polynomial or its significant parts, which leads to higher proof sizes. This is one of the trade-offs of polynomial commitment schemes: while they offer succinctness in terms of verification, the transmission of large polynomials can lead to increased proof sizes. In further sections, we describe how Merkle trees with auxiliary data are used to compress the polynomial, making transmission and verification more lightweight while achieving the asymptotic goals we laid out in section 4.

Quantum Security Considerations: The security of the polynomial commitment scheme against quantum adversaries relies on the cryptographic hardness of the underlying primitives. In many classical schemes, the commitment function *Com* could be based on assumptions like the discrete logarithm problem, which is vulnerable to quantum attacks via Shor's algorithm. Our design, however, avoids these vulnerabilities by relying on error-correcting codes and hash functions; two primitives that are conjectured to be secure under a quantum attack.

- **Hash Function Security**: In place of pairing-based or number-theoretic assumptions, our design utilizes hash functions to generate commitments. While Grover's algorithm can reduce the bit-security of hash functions by a square root factor, using hash functions with sufficiently large output sizes mitigates this. For example, a 256-bit hash output offers 128-bit quantum security against Grover's algorithm.
- **Quantum-Secure Binding**: The binding property in our commitment scheme depends on the difficulty of finding two distinct polynomials $P(x)$ and $P'(x)$ that, when perturbed by a verifier challenge, generate the same commitment c. This binding is maintained as long as the hash function remains quantum-resistant, ensuring that a quantum adversary cannot efficiently forge a new polynomial $P'(x)$ matching the committed value.
- **No Pairing-Based Assumptions**: By using hash-based commitments, our design avoids reliance on cryptographic assumptions vulnerable to quantum attacks, such as pairings or discrete logarithms. This design choice helps preserve post-quantum security.

Thus, the polynomial commitment scheme maintains security against quantum adversaries, provided that the hash functions used are quantum-resistant and offer sufficient bit-security.

5.2 Reed-Solomon Codes

Reed-Solomon error-correcting codes play a crucial role in constructing a quantum-resilient commitment scheme. They serve two primary purposes:

First, they function as error amplification mechanisms. With high probability, an invalid codeword containing errors will fail to pass verification by a verifying party. This error amplification property ensures the soundness of the argument system, as it becomes computationally infeasible for an adversary to alter a codeword without detection.

Second, Reed-Solomon codes remain robust against quantum adversaries. Unlike certain cryptographic primitives vulnerable to quantum attacks, error-correcting codes like Reed-Solomon preserve their security properties in quantum settings. We expand on these notions with formal definitions below:

Definition 5. *(Reed-Solomon Error-Correcting Codes):*

Let q be a prime power, and let $\mathbb{F}_q$ denote the finite field with q elements. A Reed-Solomon code is a linear error-correcting code defined by evaluating polynomials over $\mathbb{F}_q$. Given parameters n, k, with $k \leq n \leq q$, the Reed-Solomon code encodes a message as follows:

1. **Message as Polynomial**: A message is represented as a vector $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ of k elements in $\mathbb{F}_q$. This vector corresponds to a polynomial $m(x) \in \mathbb{F}_q[x]$ of degree less than k:

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}.$$

2. **Encoding**: The message is encoded by evaluating the polynomial $m(x)$ at n distinct points $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$. The codeword corresponding to the message is:
$$\mathbf{c} = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n)) \in \mathbb{F}_q^n.$$
This vector $\mathbf{c}$ is the Reed-Solomon codeword of length n, and since it is generated by evaluating a polynomial of degree less than k, the code is a linear code with dimension k.
3. **Error-Correction Capability**: A Reed-Solomon code can correct up to $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors. Given a received word $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, the decoding algorithm reconstructs the original message polynomial $m(x)$ as long as the number of errors in $\mathbf{r}$ is less than or equal to $\left\lfloor \frac{n-k}{2} \right\rfloor$.

Thus, the Reed-Solomon code is denoted as $RS[n, k]_{\mathbb{F}_q}$, where n is the length of the codeword, k is the dimension (or number of message symbols), and the code operates over the finite field $\mathbb{F}_q$.

Definition 6. *(List Decoding of Reed-Solomon Codes):*
Let $\mathbb{F}_q$ be a finite field of size q, and let $n \leq q$ be the length of a Reed-Solomon code $RS[n, k]_{\mathbb{F}_q}$. Given a received word $\mathbf{r} = (r_1, r_2, \ldots, r_n) \in \mathbb{F}_q^n$, the task of list decoding is to find all polynomials $m(x) \in \mathbb{F}_q[x]$ of degree less than k, such that the corresponding codeword $\mathbf{c} = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n))$ satisfies:

$$d_H(\mathbf{r}, \mathbf{c}) \leq t'$$

where $d_H(\cdot, \cdot)$ denotes the Hamming distance, and $t' > \left\lfloor \frac{n-k}{2} \right\rfloor$ is the error radius allowed for list decoding.

Key Properties: The Reed-Solomon code $RS[n, k]_{\mathbb{F}_q}$ is a linear error-correcting code defined by the evaluation of polynomials over $\mathbb{F}_q$. In traditional (bounded distance) decoding, the decoder corrects up to $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ errors, producing a unique valid codeword or reporting failure. List decoding extends this by allowing up to $t' > t$ errors, where the number of possible valid codewords may increase significantly, and the decoder returns a list of all such codewords.

