National College of Ireland

Privacy-Preserving Public Auditing of Outsourced Cloud Data with Zero Knowledge Proofs

MSc Research Project

Msc Cloud Computing

Ikenna Ughanze

Student ID: 23384069

School of Computing

National College of Ireland

Supervisor: Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	.....Ikenna Ughanze 23384069
Student ID:	Msc Cloud Computing 2025
Programme:	..... **Year:** .....
Module:	Research Project
Supervisor:	Punit Gupta
Submission Due Date:	11-08-2025
Project Title:	Privacy-Preserving Public Auditing of Outsourced Cloud Data with Zero Knowledge Proofs
Word Count:	.....19..... **Page Count**.....8134.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
<u>ALL</u> internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ikenna Ughanze
Date:	13-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Privacy-Preserving Public Auditing of Outsourced Cloud Data with Zero Knowledge Proofs

Ikenna Ughanze
23384069

Abstract

The growth in the technology of cloud storage has meant that more and more organizations outsource large amounts of data to the cloud for storage, for archival and compliance purposes. This offers cost and scalability benefits but it introduces issues around verifying data integrity and the risk of leaking sensitive during audits by third parties. Past public auditing frameworks rely on interactive challenge-response mechanisms that leak metadata, increase audit latency and are susceptible to man in the middle attacks. This research proposes a non-interactive, privacy preserving public auding framework using Zero-knowledge Scalable Transparent Arguments of Knowledge (zk-STARKS). Using the Winterfell library to generate cryptographic proofs that a data set remains unchanged without revealing the content of the data or structural metadata to the verifier. The frame is deployed using a cloud native architecture, composed of AWS services. Evalutation of a 200MB dataset split into 2mb per block achieved proof and verification times of approx. 900 ms per block, which can be scaled up to larger datasets with limited cloud compute costs. The result confirm that zk-STARKS can be used to implement an efficient, trust less and privacy preserving framework for cold data auditing, detecting tampering of data without the need for a trusted TPA.

1 Introduction

The advancement in cloud computing technology has transformed the data storage domain providing cheap storage for large amounts of data, allowing individuals and companies to outsource large amounts of data storage to third part cloud service providers (CSPs). This provides advantages in terms of scalability and cost savings. However, it also creates challenges such as data integrity, privacy, and auditability. Data owners give up control over their data; this creates the need for verifiable mechanisms to ensure the outsourced data remains intact and tamper free. Public auditing appeared as a solution for verifying data integrity without downloading the entire dataset. However, traditional auditing frameworks suffer from some limitations, trust assumptions in third-party auditors (TPAs), interactive challenge-response mechanisms, and are susceptible to metadata leakage. Cold data (infrequently accessed data) stored for archival purposes in domains like healthcare, finance, law etc presents unique challenges. Its infrequent access makes real-time interactions costly and inefficient. (Cold Data Storage Explained | Seagate UK) reports a significant percentage of cloud storage data falls under this category, however existing auditing frameworks are inadequate in verifying data integrity for this category.

The research problem revolves around developing an efficient, trustless and privacy preserving auditing framework for cold storage data in cloud computing. Public auditing enables data owners to verify data integrity with the help of Third-party Auditors (TPAs), who ensure that cloud service providers (CSP) store data correctly without altering or deletion. However, some works (Gan et al 2024, Wang et al, 2021) rely on interactive mechanism and trusted setup, creating problems of increased latency, metadata exposure and the risk of malicious collusion. These frameworks introduce communication overhead also which makes them inefficient for cold data, which require infrequent but efficient audits. Data breaches or integrity violations in cold storage data can lead to regulatory penalties due to GDPR. Some existing alternatives like BPAO (2021) and Zhang et al 2021, address some of these limitations by eliminating TPA dependency but still face issues related to blockchain complexity.
These limitations underline the urgent need for a trustless, privacy-preserving, and non-interactive auditing mechanism suitable for the archival nature of cold data. This research proposes a novel public auditing framework leveraging zero-knowledge succinct transparent arguments of knowledge (zk-STARK s) to ensure data integrity, minimize trust, and enhance privacy for cold data in cloud storage.

Zero-Knowledge Proofs (ZKPs) is a cryptographic breakthrough, offering a compelling solution to the problem between data verification and privacy. ZKP is a protocol that allows one party (the prover) to convince another party (the verifier) the validity of statement without exposing any information beyond the fact that the statement is valid. This is achieved by three fundamental principles: Completeness (an honest prover can always convince a correct verifier if the statement is true), Soundness (a dishonest prover cannot convince the verifier of a false statement) and Zero-knowledge (the verifier learns nothing beyond the truth of the statement) (Goldwasser, Micali and Rackoff, 1985). ZKPS offers a novel cryptographic solution to the problem of data integrity auditing by using mathematical certainty of verification without compromising data privacy. This allows organizations to meet regulatory compliance requirements while protecting sensitive information. The practical applications of ZKPS extends to decentralized identity verification, privacy in financial transactions and audits, proof of reserves, secure IoT device authentication and supply chain verification (Top 10 Zero Knowledge-Proof Applications to Know). Zero-Knowledge Scalable Transparent Arguments of Knowledge (zk-STARK) is a cryptographic protocol that enables efficient, transparent and privacy-preserving proofs (*Zero Knowledge Proofs: Enhancing Blockchain Scalability*, 2024). Their transparency and scalability make them ideal for decentralized applications where auditability and trust minimization is required. Despite their potential, zk-STARKS remains unexplored in the domain of cloud storage auditing, particularly in the context of cold data.

The research aims to address some of the highlighted issues in previous works by developing a zk-STARK based public auditing framework for outsourced cloud storage, with a focus on cold data.

The research questions is:

Can zk-STARKs be used to create an efficient, privacy-preserving, non-interactive and zero trust auditing framework for outsourced cold cloud data ?

The proposed zk-STARK based framework provides several potential contributions to the field of cloud security and data auditing: first, the proposed framework aims to eliminate back and forth interactions between CSP and auditor found in current systems, second is enhanced privacy, zk-STARK are zero-knowledge ensuring that no data or metadata is leaked during verification. Zk-STARK eliminates trust or semi-trust setup eliminating risks that can arise from malicious CSP or TPA. The benefit of this solution includes improved efficiency for infrequent audits and robust privacy for sensitive archival data. The use of Zk-STARK shifts the focus of the auditing framework from a trust-based, interactive, re-execution-based approach to a computationally verifiable proof model, improving the trust and security of the system and enabling more efficient forms of data governance. instead of depending on the third party to re-execute a computation or audit data, data integrity is cryptographically proven, allowing a “generate proof once, verify anywhere model” (zkVerify: Optimizing ZK Proof Verification At Scale - Delphi Digital).

This study addresses gaps in existing research by creating a system that eliminates trust dependencies and enhance audit privacy and efficiency, contributing to the advancement of secure cloud auditing and offers a practical solution that can be extended to proving that sensitive data is compliant with GDPR and data protection audits.

The rest of the paper is organized as follows: section 2 is reviewing related works, section 3 discusses the research methodology, section 4 highlights the Design architecture of the proposed solution, section 5 discusses the implementation and 6 & 7 concludes the paper with evaluation, discussion, conclusions and future works.

2 Related Work

This literature review examines previous works on privacy preserving public audit of outsourced cloud storage data. The review is organized thematically around key areas: early works in PDP and PoR frameworks, public auditing frameworks, challenge-response mechanisms in trust-based auditing, privacy-preserving techniques and decentralised trust-based systems using blockchain based auditing. Each theme critically compares prior works, highlighting their objectives, contributions, and relevance to the research questions, while identifying their advantages and shortcomings. Scalability issues are noted as limitations where applicable, but the focus remains on interactivity, trust, and privacy, aligning with the research question of developing a trust less, privacy-preserving auditing solution.

2.1 Early PDP and PoR Frameworks

Early works on Provable Data Possession (PDP) and Proofs of Retrievability (PoR) laid the groundwork for public auditing. These frameworks introduced a mechanism to verify data integrity and retrievability in cloud storage. However, these frameworks rely on interactive protocols and trust-based systems, impacting their suitability for cold data auditing.