Why List Decoding is Hard:

Combinatorial Explosion: As t' increases beyond $\frac{n-k}{2}$, the number of valid polynomials within Hamming distance t' from the received word can grow exponentially. The search space becomes intractably large.

Information-Theoretic Limits: With a large enough error radius t', distinguishing the correct codeword from spurious codewords becomes increasingly difficult. Recovering the original message requires significant computational effort, even when aided by quantum algorithms.

Quantum Hardness: While quantum algorithms such as Shor's algorithm provide efficient solutions for certain problems (e.g., factoring and discrete logarithms), no known quantum algorithm can efficiently solve the list decoding problem for Reed-Solomon codes. The structure of Reed-Solomon codes and the need to explore a combinatorially large space of polynomials makes the problem difficult for quantum adversaries.

Cryptographic Relevance: Reed-Solomon list decoding forms the basis for various cryptographic protocols, such as commitments and zero-knowledge proofs, where proving the correctness of data is essential. In such protocols, an adversary trying to convince a verifier that an erroneous codeword is valid would need to solve this hard list decoding problem.

Given the difficulty of list decoding, even for quantum adversaries, cryptographic systems based on Reed-Solomon codes remain secure in a post-quantum context. Specifically, for cryptographic applications: The adversary cannot efficiently find a valid codeword among potentially many spurious codewords after introducing enough errors. This problem ensures the security of protocols that use Reed-Solomon codes for data integrity and proof systems.

Thus, the list decoding problem is a cornerstone of post-quantum security for Reed-Solomon-based cryptographic protocols, as no efficient classical or quantum algorithm is known to solve it when the number of errors is beyond the unique decoding threshold $\frac{n-k}{2}$.

5.3 Merkle Trees and Hash Functions

Merkle trees are a cryptographic structure that provide a succinct and efficient method for commitment and verification of large data sets. They allow one to verify that specific data belongs to a larger committed dataset with minimal computational overhead, by using a series of hash operations that form a tree-like structure. Each leaf node of the tree represents the hash of a piece of data, while non-leaf nodes represent the hash of the concatenation of their respective children.

By treating a polynomial's evaluations (or the elements of a Reed-Solomon codeword) as the leaves of a Merkle tree, we can generate a tree rooted at a single hash value. The prover can commit to the root of the Merkle tree, and only provide the authentication paths for specific leaves (evaluation points) when queried by the verifier. This allows the verifier to verify the correctness of the evaluation while only needing access to the root of the tree and the corresponding authentication paths.

The key advantage is that, by using a Merkle tree in combination with a technique known as *folding*, the size of the data sent to the verifier is reduced from the full size of the polynomial or codeword $O(n)$ to a size that scales logarithmically in n, i.e., $O(\log n)$. This compression is achieved because the verifier only needs to check the integrity of the polynomial at selected points through the Merkle root and a few nodes in the tree, significantly improving the proof's efficiency. We define folding in further sections.

Quantum Security: The security of a Merkle tree depends on the collision and preimage resistance of the underlying hash function. In this work, the Poseidon hash function is used to construct the Merkle tree. Poseidon is particularly well-suited for zero-knowledge proof systems due to its efficient performance within algebraic circuits. [9][8] gives a reduced round attack on Poseidon, but this is not understood to break security against indistinguishability nor preimage or collision resistance. See appendix for a quantum attack on Merkle trees.

The Merkle tree commitment phase is the definitive bottleneck in the proof pipeline. Its complexity is given by [13]:

$$CC = q_{\text{total}} \cdot \log |F| + AP_{\text{total}} \cdot \lambda \tag{1}$$

Where:

- λ denotes the number of output bits of the cryptographic hash function used to construct a Merkle tree in our system.
- AP_{total} denotes the total number of authentication path nodes in all subtrees of Merkle trees whose leaves are query answers.
- q_{total} denotes the total number of queries made to all proof oracles.

For the hash chain used in the recursive computation (distinct from the Merkle tree), we employ a different hash function. The function used in the hash chain does not impact the security of the commitment scheme (as it only serves as an example computation to be proven) but we take care to select a hash function with adequate bit security to resist quantum attacks. The underlying circuitry framework in use allows for arbitrary, Turing-complete programs to be proven, and so the choice of computation here is arbitrary.

5.4 FRI (Fast Reed-Solomon IOP)

FRI (Fast Reed-Solomon IOP of Proximity) is a protocol aimed at verifying that a committed polynomial has a bounded degree. In FRI, the prover transmits *codewords*, which correspond to Reed-Solomon codewords. These codewords represent evaluations of a low-degree polynomial over a domain D, where the size of D exceeds the number of non-zero coefficients in the polynomial. This size is determined by an *expansion factor* or *blowup factor*, which is the reciprocal of the code rate ρ.

The protocol functions through oracle queries, where the verifier does not access all codewords but instead queries specific points. These codewords are concealed within a Merkle tree in practical implementations. Structurally, FRI can be seen as a polynomial commitment scheme with three components: codewords, an IOP, and the Merkle tree. While these components can be considered separately for scientific rigor, they are often viewed as parts of a unified system for simplicity and accessibility.

In a standard polynomial commitment scheme, the prover commits to a polynomial $f(X)$ and opens it at a specific point z, ensuring that the committed polynomial evaluates to the correct value $f(z)$. The FRI protocol is analogous, but its focus is on verifying that a codeword corresponds to a low-degree polynomial. The prover has access to the entire codeword, while the verifier only requires the Merkle root and a few leaves, which are validated using authentication paths.

One of the key innovations in proof systems is the *split-and-fold* technique, which reduces a proof claim to two smaller claims, each half the size of the original. These claims are then merged into a single claim using random challenges provided by the verifier. After several iterations (logarithmic in the size of the

original claim), the claim is reduced to a trivially small size, whose validity can be verified quickly and cheaply. If this smaller claim is true, the original claim is also true.

For the FRI protocol, this technique is applied to verify that a codeword corresponds to a polynomial of low degree. Let the length of the codeword be N, and let d be the maximum degree of the polynomial. Consider the polynomial as:

$$f(X) = \sum_{i=0}^{d} c_i X^i$$

The *split-and-fold* approach, inspired by the divide-and-conquer strategy of the Fast Fourier Transform (FFT), divides the polynomial into even and odd terms:

$$f(X) = f_E(X^2) + X \cdot f_O(X^2)$$

where

$$f_E(X^2) = \frac{f(X) + f(-X)}{2} = \sum_{i=0}^{\frac{d+1}{2}-1} c_{2i} X^{2i}$$

and

$$f_O(X^2) = \frac{f(X) - f(-X)}{2X} = \sum_{i=0}^{\frac{d+1}{2}-1} c_{2i+1} X^{2i}$$

This decomposition allows the protocol to derive a new codeword for:

$$f^*(X) = f_E(X) + \alpha \cdot f_O(X)$$

where α is a random scalar provided by the verifier.

Let D be a subgroup of even order N of the multiplicative group of the field, and let ω generate this subgroup, i.e., $\langle \omega \rangle = D \subseteq \mathbb{F}_p \setminus \{0\}$. The codeword for $f(X)$ is constructed by evaluating the polynomial on D.

Let $D^* = \langle \omega^2 \rangle$ be another domain, half the length of D. The codewords for $f_E(X)$, $f_O(X)$, and $f^*(X)$ are evaluated on D^*.

Expanding the definition of $f^*(X)$ further, we compute:

$$\left\{ f^*(\omega^{2i}) \right\}_{i=0}^{N/2-1} = \left\{ f_E(\omega^{2i}) + \alpha \cdot f_O(\omega^{2i}) \right\}_{i=0}^{N/2-1}.$$

Expanding again using the definitions of $f_E(X^2)$ and $f_O(X^2)$, we have:

$$\left\{ f^*(\omega^{2i}) \right\}_{i=0}^{N/2-1} = \left\{ \frac{f(\omega^i) + f(-\omega^i)}{2} + \alpha \cdot \frac{f(\omega^i) - f(-\omega^i)}{2\omega^i} \right\}_{i=0}^{N/2-1}.$$

This expands to:

$$\left\{ 2^{-1} \cdot \left(\left(1 + \alpha \cdot \omega^{-i}\right) \cdot f(\omega^i) + \left(1 - \alpha \cdot \omega^{-i}\right) \cdot f(-\omega^i) \right) \right\}_{i=0}^{N/2-1}.$$

Since $\omega^N = 1$, we know that $\omega^{N/2} = -1$, and therefore $f(-\omega^i) = f(\omega^{N/2+i})$. This substitution reveals that, although the index iterates only over half the

range, from 0 to $N/2-1$, all points of $f(\omega^i)$ are used in the derivation of $\left\{f^*(\omega^{2i})\right\}_{i=0}^{N/2-1}$. It doesn't matter that this codeword has half the length; its polynomial has half the degree.

At this point, we can describe the mechanics for a single round of the FRI protocol. The prover commits to the polynomial $f(X)$ by sending the Merkle root of its codeword to the verifier. The verifier responds with a random challenge α. The prover computes $f^*(X)$ and commits to it by sending the Merkle root of $\left\{f^*(\omega^{2i})\right\}_{i=0}^{N/2-1}$ to the verifier.

The verifier now has two commitments, to $f(X)$ and $f^*(X)$, and must verify their relationship. Specifically, the verifier rejects the proof if:

$$f^*(X^2) \neq 2^{-1} \cdot \left(\left(1+\alpha X^{-1}\right) \cdot f(X) + \left(1-\alpha X^{-1}\right) \cdot f(-X)\right).$$

(We ignore the case where $X = 0$.)

To do this, the verifier randomly samples an index $i \in \{0, \ldots, N/2-1\}$, defining three points:

- $A : (\omega^i, f(\omega^i))$
- $B : (\omega^{N/2+i}, f(\omega^{N/2+i}))$
- $C : (\alpha, f^*(\omega^{2i}))$

The next step in the FRI protocol involves the colinearity check. Observe that the x-coordinates of points A and B are the square roots of ω^{2i}. After receiving the index i from the verifier, the prover provides the corresponding y-coordinates, along with their Merkle authentication paths. The verifier uses these to verify membership in the tree and then performs a colinearity check by testing whether points A, B, and C lie on a straight line.

This can be understood by computing the line passing through A and B. Using Lagrange interpolation, we find that:

$$y = \sum_j y_j \prod_{j \neq i} \frac{x - x_j}{x_i - x_j},$$

which can be simplified in the FRI context to:

$$y = 2^{-1} \cdot \left((1 + x \cdot \omega^{-i}) \cdot f(\omega^i) + (1 - x \cdot \omega^{-i}) \cdot f(\omega^{N/2+i})\right).$$

By setting $x = \alpha$, the verifier obtains exactly the y-coordinate of C.