(Ateniese *et al.,* 2007) introduced PDP, using a homomorphic-tag scheme that lets a TPA verify data integrity by checking random blocks instead of retrieving the entire full files. It has low client storage and verification costs which makes it efficient, but the challenge-response rounds incur latency and bandwidth overhead coupled with the reliance on a trusted TPA which introduces trust risks. (Juels and Kaliski, 2007) proposed PoR, building on the work on PDP, adding error-correcting codes and sentinels to prove full data recoverability. While the guarantee of data recovery is crucial for archival storage, PoR has similar limitations to PDP as it uses interactive Audits and relies on TPA trust on top of the additional costs of encoding and decoding sentinels. PDP focuses on efficiency while PoR focuses on retrievability, but both suffer from interactivity and trust issues which the proposed zk-STARK framework overcomes via non-interactive, publicly verifiable proofs, eliminating both round-trip costs and TPA trust.

2.2 Early Approach to Public auditing

(Wang *et al.,* 2010) in one of the early works in public auditing of cloud storage data introduced a privacy-preserving public auditing framework using homomorphic linear authenticators (HLAs) in combination with random masking to allow TPA verify data integrity without downloading the entire dataset. While this reduces client-server bandwidth overhead, the HLA tags used in this framework is vulnerable to existential forgery attacks. To improve security (Wang *et al.,* 2013) replaced HLAs with Boneh-Lynn-Shacham signatures for authentication, however this created computationally intense pairing operations that increased prover and verifier costs making it not usable for large could data storage or frequent audits. Both approaches achieved public verification, The 2013 work by Wang et al improved security at the cost of efficiency, however there is an issue with balancing computational cost and guaranteeing security.

In a goal to improve performance (Wang *et al.,* 2011) Introduced a batch auditing framework that uses Merkle Hash Trees (MHT), that allowed a TPA verify multiple data owners proofs at a go ,reducing verification time significantly. However, embedding tree hashes in the clear exposed metadata hampering data confidentiality.

(Solomon et al., 2014) created another auditing framework that used aggregated and ordered BLS signatures to enable verification for multiple data blocks. This technique achieved good efficiency results compared to wang et al but still had computation overhead issues that impacted real world usage. Both These works focused on verification efficiency, but strong data confidentiality is not guaranteed.

2.3 Challenge-Response Mechanisms in Trust-Based Auditing

Auditing cloud storage data often involved challenge-response mechanisms that required a trusted TPA to send challenges to the cloud service provider, which responded with data integrity proofs. These frameworks are inefficient for cold data audits due to communication overhead and trust assumptions.

(Gan *et al.,* 2024) proposed an online/offline auditing framework that is protected from key exposure by precomputing signatures offline, thereby reducing computation overhead during the online phase. This approach is effective in resource constrained devices, and the offline component aligns well with the infrequent access characteristic of cold data storage. However, the online phase relies on challenge-response interactions, which introduces communication overhead, and its dependence on a TPA introduces trust concerns. (Yang *et al.,* 2021) developed a lightweight delegatable proofs of storage framework that uses homomorphic encryption to minimize computational overhead. Its strength is in its low resource demand, which is suitable for static data, but the challenge-response mechanism and TPA reliance increase latency and trust-related vulnerabilities. Both frameworks have their advantages but are limited by their interactive nature, a limitation overcome by the proposed zk-STARK based framework, which replaces back and forth interactions with non-interactive proofs, thereby reducing trust dependencies and improving auditing efficiency for cold data. (Tian, Wang and Wang, 2021) focused on optimizing audits for frequently accessed data through user behavior prediction. Although effective for active data, this model is not well-suited to cold data environments and still requires interactive challenge-response protocols.

These frameworks highlight the challenge of interactivity and trust in TPAs, limiting their effectiveness for cold data audits. The proposed zk-STARK solution eliminates these constraints by supporting trustless, non-interactive auditing tailored to the needs of archival or cold data storage.

2.4 Public Auditing Against Malicious Auditor

Public Auditing frameworks for cloud storage rely on TPAs to verify the data integrity of outsourced data, reducing the computational burden on data owners. However, some TPAs can act maliciously by forging audit results, colluding with CSPs or falsifying results to harm CSP reputations.

To tackle malicious TPAs (Armknecht *et al.,* 2014) proposed fortress a framework where the TPAs perform two PoR audits, one for them and the other for the data owner. The results are logged for transparency, allowing the users to detect forged outcomes. While this approach enhances transparency, it required the data owner to download and verify entire logs, incurring communication overhead costs. (Zhang *et al.,* 2017) introduced a public auditing framework that uses indistinguishable obfuscation to restrict malicious TPAs ability to forge results. By wrapping the audit computations in an obfuscated program, their method ensured the TPA cannot alter the results without detection, preserving data integrity. However, generating these programs generates computational burden on data owner, making the framework inefficient for large-scale cold data storage.

(Ben Haj Frej, Dichter and Gupta, 2018) developed a lightweight auditing framework that mitigates malicious TPA by using a trusted entity to distribute secret random shared keys among system participants. This reduces the TPA's ability to forge results without access to these keys and lowers communication overhead compared to Fortress. However, another trusted party is introduced to issue a group of secret random shared keys to the roles in the system, this introduces a new point of trust, creating a new point of risk and failure.

The reviewed frameworks highlight a persistent challenge of malicious TPAs in public auditing, with other limitations of communication and computational overhead. The proposed zk-STARK based framework eliminates the malicious TPA problem entirely by removing the need for TPAs, using non-interactive zero-knowledge proofs that can be verified by the data owner themselves or on a blockchain.

2.5 Decentralized Trust based Auditing

Decentralized auditing frameworks use blockchain and smart contracts to remove or reduce TPA dependency, offering a trustless, transparent alternative way for cloud data auditing.

(Lin *et al.,* 2021) (BPAO) Proposed a blockchain-based auditing framework that outsources verification to cloud servers using smart contracts, eliminating TPAs. The strength of this framework is its transparent and trustless design, suitable for data audits due to automated and periodic checks. However, blockchain overhead costs and smart contract complexity can limit efficiency. PPTPS used blockchain to create tamper-proof verification and timestamping and ensured privacy with random masking. Its traceability feature is valuable for auditing, but it relies on valid blockchain transactions and TPA interactions increase complexity. Both frameworks implement a trustless system with BPAO more so, but both introduce blockchain-related overhead.

(Wang *et al.,* 2024) enhanced identity-based auditing with blockchain logging to resist malicious auditors, improving security over Xue et al. (2019), which was vulnerable to tag forgery. Wang et al.'s strength is its tight security reduction, but its challenge-response mechanism and metadata exposure limit its suitability for cold data audit. Decentralized audit frameworks eliminate trust, but blockchain technology introduces complexity. The proposed zk-STARK framework employs a trustless protocol while avoiding the added blockchain overhead, enhancing efficiency for cold storage data auditing.

The literature review covers several works in public auditing of outsource cloud data from early PDP and PoR frameworks to decentralized trustless auditing frameworks. However, these implementations have some limitations due to: interactive nature of challenge-response mechanisms introducing communication overhead, trust dependencies in TPA-based audit frameworks creating vulnerabilities of collusion or misuse, metadata leakage in interactive protocols compromising privacy and some other frameworks are limited by scalability issues. These gaps are addressed by the proposed zk-STARK framework which provides a non-interactive, zero-knowledge proof system with succinct proofs that have small size and quick verification times, eliminating TPA trust and metadata leakage, making it ideal for cold data storage auditing. This research is justified in developing a trustless, efficient, and privacy preserving solution for cold storage data auding.

Aspect	My Research	Gan et al (2024)	Zhang et al (2022)	BPAO (2021)	Yang et al(2021)
Methodology	zk-STARK s with Merkle hash trees; non-interactive, transparent proofs	Online/offline auditing; cryptographic signatures	Lattice-based IBPA; public verification	Blockchain with smart contracts; outsourced auditing	Homomorphic encryption; delegatable proofs
Privacy	No data/metadata leakage (zk-STARK s ensure zero-knowledge)	High (TPA cannot access data)	High (no data access)	High (blockchain ensures privacy)	High (homomorphic encryption)
Efficiency	High (succinct proofs, milliseconds verification)	High (offline pre-computations reduce online cost)	Moderate (lattice cryptography is complex)	Moderate (blockchain overhead)	High (lightweight proofs)
interactivity	Non-interactive (eliminates challenge-response)	Low (online phase minimizes interaction)	Low (minimal TPA interaction)	Low (automated smart contracts)	Low (delegatable auditing)
Security	Strong (zk-STARK s, transparent, post-quantum secure)	Strong (key-exposure resilience)	Strong (post-quantum secure)	Strong (blockchain tamper-resistance)	Moderate (bilinear pairing assumptions, not post-quantum)
Scalability	Proof size are succinct, verification time in milliseconds	Moderate (large dataset concerns due to offline computation)	Moderate (lattice complexity limits large datasets)	Limited to choice of Blockchain	Moderate (homomorphic encryption limits large datasets)
Limitations	ZKP computational complexity (Large data)	TPA dependency; offline computational overhead	Computational complexity; TPA dependency	Blockchain overhead; smart contract risks	TPA dependency; no post-quantum security

Table 1: Comparision Table of Key Works.

3 Research Methodology

The aim of this research is to design, implement and evaluate a zk-STARK -based, non-interactive privacy-preserving public auditing framework that ensures the integrity of cold data in outsourced cloud storage. The methodology will detail the research procedure used, tools, techniques and evaluation of the proposed framework, addressing the research question on non-interactivity, privacy preservation and trust elimination. To achieve this, the proposed solution will:

1. Develop an auditing framework, where the cloud server generates proof that the data remains unchanged without revealing any information about the data.
2. Use Winterfell zk-STARK library to generate efficient proofs for data integrity verification.
3. Eliminate challenge-response overhead using Non-interactive Zero-Knowledge proofs.

The rest of the research methodology is structured as follows: key components needed for creating this framework, high level overview of the zk-STARK based audit framework and evaluation methodology.

3.1 Merkle Tree and Authentication Paths

Merkle tress (hash trees) are authenticated data structures designed to efficiently and securely verify the integrity of large datasets. In a Merkle tree, each leaf node contains the cryptographic hash of a data block, while every non-leaf node contains concatenated hashes of its child nodes (Merkle Tree — iden3 0.1 documentation). This hierarchy of hashing creates a single unique root hash at the top, known as Merkle root, which succinctly represents the integrity of the whole dataset (Merkle Tree — iden3 0.1 documentation). The key advantage of Merkle trees is the ability to perform lightweight verification, instead of requiring a verifier to download and re-hash an entire dataset to confirm the integrity of a single data block, only a small subset of hashes known as the Merkle tree authentication path or Merkle proof is needed. If the root hash is recomputed from this authentication path and it matches the known Merkle tree root, it cryptographically proves the inclusion and integrity of the data (it proves the data block is from the original entire dataset). The Zero knowledge proof systems role is to prove the correctness of the Merkle path computation in zero-knowledge, convincing the verifier that the data block is part of the Merkle tree (the data set) without learning any information.

3.2 Zero-Knowledge Proof Systems

Zero-knowledge Proof (ZKPs) is a cryptographic technology that allows a prover to show knowledge of a secret value to a verifier without revealing any information about the secret itself (Ben-Sasson et al.).
This ability is covered by three core principles:
Completeness: if a statement is true, an honest prover can always convince a correct verifier of its truth.
Soundness: If a statement is false, no malicious prover can convince the verifier that it is true.
Zero-knowledge: The verifier learns nothing besides the fact that the statement is true (Zero-Knowledge Proofs in Blockchain: Ultimate Scalability Guide).
ZKPs can be categorized into interactive and non-interactive proofs. interactive proofs require a series of challenges and responses between the prover and verifier until the verifier is convinced. While effective, this requires real-time communication. however, non-interactive proofs allow the prover to generate a single message (the proof) that the verifier can check independently, making them suitable for asynchronous environments like blockchain where reduced communication is crucial for efficiency (Zero-knowledge proof - Wikipedia) This research focuses on the use of zk-STARK S (Zero-Knowledge Scalable Transparent Arguments of Knowledge) for the creation of the novel public audit framework. "A STARK is a novel proof-of-computation scheme to create efficiently verifiable proofs of the correct execution of a computation" (Ben-Sasson *et al.,* 2018). Starks offers the advantage of transparency, no trusted setup and post quantum resistance which are valuable for long-term trust and robustness in enterprise and data integrity.

3.3 Statistical sampling

For large scale data integrity auditing, verification of the entire uploaded data can be computationally and economically costly. Statistical sampling offers a solution to achieve high audit confidence with reduced cost and computational overhead. Binomial probability model provides a suitable mathematical foundation for this, it allows for calculating the likelihood of detecting a certain number of data corruptions within a fixed number of sampled trials, given an assumed corruption rate (Binomial Distribution: Uses & Calculator - Statistics By Jim). A crucial part of statistical sampling is determining the appropriate sample size and calculating the confidence interval. To ensure that audit results are statistically representative of the entire dataset, formulas exist to compute the minimum sample size required to achieve a desired level of accuracy and confidence (e.g 95%) considering an acceptable margin of error. While auditing the entire data offers 100% certainty, the cost can be extremely high for massive datasets, sampling methods can be more cost efficient. The inclusion of a configurable corruption rate input field enables the user can input their estimated data corruption rate, the system can dynamically adjust the sample size required to meet the 95% threshold, therefore optimizing computational costs and audit duration based on the actual risk profile of the data.
Sampled audit system are dependent on the cryptographic security and unpredictability of the random number generation algorithm. A Cryptographically Secure Pseudorandom Number Generator (CSPRNG) is critical to ensure that the selection of data blocks during auditing is unpredictable and resistant to manipulation (Rukhin *et al.,* 2010). Any vulnerability in the CSPRNG could allow an attacker to predict the data blocks which are to be audited and selectively tamper with them, rendering the audits invalid and the entire system unreliable. The quality

of randomness is a security-critical component, Monte Carlo simulation a technique that estimates and analyzes complex systems using random sampling, can be used to validate the confidence intervals and overall effectiveness of the sampling strategy (Understanding the Monte Carlo Simulation | Baeldung on Computer Science).

Total Blocks (N)	Assumed Corruption (%)	Blocks Audited (c)	Detection Probability (%)
1000	1% (10 blocks)	298	95%
1000	5% (50 blocks)	59	95%
1000	10% (100 blocks)	29	95%

Table 2: Typical scenario in practice (verifying 3–10% of blocks) per audit round

3.4 Cryptographic Workloads on the Cloud

Cloud native serverless architecture provides an ideal execution environment for computationally intensive ZKP proof generation due to the on-demand nature, and pay-per-use model, it reduces the potential cost challenge by converting it into variable usage-based expenses. Serverless applications such as AWS Lambda, offer a solution for deploying cryptographic workloads. The frameworks cryptographic operations, mainly STARK proof generation can be efficiently implemented in RUST (Facebook's Winterfell library) and deployed as containerized functions on AWS EC2. This approach enables fine-grained control over the execution environment while offloading infrastructure management to AWS.
Orchestrating a complex, multi-stage cryptographic audit workflow over different decoupled serverless components (eg, fetching data blocks, proof generation, verification) requires a workflow management tool. AWS Step functions provide serverless orchestration to manage workflows, maintaining application state, built in error handling, timeouts, and parallel processing for distributed applications.

3.5 Proposed Audit Framework

The audit framework will follow these steps:

1. **Data owner uploads data:** Data is uploaded and stored on the cloud and a cryptographic commitment is generated. This commitment is mathematical proof that the data exists in a verifiable form without revealing any information about the data. The proof is computed using a hash-Merkle tree, this ensures that future verification can be conducted without access to the raw data.
2. **Cloud server Generates Proof:** The CSP receives a challenge of random blocks to verify integrity of. The CSP generates a zk-STARK proof integrity proof. This proof confirms that the stored data has not been altered since the initial commitment was made.
3. **Verifier:** The Verifier checks the proof without accessing or revealing any underlying data (metadata or encryption details), ensuring that the data verification process is privacy-preserving. This framework ensures that data integrity can be verified while maintaining confidentiality, reducing the risk of sensitive data exposure and eliminates the need for challenge-response mechanism which slows down verification process and increases audit complexity.