This covers one full round of the FRI protocol. After one round, the prover and verifier return to the same state but with halved codeword lengths and fewer polynomial coefficients. After running $\log_2(d+1)-1$ rounds, where d is the degree of the original polynomial, the prover and verifier reduce the polynomial to a constant. The prover then sends this constant value, completing the process by proving that the codeword corresponds to a polynomial of degree 0.

In the context of quantum security, the security of the FRI protocol relies on its ability to detect fraudulent codewords that do not correspond to low-degree polynomials. This is achieved through a *colinearity check*, which prevents

a dishonest prover from presenting a codeword that passes verification while not corresponding to a valid low-degree polynomial.

A dishonest prover might attempt to present a fraudulent codeword, say $\{f(\omega^i)\}_{i=0}^{N-1}$, which corresponds to a polynomial $f(X)$ of degree greater than d. The colinearity check ensures that the linear combination

$$f^*(X) = f_E(X) + \alpha \cdot f_O(X)$$

remains consistent with the expected polynomial degree constraints, where $f_E(X)$ and $f_O(X)$ represent the even and odd parts of $f(X)$, and α is a random challenge supplied by the verifier.

A complete treatment of the quantum resilience of FRI is currently identified as a gap in the literature. We conjecture that the FRI protocol remains secure against quantum attacks, as the colinearity checks ensure that a fraudulent codeword is unlikely to pass unless it agrees with the valid low-degree polynomial across many points. FRI produces codewords which are represented as Merkle trees composed of leaves.

The problem of producing a fraudulent proof in FRI is reduced to producing valid leaves for invalid codeword evaluations. The prover's challenge then is to find collisions or preimages of the hash used for the Merkle tree. Currently, the best known quantum attacks on hash functions reduce collision resistance by $1/2$ and preimage resistance by $1/3$rd. We posit that increasing the bit security of the hash in use is sufficient to withstand quantum attacks on FRI. A full treatment of this notion, and an evaluation of the relationship between error-correcting codes and fraudulent Merkle leaves, are gaps in the literature and are targeted for future work.

6 Implementation Details

6.1 Frameworks and Tools

We leverage the circuit-building tools from Polygon's Plonky2 project[20]. Plonky2 combines a FRI-based argument with a PLONK-ish arithmetization framework. Encoding a computation into any given argument framework typically requires meticulous effort into ensuring the proper constraining of state transitions. Plonk-based arithmetization frameworks ease this process with a circuit-builder paradigm, abstracting away many of the lower-level details and facilitating a richer expression of computation. Plonky2 additionally supports recursive circuit construction and is well-suited to arguing the integrity of hash chains.

PLONK as an Interface for Circuit Construction

PLONK[19] (**P**ermutations over **L**agrange-bases for **O**ecumenical **N**oninteractive arguments of **K**nowledge) serves primarily as a framework for expressing computational circuits. Its key characteristics are:

- **Circuit Abstraction**: PLONK abstracts computations into polynomial equations and constraints without embedding cryptographic assumptions into the circuit design.
- **Cryptographic Independence**: The construction and constraint of the circuit require no cryptographic assumptions.

Quantum Security with FRI

When paired with FRI:

- **Quantum Resistance**: FRI relies on hash functions and Reed-Solomon codes, which are considered quantum-resistant since quantum algorithms like Grover's algorithm offer only a quadratic speedup.
- **Elimination of Vulnerable Assumptions**: This combination removes reliance on cryptographic assumptions susceptible to quantum attacks (e.g., discrete logarithm problems).

Comparison with KZG Commitments

KZG[25] commitments, while delivering short proofs, have notable drawbacks:

- **Trusted Setup Requirement**: KZG requires a trusted setup, introducing potential vulnerabilities if the setup is compromised.
- **Weak Group Operations**: Relies on cryptographic group operations (e.g., elliptic curve pairings) that are vulnerable to quantum attacks via Shor's algorithm.
- **Cryptographic Assumptions**: Depends on hard problems like the discrete logarithm problem, which are solvable by quantum computers.

Summary

In essence, PLONK acts as an interface for circuit construction without embedding cryptographic assumptions. When integrated with FRI:

$$\text{PLONK} + \text{FRI} \implies \text{Quantum Secure Zero-Knowledge Proofs}$$

Conversely, using KZG commitments:

$$\text{PLONK} + \text{KZG} \implies \text{Short Proofs but Vulnerable to Quantum Attacks}$$

7 Performance Evaluation

7.1 Benchmarking and Metrics

The following charts and tables present our observed performance characteristics for our circuit implementation. Metrics we are interested include prover runtime, verifier runtime, the system RAM in use during proving, and the size of the proof.

Our earlier specification in Section 4 described the prover as linear, meaning runtime should have a linear growth rate with respect the depth of the recursion. The verier should exhibit a logarithmic size growth rate with respect to the size of a single layer of recursion, or single invocation of the hash chain. If the verifier is always logarithmic to a single recursive layer, then it runs in constant-time with respect to all layers.

The system RAM in use is measured in order to quantify the footprint of the program as the depth of recursion increases. STARK systems can consume considerable RAM during the proving process. Measuring the RAM in use at each step helps us understand the cost in memory of each new layer of recursion.