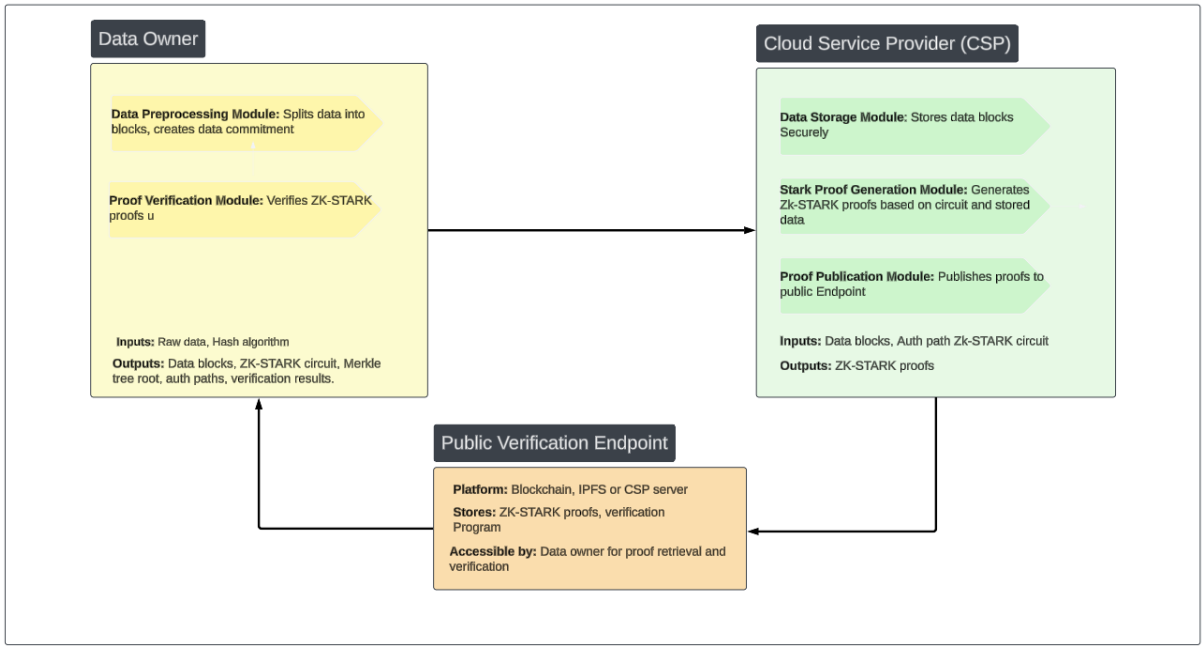


Figure 1: Audit Framework Flow

Technologies and frameworks	Description
Sha3-256	Data commitment hashing algorithm
Merkle Tree	Hash representation of entire dataset
Python	Genrate fake test data Generate fake data and hash data blocks
Winterfell Library (Rust)	STARK proofs generator and verification library
Web technologies (HTML, CSS, JS frameworks)	Develop a Front-end view to access audit results and verify proofs
AWS Cloud Platform (s3,Lambda, Dynamo DB, Step functions)	Simulate cloud storage environment and test auditing framework workflow

Table 2: Tools, Technologies and Techniques used

3.6 Evaluation Methodology

To evaluate the proposed zk-STARK based privacy-preserving audit framework, the following was considered: Functionality and performance. The goal is to demonstrate that the solution proves data integrity of stored data without exposing any information (privacy analysis), and that it eliminates trust issues (TPA/CSP) while generating competitive proof sizes and fast verification speed. The solution will also be evaluated on if it can detect if data has been tampered with or not.
The following metrics are taken:

Metric	Description
Information leakage	qualitatively assess what an external verifier learns from each proof.
Proof size	Record Proof size.
Proof and Verification time	Record the Stark proof generation and verification time

To assess the information leakage of this solution, The zk-STARK proof process is evaluated against a traditional methodology of proofing Merkle tree path verification to assess the information leakage of the solution. To assess the solutions ability to detect tampering, data blocks will be selectively altered and parsed into the zk-STARK based audit system to generate proofs and verify the proofs. If the proof verification fails, it proves that the system is able to detect tampering. Finally, to evaluate the performance and scalability of the proposed zk-STARK based auditing framework, proof generation and verification is recorded for a fixed dataset size and results are extrapolated for larger datasets. This approach assumes that the proof generation and verification processes scale

proportionally with the number of blocks. This scaling analysis enables an informed projection of how the system will perform in a large scale could storage auditing scenarios without having to setup long experimental runs for very large datasets.

4 Design Specification

The proposed zk-STARK data integrity auditing framework is designed to be modular, cloud native, and serverless application, configured to handle large scale data integrity verification with privacy preservation. The overall architecture is designed to enable event driven data flow from initial data ingestion to final audit reporting. The system comprises of 3 main components; The front-end and backend application, the Zero-knowledge proof system, the cloud infrastructure services (S3,Lambda,DynamoDB,Ec2 and API gateway) that ties it all together. Data is uploaded through the front-end UI and routed using Amazon API gateway service, which securely exposes backend services to the users. API gateway is the entry point for clients requests, handles authentication and authentication as well as traffic management and request throttling. It ensures that all data upload and audit request are properly validated and routed to the appropriate AWS lambda functions. Data gets passed and processed in a lambda function pipeline, where raw uploaded data is split into blocks and cryptographic commitment is applied on it. The data blocks are stored in amazon s3 bucket and the Merkle root and authentication paths are stored in a DynamoDb table as metadata of the uploaded dataset. When a user wants to perform an audit, the request is passed through API gateway, which invokes the backend lambda function responsible for the data sampling. The sampling component selects a subset(random data blocks) of data for audit. The selected data blocks and their associated Merkle paths are inputted into the zero-knowledge proof system running as a containerized rust docker application, which generates STARK proofs of their integrity without revealing details of the underlying data and verification is done on the proof to determine the blocks data integrity. The entire workflow is orchestrated and executed within a cloud infrastructure, providing compute, storage and networking resources. Finally, a user-friendly front-end interface allows organizations to manage audits, monitor progress and view audit results.

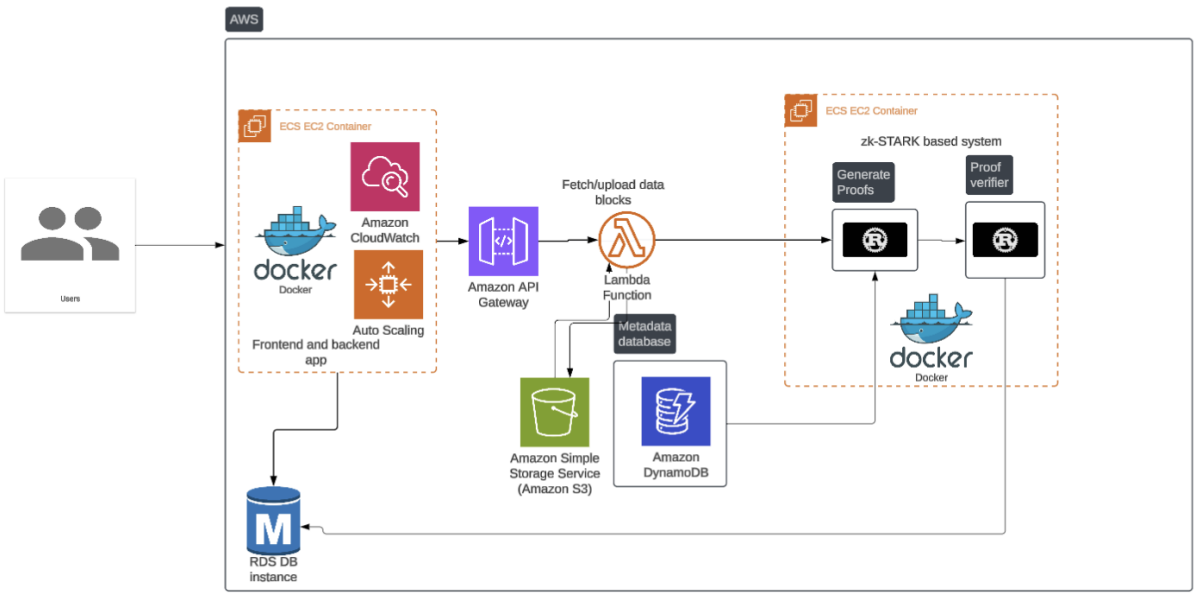


Figure 2: Audit Framework Architecture

5 Implementation

The development of zk-based public auditing framework is implemented using modern technology stack designed for performance, scalability and maintainability. The primary technology stack is composed of Rust for creating zk-STARK prove generation and verification components, Python for backend operations (API, orchestration logic) and React is used to create the front-end user interface. Some key development tools are used to fast track development. Cargo is Rusts build tool and package manager used for dependency management and project compilation. FastAPI is used to manage python backend development and Node package manager (NPM) is used to manage React dependencies.

5.1 Data Pipeline implementation

For the cloud deployment implementation, a pipeline of ingestion, storage and metadata management is created. This is a fundamental layer for the auditing system. This layer handles the processing of uploaded files stored by the CSP (on S3 bucket). This layer also retrieves the data blocks requested for audit by the user from the S3 buckets. There is also the metadata management layer, created using Amazon DynamoDB, that stores the public Merkle root and the Merkle authentication paths of uploaded data sets. The Merkle root (Public input) Data block and authentication path (private inputs) are parsed in the zk-STARK circuit to generate the proofs.
This layer is implemented with S3 and DynamoDB for storage and Lambda functions to execute operations.

5.2 zk-STARK components implementation

This core cryptographic implementation is central to the entire system's ability to provide privacy-preserving data integrity and a trustless system. The STARK proof system is implemented using Winterfell, a Rust based library from Facebook (Meta)(Irakliy, 2021). The system works like this, a data block , the authentication path(sibling hashes) and the Merkle root are provided. The proof system records the computation of Merkle proof (hashing up the path) in the trace, the AIR records the rules for a valid Merkle update and the generated proof allows anyone with the Merkle root to verify the data blocks membership in the tree without re-executing the computation or seeing the full trace.

Below is an overview of the zk-STARK system

Component	**Purpose and function**
Execution trace	This trace provides a complete, step-by-step record of how the leaf hash (data blocks hash) is combined with the authentication path to produce the Merkle root.
AIR (algebraic intermediate representation)	The AIR is a set of algebraic rules that define what a *valid* trace looks like. It specifies the allowed transitions between one row of the trace and the next, and boundary conditions that tie the trace to public inputs.
Prover	The prover is the algorithm that constructs the execution trace and then uses Winterfell's machinery to commit to it and produce a proof. Once the trace is built, the prover runs Winterfell's prove() function. This function commits to each trace column via a Merkle tree, and outputs cryptographic proof.
Verifier	The verifier takes the proof, the public Merkle root and (optionally) other public inputs, and checks that the proof is valid. It doesn't see the full trace; instead it verifies the Merkle commitments, samples a few rows of the computation and evaluates the AIR constraints on those rows. If all checks pass, the verifier returns success, meaning that there must exist some execution trace consistent with the AIR that leads from your block to the public root.

Table 3: 4 Key components essential to a zk-STARK proof and verifier system

These clear constraints ensures that the prover cannot generate proofs without the original data. The zk-STARK system generates succinct proof that the CSP provider has executed the computation correctly.

Input:

Term	**Description**
Block data	The data we want to prove has not been tampered with and is still part of the merkle tree.
Merkle path	A list of sibling hashes and the leaf's position (the "authentication path"). This shows how to combine your block's hash with its siblings to re-compute the tree's root.

Merkle root	The single hash at the top of the tree that commits to all blocks. This public and is used as input to the proof verifier.
Hash function & proof parameters	The hash function used to build the tree (e.g. Sha3-256, Rescue or Blake3) and the STARK proof settings (security level, field extension).
Index bits	Bits indicating whether, at each tree level, your block sits on the left or right. These are used to place sibling hashes correctly during the proof.

Table 4: Inputs into the ZKP system

Output:

Term	**Description**
Merkle proof (authentication path)	The list of sibling hashes needed to recompute the root hash.
STARK proof of membership	A compact, non-interactive proof generated by the Winterfell library. It proves that if you know the Merkle root hash that the computation of "hashing up the path to the root" was done correctly, without revealing any additional internal information.
Verification result (true/false)	When someone runs the verifier against the proof and the public Merkle root, they get back a simple pass/fail: it tells them whether your block is indeed part of the committed data set.

Table 5: Output of the ZKP system

.

```rust
// MERKLE PATH VERIFICATION AIR
// ================================================================================

pub struct PublicInputs {
    pub tree_root: [BaseElement; 2],
}

impl ToElements<BaseElement> for PublicInputs {
    fn to_elements(&self) -> Vec<BaseElement> {
        self.tree_root.to_vec()
    }
}

pub struct MerkleAir {
    context: AirContext<BaseElement>,
    tree_root: [BaseElement; 2],
}
```

Figure 3: AIR code snippet

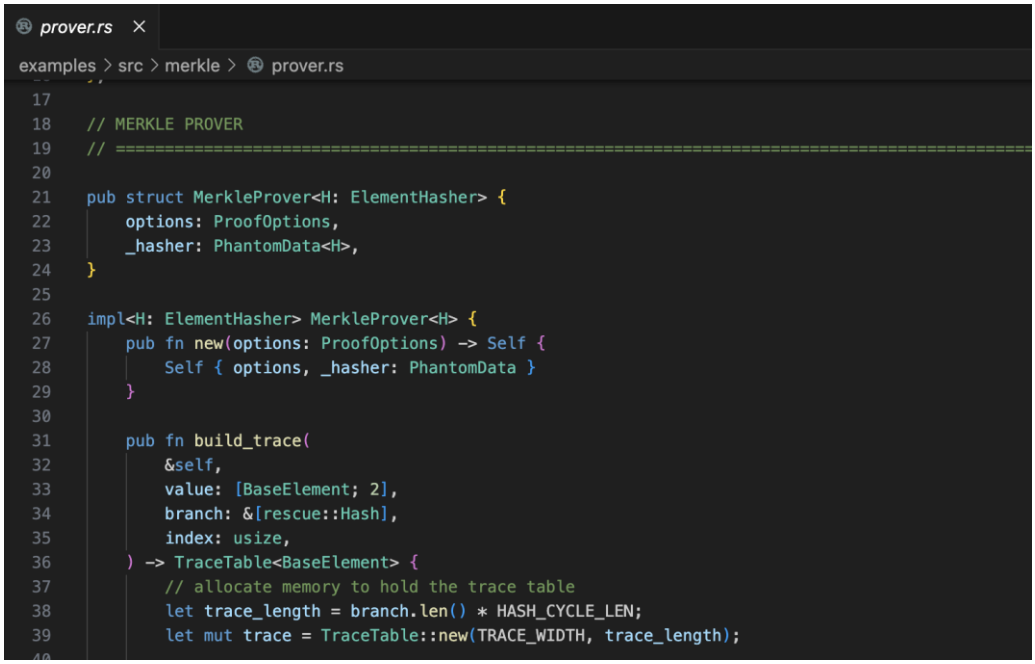


Figure 4: Generate Stark proof code snippet

```
    fn verify(&self, proof: Proof) -> Result<(), VerifierError> {
        let pub_inputs = PublicInputs { tree_root: self.tree_root.to_elements() };
        let acceptable_options =
            winterfell::AcceptableOptions::OptionSet(vec![proof.options().clone()]);
        winterfell::verify::<MerkleAir, H, DefaultRandomCoin<H>, MerkleTree<H>>(
            proof,
            pub_inputs,
            &acceptable_options,
        )
    }

    fn verify_with_wrong_inputs(&self, proof: Proof) -> Result<(), VerifierError> {
        let tree_root = self.tree_root.to_elements();
        let pub_inputs = PublicInputs { tree_root: [tree_root[1], tree_root[0]] };
        let acceptable_options =
            winterfell::AcceptableOptions::OptionSet(vec![proof.options().clone()]);
        winterfell::verify::<MerkleAir, H, DefaultRandomCoin<H>, MerkleTree<H>>(
            proof,
            pub_inputs,
            &acceptable_options,
        )
    }
}
```

Figure 5: Verify Stark proof code snippet

5.3 User Interface implementation

The front-end user interface serves as the primary interaction point between the data owner, the verifier, and the underlying zk-STARK -based auditing framework. It is developed with the using of react.js components to provide user with user friendly interface to manage audits and view audit results.