The last metric measured is the proof size in bytes. STARK systems and FRI in general produce large proofs relative to CRS/SRS based arguments. The proof size for our scheme however, should remain constant throughout the entire evaluation of the recursive circuit for any depth. As described in earlier sections, this is due to the all previous proofs being composed into a single proof through recursion.

Table 1: Preliminary Performance Metrics

d (Depth)	$\mathcal{P}$ **Runtime** (s)	$\mathcal{V}$ **Runtime** (ms)	RAM **Used** (MB)	**Proof Size** (bytes)
2	3.3680	3.1013	375.692	133,440
4	4.2126	3.1220	381.536	133,440
8	5.7366	3.0812	392.716	133,440
16	8.8146	3.1098	405.516	133,440
32	14.957	3.0865	417.704	133,440
64	27.294	3.1625	436.424	133,440

7.2 Scalability Tests

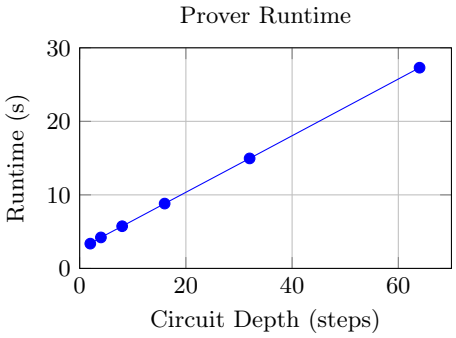

Fig. 1: Prover Runtime vs Circuit Depth

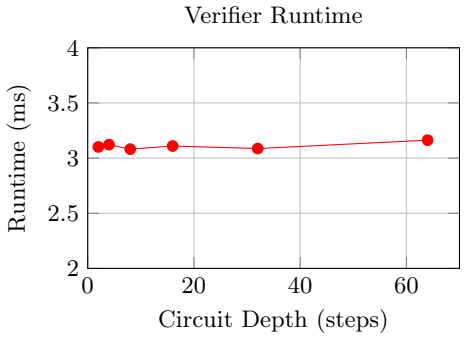

Fig. 2: Verifier Runtime vs Circuit Depth

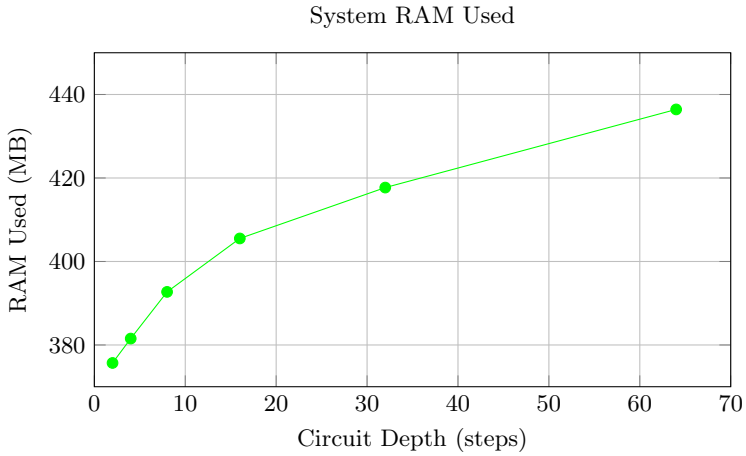

Fig. 3: System RAM Used vs Circuit Depth

7.3 Key Performance Findings

Our preliminary performance analysis reveals several important trends in the observed metrics as circuit depth increases:

Firstly, the prover runtime exhibits a clear linear growth rate. As the circuit depth increases, the time required for the prover increases proportionally. While this indicates an increase in computational overhead on the prover's side, it

suggests that the system remains scalable since the growth remains predictable and manageable.

In contrast, we observe a constant growth rate in verifier runtime. We observe that a single hash invocation can be verified at the same cost as a hash-chain of depth 64. After a rigorous evaluation of our circuit design, this finding confirms that our implementation has successfully produced a recursive argument of integrity for all previous arguments.

Despite a moderate initial size, the final proof size remains constant, showing no growth as circuit depth increases. This property ensures that the amount of data being communicated between the prover and verifier does not increase, maintaining efficiency in terms of communication overhead, regardless of the computation complexity.

Finally, the RAM usage for the prover demonstrates a poly-logarithmic growth rate. This indicates that while memory consumption does increase with circuit depth, it grows at a much slower rate than the computational requirements, making the system resource-efficient in terms of memory usage. This usage growth rate occurs because each recursive layer contains a verification step which is carried out by the prover.

Since the verification step is log of the previous circuit size, we can expect the prover memory footprint to grow at a poly-logarithmic rate. This effect is not immediately visible in the proving runtime because the running time of the circuit is independent of memory usage. Overall, these findings suggest a system that is computationally scalable with manageable resource demands, making it suitable for practical implementations, particularly where verifier efficiency and memory usage are critical.

8 Comparison with Other Works

In recent years, a significant trend has emerged in the development of argument systems: the generalization of these systems over Instruction Set Architectures (ISAs)[27][26][28]. Rather than encoding a single computation into the argument system, this approach encodes the ISA itself as a provable circuit, essentially creating a "provable central processing unit (CPU)"—a concept more commonly referred to as a Zero-Knowledge Virtual Machine (ZKVM). This allows the argument system to prove the correctness of an arbitrary set of instructions executed by the ISA. Instead of proving the integrity of a hard-coded computation, the argument system verifies that the registers within the ISA transition between states correctly, based on the instructions being executed.