Key Functional Modules:

1. **Data Upload Interface**

a. Allows the data owner to upload large datasets (CSV, JSON, or binary files) for storage in the cloud.
b. Upon upload, the front end triggers an API request to the backend, which splits the data into fixed-size blocks, generates block hashes, constructs the Merkle tree, and stores the Merkle root and authentication paths in the metadata database.

2. **Audit Request Interface**
   a. Enables the Verifier (or the data owner in a self-audit scenario) to initiate an integrity verification process.
   b. The auditor specifies the dataset ID or selects it from a list of previously uploaded datasets.
   c. The UI communicates with the backend to trigger the zk-STARK proof generation process in the cloud.
   d. Random sampling parameters (number of blocks to audit) can be set by the user to balance between verification confidence and computational cost.
3. **Proof Verification Dashboard**
   a. Displays audit results in real-time, including verification success/failure, verification time, and proof size.
   b. Uses visual indicators (e.g., green checkmarks for valid proofs, red alerts for failures) to simplify interpretation.
4. **Audit History**
   a. Maintains a record of previous audits, including timestamps, dataset identifiers, verification outcomes, and proof verification times.

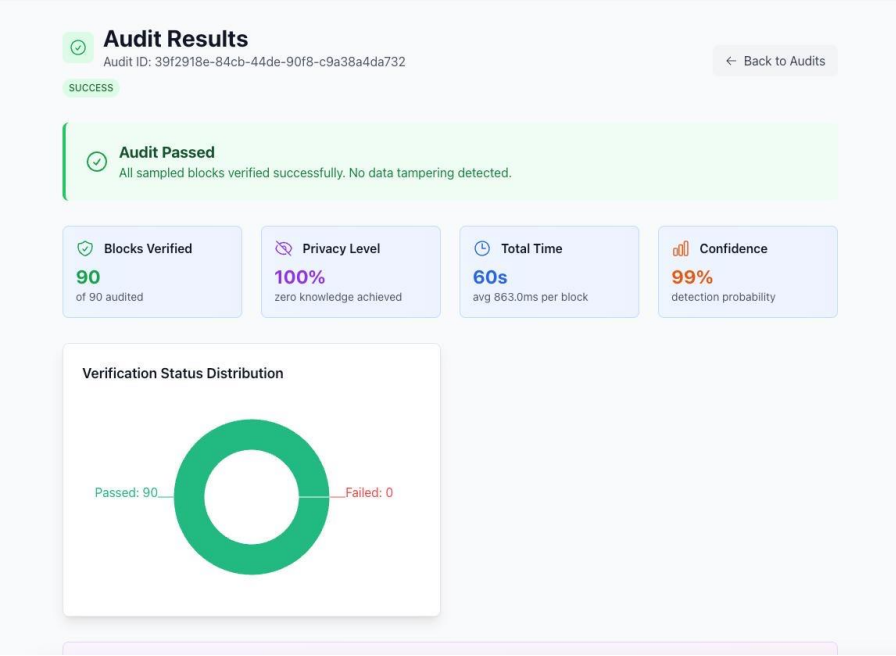
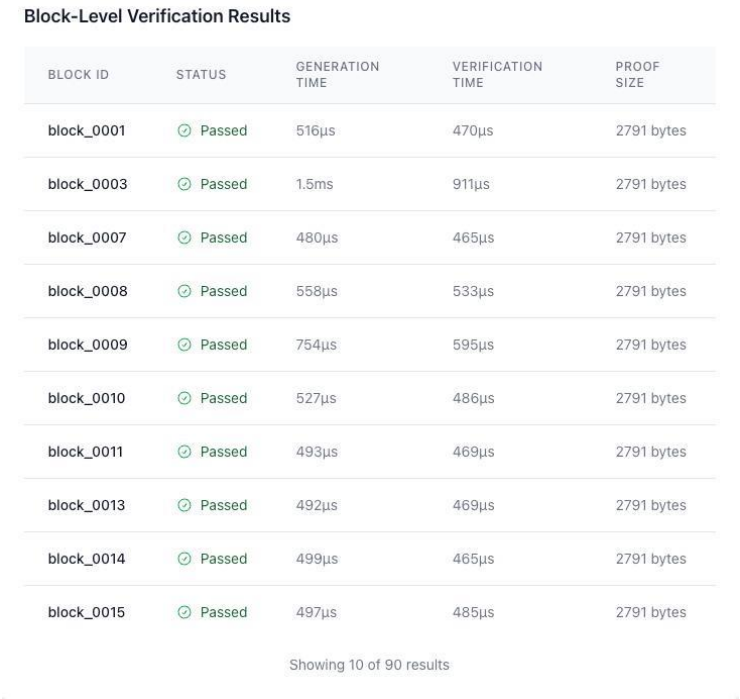


Figure 6: Audit results Dashboard



Block-Level Verification Results

BLOCK ID	STATUS	GENERATION TIME	VERIFICATION TIME	PROOF SIZE
block_0001	Passed	516µs	470µs	2791 bytes
block_0003	Passed	1.5ms	911µs	2791 bytes
block_0007	Passed	480µs	465µs	2791 bytes
block_0008	Passed	558µs	533µs	2791 bytes
block_0009	Passed	754µs	595µs	2791 bytes
block_0010	Passed	527µs	486µs	2791 bytes
block_0011	Passed	493µs	469µs	2791 bytes
block_0013	Passed	492µs	469µs	2791 bytes
block_0014	Passed	499µs	465µs	2791 bytes
block_0015	Passed	497µs	485µs	2791 bytes

Showing 10 of 90 results

Figure 7: Block level verification Results

5.4 Deployment

A modular approach is followed in deploying the zk-STARK based auditing framework. Allowing each component to operate independently while communicating securely with each other. The front-end user interface is hosted on AWS s3 as a static website. This approach removes the need for a dedicated web server for frontend delivery providing low latency content delivery, automated scaling to handle multiple users and high availability through S3's multi-region feature (Deploy React and Server Side Rendered Apps and Static Sites - Amplify Hosting - AWS). The backend application responsible for handling API calls. Data set processing, Merkle tree generation and Audit operations is hosted on Amazon EC2. This offers flexible instance types to that scale compute and memory based on workload. If offers secure connections to AWS dynamoDB, and S3 for metadata and block storage. The zk-STARK prover and verification component is containerized using docker and deployed on an EC2 instance. This isolates the cryptographic computation from the main application logic, enabling the Rust-based winterfell proof and verifier to run without any conflicts from other services.

6 Evaluation

This section provides analysis and discussion of metrics obtained from the research implementation. The findings are sectioned into Privacy analysis, tampered data detection, Proof Generation and Verification results, and Scalability Analysis.

A. Privacy analysis

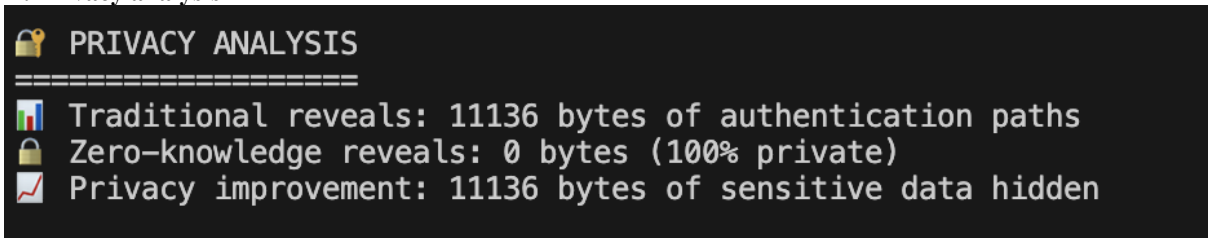


Figure 8: Information leakage analysis

B. Tampered Data detection

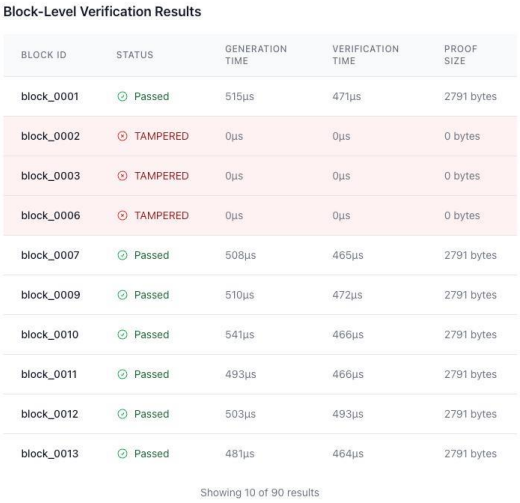


Figure 9: Results of Proof Verification

C. Scalability Analysis

The audit process was simulated on an Amazon EC2 instance configured with t3.medium instance type, 2 vCPU and 4Gb RAM to reflect a real-world cloud deployment environment. A dataset of 200MB is split into a block of 2MB each totalling 100 blocks, each of the blocks is hashed and a Merkle tree root is generated. The Winterfell library is used to generate the zk-STARK proofs of the 100 blocks, and the generation time and verification time is recorded. To estimate performance for larger datasets, the measured per-block computation times were extrapolated linearly to predict the time and computational cost for datasets of 1 GB, 100 GB, and 1 TB.

Dataset size	Blocks (2 MB)	Total generation time (hrs)	Total verification time (s)	Total runtime (s)	Est. EC2 cost (USD)	Total proof size (bytes)	Total proof size (MB)
200 MB	100	0.004306	0.50	16.00	0.0002	279100	0.266
1 GB (1024 MB)	512	0.022044	2.56	81.92	0.0009	1428992	1.363
100 GB (102400 MB)	51,200	2.204444	256.00	8192.00	0.0947	142899200	136.279
1000 GB (1,000,000 MB)	500,000	21.527778	2500.00	80000.00	0.9244	1395500000	1330.853
1 TB (1,048,576 MB)	524,288	22.573511	2621.44	83886.08	0.9694	1463287808	1395.500

Table 6: Scalability Analysis

6.1 Discussion

The zk-STARK based audit framework does not leak any information (figure 9), in the traditional Merkle proof approach, verifying that a leaf (data block) is in a Merkle tree requires all the sibling hash to be provided at each level of the tree. The verifier combines these sibling hashes iteratively to recompute the Merkle root. however, this method reveals sensitive information about every sibling hash in the path, the positional index bits that indicate left and right relationships at each level and the hash output at the intermediate nodes (hash result of two leaves). In auditing, revealing this information can lead to an exploit as relationships can be inferred between stored blocks, to reconstruct sections of the Merkle tree, or attacks can be targets at specific nodes, therefore weaking confidentiality. However, the STARK audit framework eliminates such leakage. Using the winterfell library, the computation of hashing the leaves to build the root is recorded in an execution trace. The AIR (Algebraic intermediate Representation) asserts that the final state the computation is equal to the known public Merkle root. The prover then generates a proof confirming the process was done correctly, while the verifier learns

only the public Merkle root used to check the proof. The Merkle path, index bits and intermediate hashes remain hidden ensuring that during data integrity proof no information about the datasets structure or content is leaked preserving privacy.

As shown in figure 8, the zk-STARK based audit framework can detect tampering of data by verifying that the committed data blocks stored in the cloud match the initial cryptographic commitments generated during the upload of the data. When data is first uploaded, it is split into fixed sized blocks, and each block is hashed using a hash function (sha3-256) and hashes are combined to generate a Merkle tree. The Merkle tree root is a immutable commitment representing the entire data set. During audit process, the verifier will request for a random number of data blocks with the corresponding authentication paths in the Tree. The zk-STARK proof system is used to prove in zero knowledge that the blocks and their paths correctly reconstructs the original Merkle root without revealing the content of the blocks. If any blocks have been changed or data deleted this will change the hash, causing a mismatch between the roots. This will cause the proof to fail, indicating that data has been tampered with. By pairing sampling and cryptographic soundness of zk-STARK, the system will ensure that malicious modification of data detected. The zero knowledge properties ensure that the verifier learns nothing about the data and preserves confidentiality in achieving data integrity checks.
As highlighted in table 6, The performance evaluation demonstrates that the zk-STARK-based auditing protocol achieves extremely low proof generation and verification times per 2 MB data block. For the baseline 200 MB dataset, which consists of 100 blocks, the total proof generation time was measured at only 15 seconds, while verification required 0.05 seconds, resulting in a combined runtime of 15.05 seconds. Overall, the evaluation confirms that the proposed zk-STARK auditing method can handle large-scale datasets efficiently while maintaining strong privacy guarantees. The linear scaling trend indicates that performance will remain predictable as storage volumes grow, and the low per-audit costs make the scheme suitable for deployment in real-world cloud environments where both integrity and confidentiality are critical.

7 Conclusion and Future Work

This research set out to investigate whether zk-STARKs could be used to create a non-interactive, privacy-preserving public auditing framework for outsourced cloud storage, with a focus on cold data. The objectives were to eliminate the trust dependencies found in traditional and past Third-Party Auditor (TPA) models, prevent metadata leakage during verification, and achieve low-latency, scalable proof generation and verification. The proposed solution was designed, implemented, and evaluated on a cloud-based architecture, integrating the Winterfell zk-STARK library with a Merkle tree commitment scheme and deployed on AWS infrastructure. The results demonstrate that the framework successfully addresses the research questions. The system achieves privacy preservation by ensuring that Merkle paths, intermediate hashes, and index bits remain hidden during verification, compared to traditional Merkle proof methods that disclose all sibling hashes and positional metadata. Proof generation and verification times were measured at 0.0.7s and 0.0025s per 2 MB block respectively, with linear scalability up to terabyte-scale datasets, and negligible estimated compute cost on EC2 instances. The tamper detection tests confirmed that even slight modifications in any data block result in proof verification failure, ensuring data integrity. However, the research has some limitations. The current implementation considers only static datasets between audits, supporting dynamic data operations would require redesign of the framework.
This work can be extended by to improving the framework to support dynamic data auditing, enabling efficient proof updates for data insertions, deletions, or modifications without regenerating proofs for the entire dataset.

8 Questions from Second Examiner

Q1

As part of your Research Thesis , you are required to have one clearly defined research question. You currently list 4 separate objectives. Can you please provide a single research question for this work.

Can zk-STARKs be used to create an efficient, privacy-preserving, non-interactive and zero trust auditing framework for outsourced cold cloud data?.

Q2

Can you clarify some of the complications you may have faced when dealing with zero-knowledge proofs from a technical complexity perspective.

I faced some challenges in implementing this research project, one of which is the cryptographic complexity and working with the winterfell library which is written in Rust. in zk-STARK the AIR (algebraic intermediate representation) defines constraints for what a valid trace looks like and my zk-stark system is meant to proof leaf membership of merkle tree which requires ensuring that each leaf hash from leaf to root is done correctly depending on the sibling and index bit, the winterfell library provides the frame work for defining these constraints.
Understanding the Rust language was challenging and how it was used to implement the Stark proof and verifier components (field elements, execution traces, algebraic constraints). To help with this I made use of LLM to assist in explaining the example codes in the github repo. For example what an execution trace is, how does the AIR (algebraic immediate representation) define a valid trace and how the prover and verifier all come together to create the zero knowledge proof system. This enabled me to understand how Winterfell's prover constructs proofs from the trace and how the verifier checks them, so I could integrate those pieces into my cloud-native architecture. I used the available library and documentation to build the zk-component of my research, I did not write every trace constraint from scratch. My main contribution was designing the architecture and integrating the cryptographic components into my cloud-native workflow. Another technical challenge I faced was with running the prove generator and verification zk-STARK components in AWS lambda as the initial architecture design was to an event driven architecture with each operation handled by its own lambda function, however AWS does not support Rust programming language. I tried to work around this by dockerize the Rust components and then upload it to lambda but however the run time to build/compile the rust files took too long (3 hours plus) so i made the decision to use an Ec2 instead and have this components running like it would on a local machine.