The Nexus ZKVM[26] implements a form of incrementally verifiable computation (IVC), a design which closely aligns with our study of recursive argument systems. IVC is the exact mechanism that allows an argument to update continually from the current state, rather than starting fresh, and does not necessarily require the computation to be recursive in nature. The Nexus argument, however, is based on hard problems involving group operations, and is thus unsound in a quantum attack setting. We also find its performance of a single invocation

of a hash to be extremely slow compared to our circuit, and thus an apples-to-apples comparison is not possible. Because it offers no quantum resistance, and because of the wide gap in performance, we omit a programmatic analysis of the Nexus ZKVM from this study.

RISC-Zero[27] is an implementation of the RISC-V ISA in a STARK argument. With a recursive hash chain function, we observe a cost of approximately 7.28 seconds for a single hash invocation on the CPU, and a cost of 120.72 seconds for 64 recursive invocations. Our measurements indicate that our ASIC-style design is about 4.3 times faster than the same function in the ZKVM.

RISC-Zero can leverage a GPU to execute the argument at substantially faster speeds, at the cost of GPU-RAM. We observe a GPU runtime of approximately 1.43 seconds for a recursive depth of 4 at the cost of all available 8 GB of GPU memory. By contrast, our ASIC prover circuit at 4 recursions is about 4 times slower, but uses 21 times less RAM. In both tests, our circuit produced constant sized proofs, while the ZKVM proof was logarithmic in size. Certain configurations of RISC-Zero wrap the resulting CI statement with a Groth16 argument. This produces constant-sized proofs at the cost of quantum resilience, and incurs extra runtime overhead. Our measurements and asymptotic analyses do not consider the the argument with a Groth16 wrapper.

Table 2: Execution time comparison with recursion depth = 4

System	**Proof Time (s)**
RISC-Zero GPU	1.43
ASIC	4.21
RISC-Zero CPU	21.84

Table 3: Memory usage comparison for recursion depth = 4

System	**Memory Used (GB)**
ASIC	0.38
RISC-Zero	8

Table 4: Comparison of Zero-Knowledge Proof Systems. RISC-Zero and SP1 are considered here without a Groth16 wrapper. $\mathbb{G}$ indicates assumptions on group-based problems, while $\mathbb{H}$ indicates a reliance on hash function security. l can be viewed as the verifier runtime in Groth16. It is usually smaller than the circuit size.

System	**Prover**	**Verifier**	**Proof Size**	**Hard Problem**	**ASIC/ZKVM**	**Recursive**
Plonky2	$O(n)$	$O(C)$	$O(C)$	$\mathbb{H}$	ASIC	Yes
RISC Zero	$O(n)$	$O(\log(n))$	$O(\log(n))$	$\mathbb{H}$	ZKVM	Yes
SP1[28]	$O(n)$	$O(\log(n))$	$O(\log(n))$	$\mathbb{H}$	ZKVM	Yes
Nexus	quasi(n)	polylog(n)	polylog(n)	$\mathbb{G}$	ZKVM	No
Groth16	n	$l < n$	$O(C)$	$\mathbb{G}$	ASIC	No

9 Conclusion

Recursive argument systems capture a particularly elegant aspect of computational integrity frameworks. Such statements verify themselves, and can compress a claim about an unbounded computation into a static size. We provide an implementation which encodes a computation as a provable circuit and characterize its algorithmic asymptotics. We additionally address multiple gaps in the literature surrounding quantum-safe argument systems by providing a concrete analysis of various cryptographic assumptions and their hardness in the proximity of a quantum adversary.

This work shows that not only can a quantum-resilient argument encode arbitrary and unbounded computation, but that these systems are among the best-performing frameworks with respect to computational integrity statements, zero-knowledge proofs, and general-purpose argument systems. For future work, we target a formal proof of knowledge soundness when hash function security is compromised due to quantum attacks. In the meantime, in this work we show that commitment schemes based only on hash functions can easily be configured to mitigate any potential known quantum threats. This research suggests overall that a transition to quantum-safe argument systems may lead to performance and usability improvements over systems in wide deployment today.□

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Sci. Statist. Comput. 26, 1484 (1997). https://doi.org/10.48550/arXiv.quant-ph/9508027
2. Quantum-Resistant Cryptographic Algorithms (2022). https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms (Accessed: 2023-10-11)
3. Castryck, W., Decru, T.: An Efficient Key Recovery Attack on SIDH. Cryptology ePrint Archive 2022, 975 (2022). https://eprint.iacr.org/2022/975.pdf
4. Dartois, P., Leroux, A., Robert, D., Wesolowski, B.: SQISignHD: New Dimensions in Cryptography. Cryptology ePrint Archive, Paper 2023/436 (2023). https://eprint.iacr.org/2023/436 (To appear in EUROCRYPT 2024, DOI: https://doi.org/10.1007/978-3-031-58716-0_1)
5. National Institute of Standards and Technology: FIPS 203 (IPD): Initiative for Privacy-Enhancing Data Anonymization (2023). https://csrc.nist.gov/pubs/fips/203/ipd (Accessed: 2023-10-11)
6. Scroll: A zkEVM-based zkRollup for Ethereum. https://scroll.io/ (Accessed: 2023-10-11)
7. Zcash: Zcash - Internet Money. https://z.cash/ (Accessed: 2023-10-11)
8. Zellic, "Algebraic Attacks on ZK Hash Functions," Zellic Blog, https://www.zellic.io/blog/algebraic-attacks-on-zk-hash-functions/\#algebraic-attacks, accessed October 12, 2024.

9. A. Bariant, C. Bouvier, G. Leurent, and L. Perrin, "Algebraic Attacks against Some Arithmetization-Oriented Primitives," *Transactions on Symmetric Cryptology*, vol. 2022, no. 3, pp. 73-101, Sept. 2022. DOI: https://doi.org/10.46586/tosc.v2022.i3.73-101.
10. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT 2016, pp. 305–326 (2016). https://eprint.iacr.org/2016/260
11. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. 17, 297–319 (2004). https://doi.org/10.1007/s00145-004-0314-9
12. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. Cryptology ePrint Archive, Paper 2022/1608, 2022. https://eprint.iacr.org/2022/1608
13. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. https://eprint.iacr.org/2018/046
14. Goldwasser, S., Rothblum, G., Shafer, J., Yehudayoff, A.: ECCC - TR20-058. Available online: https://eccc.weizmann.ac.il/report/2020/058/ [Accessed 20-09-2023]
15. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Paper 2011/506, 2011. https://eprint.iacr.org/2011/506.pdf
16. Szepieniec, A., Lemmens, A., Sauer, J. F., Threadbare, B., Al-Kindi.: The Tip5 hash function for recursive STARKs. Cryptology ePrint Archive, Paper 2023/107, 2023. https://eprint.iacr.org/2023/107
17. Bowe, S., Grigg, J., Hopwood, D.: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019. https://eprint.iacr.org/2019/1021
18. Sean Bowe, "Explaining Halo 2," Electric Coin Company, Sep. 1, 2020. [Online]. Available: https://electriccoin.co/blog/explaining-halo-2/. [Accessed: Oct. 16, 2024].
19. Ariel Gabizon, Zachary J. Williamson, Oana Ciobotaru, "PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge," Feb. 23, 2024.
20. Polygon Labs, "Introducing Plonky2," Polygon Technology, Jan. 10, 2022. [Online]. Available: https://polygon.technology/blog/introducing-plonky2. [Accessed: Oct. 16, 2024].
21. Anonymous.: Non-recursive SHA-256 hashing with Plonky2. arXiv preprint arXiv:2407.03511, 2024. https://arxiv.org/pdf/2407.03511
22. Anonymous.: Plonky2 recursive Poseidon. GitHub repository, 2024. https://github.com/morgana-proofs/plonky2-hashchain/blob/main/src/hashchain.rs
23. Anonymous.: Plonky2 test code. Plonky2 Documentation, 2024. https://docs.rs/plonky2/latest/src/plonky2/recursion/cyclic_recursion.rs.html#136-155
24. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive Oracle Proofs. Cryptology ePrint Archive, Paper 2016/116, 2016. https://eprint.iacr.org/2016/116
25. Aniket Kate, Gregory M. Zaverucha, Ian Goldberg, "Constant-Size Commitments to Polynomials and Their Applications," Max Planck Institute for Software Systems (MPI-SWS), Certicom Research, University of Waterloo.
26. Nexus-xyz. *Nexus ZKVM*, GitHub repository. Available at: https://github.com/nexus-xyz/nexus-zkvm (Accessed: 12 October 2024).
27. Risc Zero, Inc. *RISC Zero ZKVM*, GitHub repository. Available at: https://github.com/risc0/risc0 (Accessed: 12 October 2024).

28. Succinct Labs. *SP1: Zero-Knowledge Succinct Proof System*, GitHub repository. Available at: `https://github.com/succinctlabs/sp1` (Accessed: 12 October 2024).

A Quantum Adversary Simulations

We have defined in previous sections the requisite components of our argument system, and we show that the main primitive in use in this scheme is a secure hash function. In this section we provide a rough proof sketch of a quantum adversary against a hash function. The purpose of this section is to provide a concrete and formal analysis of the quantum security of this scheme, which is identified as a gap in current research with respect to argument systems.

Let $H : \{0,1\}^* \to \{0,1\}^n$ be a cryptographic hash function with n-bit output.

Cryptographic Properties

- **Pre-image Resistance**: Given a hash value $y \in \{0,1\}^n$, it is computationally infeasible to find any $x \in \{0,1\}^*$ such that $H(x) = y$.
- **Collision Resistance**: It is computationally infeasible to find any two distinct inputs $x, x' \in \{0,1\}^*$ such that $H(x) = H(x')$.

Quantum Adversary Model Let $\mathcal{A}_Q$ be a quantum adversary with the following capabilities:

- Executes Grover's algorithm to find pre-images and collisions in $O(2^{n/2})$ time.
- Has access to the quantum equivalent of an oracle for H.

Simulation of the Attack & Breaking Pre-image Resistance

Adversary's Goal Given a hash value $y \in \{0,1\}^n$, find an input $x \in \{0,1\}^*$ such that $H(x) = y$.

Attack Procedure

1. **Set Up the Quantum Oracle**
   The adversary prepares a quantum oracle U_H that, for any input state $|x\rangle$, computes:
   $$U_H|x\rangle|0\rangle = |x\rangle|H(x)\rangle$$
2. **Apply Grover's Algorithm**
   The adversary uses Grover's algorithm to search for x such that $H(x) = y$. The algorithm operates in $O(2^{n/2})$ time.
3. **Outcome**
   With high probability, the adversary finds an x such that $H(x) = y$.

Implications

- The adversary can invert the hash function, violating pre-image resistance.
- In the context of the argument system, this allows the adversary to forge proofs or commitments that rely on the hash function.

A.1 Breaking Collision Resistance

Adversary's Goal Find two distinct inputs $x, x' \in \{0,1\}^*$ such that $H(x) = H(x')$.

Attack Procedure

1. **Prepare Superposition**
   The adversary prepares a uniform superposition over all possible inputs:
   $$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle$$
   where $N = 2^n$.
2. **Construct the Oracle**
   The oracle marks states where $H(x) = H(x')$ for $x \neq x'$.
3. **Apply Quantum Collision-Finding Algorithm**
   Use the BHT (Brassard-Høyer-Tapp) quantum algorithm to find collisions in $O(2^{n/3})$ time.
4. **Outcome**
   The adversary finds x and x' such that $H(x) = H(x')$.

Implications

- Collision resistance is compromised.
- The adversary can produce multiple inputs mapping to the same hash value, undermining the integrity of commitments in the argument system.

A.2 Impact on the Argument System

A.3 Compromising Soundness

The soundness property relies on the inability of an adversary to produce valid proofs for false statements. By breaking the hash function, the adversary can:

- Forge commitments that appear valid to the verifier.
- Generate fraudulent proofs that pass verification.

A.4 Attack on Commitment Schemes

Commitment Scheme Overview Assume the argument system uses a hash-based commitment scheme:

$$\text{Commit}(m, r) = H(m \parallel r)$$

where m is the message and r is a random nonce.

Adversary's Attack

1. **Breaking Hiding Property**
   By finding a pre-image of a commitment, the adversary can discover m without the nonce r.
2. **Breaking Binding Property**
   By finding collisions, the adversary can produce different (m', r') such that:

   $$H(m \parallel r) = H(m' \parallel r')$$

   This allows the adversary to change the committed message after the commitment is made.

A.5 Forging Proofs

In zero-knowledge proofs that utilize hash functions for challenge generation, breaking the hash function allows the adversary to predict or manipulate challenges.

Adversary's Strategy

1. **Challenge Prediction**
   By inverting the hash function, the adversary can predict the verifier's challenge ahead of time.
2. **Proof Manipulation**
   The adversary can craft responses that satisfy the verification equation without possessing a valid witness.

A.6 Adjusting Bit Security to Mitigate BHT Quantum Attacks

To mitigate the Brassard-Høyer-Tapp (BHT) quantum algorithm, which finds collisions with a complexity of $O(n^{1/3})$, the bit-length of the hash function needs to be adjusted to ensure that the hash function remains secure against quantum adversaries.

A.7 Understanding the BHT Attack Complexity

BHT reduces the complexity of finding a collision from the classical $O(n^{1/2})$ (as per the birthday paradox) to $O(n^{1/3})$ using quantum techniques. For a hash function with an output size of b bits:

- A classical **birthday attack** would require $O(2^{b/2})$ operations to find a collision.
- The BHT algorithm, however, requires only $O(2^{b/3})$ operations to find a collision, offering a quantum speedup over classical methods.

A.8 Desired Security Level

To achieve a desired security level of k bits against quantum adversaries, we need to ensure that the quantum attack complexity is at least 2^k operations.

For a quantum adversary using the BHT algorithm, this means:

$$2^{b/3} \geq 2^k$$

which simplifies to:

$$b \geq 3k.$$

A.9 Example: Achieving 128-bit Quantum Security

If we desire 128-bit security against quantum adversaries (a common target for post-quantum cryptography), we must choose the bit-length b of the hash function such that:

$$b \geq 3 \times 128 = 384 \text{ bits.}$$

Thus, a hash function with at least 384-bit output is required to achieve 128-bit quantum security.

A.10 Summary of Bit-Length Adjustments

- For classical birthday attacks, which have complexity $O(2^{b/2})$, a 256-bit hash function (e.g., SHA-256) is sufficient for 128-bit **classical security**.
- To mitigate the BHT quantum attack (which has complexity $O(2^{b/3})$), we need a hash function with at least 384 bits of output to achieve 128-bit **quantum security**.

A.11 Recommended Hash Functions

For 128-bit quantum security:

- If a standard hash function such as **SHA-256** (with a 256-bit output) is in use, it provides:
  - 128-bit classical security (since $2^{256/2} = 2^{128}$).
  - 85-bit quantum security against BHT (since $2^{256/3} \approx 2^{85}$).

- To achieve 128-bit quantum security, it is recommended to use **SHA-384** (which outputs 384 bits) or a truncated version of **SHA-512** to 384 bits, providing:
  - 192-bit classical security (since $2^{384/2} = 2^{192}$).
  - 128-bit quantum security against BHT (since $2^{384/3} = 2^{128}$).

This simulation demonstrates that a quantum adversary utilizing Grover's algorithm can effectively break the pre-image and collision resistance properties of cryptographic hash functions used in the argument system. This compromises the soundness of the system, allowing the adversary to forge commitments and proofs. These attacks are effectively mitigated by selecting a hash function with a sufficiently high level of bit security.