Q3
Can you address and clarify which of the papers referenced relates closet to your own research work.

These papers relate closest to my research work:
(Gan et al 2024) who proposed an online/offline auditing framework for cloud storage, just like my work their motivation was focused on cost and efficiency challenges related to auditing outsourced data. However their work relies on challenge-response interactions with a third-party auditor which introduces trust assumptions and communication overhead. My work focused on using zk-STARKs to eliminate interactivity and create a non-interactive zero-knowledge proof system for data integrity verification. Another similar work is (Lin et al 2021) (BPAO) which focused on the elimination of TPA dependency using blockchain and smart contracts. My work aims for the same objective in creating a trustless system but without the overhead that comes with using a blockchain.

Q4
What is the potential industry implementation of such a solution proposed in this work and is there already existing implementations similar to your pre-existing in industry today?

The proposed solution can be implemented in the following industries that outsource sensitive but rarely accessed (cold) data to the cloud and where third party audits are required:

1. Finance & Banking (transaction records auditing).
2. Healthcare (archived patient records under HIPAA/GDPR, prove integrity without exposing sensitive data).
3. Law (archived contracts, evidence files, compliance records).
4. Companies under GDPR (proving data integrity for regulatory audits without leaking user information).

Similar works in the industry:

1. Crypto exchange binance uses zk-SNARKs(alternate to zk-STARKs) and Merkle trees to perform proof of reserves (to prove it holds users assets) without revealing their balances.Binance
2. Qedit uses zk proofs to "enable companies to accelerate growth, mitigate risk and monetize insights via regulatory-compliant, cross-organizational data collaborations". Qedit

8.1.1 References

Gan, Q. et al. (2024) 'Online/offline remote data auditing with strong key-exposure resilience for cloud
storage', Computer Standards & Interfaces, 88, p. 103798. Available at: https://doi.org/10.1016/J.CSI.2023.103798

Lin, Y. et al. (2021) 'Blockchain based Public Auditing Outsourcing for Cloud Storage', Proceedings
of the International Conference on Parallel and Distributed Systems - ICPADS, 2021-December, pp.
482–489. Available at: https://doi.org/10.1109/ICPADS53394.2021.00066.

References

1. Armknecht, F. *et al.* (2014) 'Outsourced proofs of retrievability', *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 831–843. Available at: https://doi.org/10.1145/2660267.2660310;TOPIC:TOPIC:CONFERENCE-COLLECTIONS>CCS;PAGE:STRING:ARTICLE/CHAPTER.
2. Ateniese, G. *et al.* (2007) 'Provable data possession at untrusted stores', *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 598–610. Available at: https://doi.org/10.1145/1315245.1315318.
3. Ben-Sasson, E. *et al.* (2018) 'Scalable, transparent, and post-quantum secure computational integrity', *Cryptology ePrint Archive* [Preprint]. Available at: https://eprint.iacr.org/2018/046.
4. Ben-Sasson, E. *et al.* (no date) 'Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture'. Available at: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.
5. *Binomial Distribution: Uses & Calculator - Statistics By Jim*. Available at: https://statisticsbyjim.com/probability/binomial-distribution/.
6. *Cold Data Storage Explained | Seagate UK*. Available at: https://www.seagate.com/gb/en/blog/what-is-cold-data-storage/.
7. *Deploy React and Server Side Rendered Apps and Static Sites - Amplify Hosting - AWS*. Available at: https://aws.amazon.com/amplify/hosting/?trk=db9bf51a-4056-4b40-8c97-f890622956c3&sc_channel=ps&ef_id=CjwKCAjwwNbEBhBpEiwAFYLtGCYPL3Jx40-youajlAFm9eBN9uuoKpN7wEIt_wTLc-gaZfZza7i44xoCstQQAvD_BwE:G:s&s_kwcid=AL!4422!3!647301968344!p!!g!!static%20website!19621293733!148358899409&gad_campaignid=19621293733&gbraid=0AAAADjHtp8WNOq1iH2OYnttPiCWs3QnG&gclid=CjwKCAjwwNbEBhBpEiwAFYLtGCYPL3Jx40-youajlAFm9eBN9uuoKpN7wEIt_wTLc-gaZfZza7i44xoCstQQAvD_BwE.
8. Gan, Q. *et al.* (2024) 'Online/offline remote data auditing with strong key-exposure resilience for cloud storage', *Computer Standards & Interfaces*, 88, p. 103798. Available at: https://doi.org/10.1016/J.CSI.2023.103798.
9. Goldwasser, S., Micali, S. and Rackoff, C. (1985) 'KNOWLEDGE COMPLEXITY OF INTERACTIVE PROOF-SYSTEMS.', *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pp. 291–304. Available at: https://doi.org/10.1145/22145.22178/ASSET/56A706C0-9F15-409B-BFBD-CCAF1D63C847/ASSETS/22145.22178.FP.PNG.
10. Ben Haj Frej, M., Dichter, J. and Gupta, N. (2018) 'Light-weight accountable privacy preserving (LAPP) protocol to determine dishonest role of third party auditor in cloud auditing', *2018 IEEE International Conference on Consumer Electronics, ICCE 2018*, 2018-January, pp. 1–6. Available at: https://doi.org/10.1109/ICCE.2018.8326350.

11. Irakliy, K. (2021) *facebook/winterfell: A STARK prover and verifier for arbitrary computations*. Available at: https://github.com/facebook/winterfell.
12. Juels, A. and Kaliski, B.S. (2007) 'Pors: Proofs of retrievability for large files', *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 584–597. Available at: https://doi.org/10.1145/1315245.1315317;TOPIC:TOPIC:CONFERENCE-COLLECTIONS>CCS;PAGE:STRING:ARTICLE/CHAPTER.
13. Lin, Y. *et al.* (2021) 'Blockchain based Public Auditing Outsourcing for Cloud Storage', *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2021-December, pp. 482–489. Available at: https://doi.org/10.1109/ICPADS53394.2021.00066.
14. *Merkle Tree — iden3 0.1 documentation*. Available at: https://iden3-docs.readthedocs.io/en/latest/iden3_repos/research/publications/zkproof-standards-workshop-2/merkle-tree/merkle-tree.html.
15. Rukhin, A. *et al.* (2010) 'A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications', *NIST* [Preprint]. Available at: https://doi.org/10.6028/NIST.SP.800-22r1a.
16. Tian, J., Wang, H. and Wang, M. (2021) 'Data integrity auditing for secure cloud storage using user behavior prediction', *Computers & Security*, 105, p. 102245. Available at: https://doi.org/10.1016/J.COSE.2021.102245.
17. *Top 10 Zero Knowledge-Proof Applications to Know*. Available at: https://www.infisign.ai/blog/zero-knowledge-proof-applications.
18. *Understanding the Monte Carlo Simulation | Baeldung on Computer Science*. Available at: https://www.baeldung.com/cs/monte-carlo-simulation.
19. Wang, C. *et al.* (2010) 'Privacy-preserving public auditing for data storage security in cloud computing', *Proceedings - IEEE INFOCOM* [Preprint]. Available at: https://doi.org/10.1109/INFCOM.2010.5462173.
20. Wang, C. *et al.* (2013) 'Privacy-preserving public auditing for secure cloud storage', *IEEE Transactions on Computers*, 62(2), pp. 362–375. Available at: https://doi.org/10.1109/TC.2011.245.
21. Wang, H. *et al.* (2024) 'An improved identity-based public audit protocol for cloud storage', *Heliyon*, 10(16). Available at: https://doi.org/10.1016/j.heliyon.2024.e36273.
22. Wang, Q. *et al.* (2011) 'Enabling public auditability and data dynamics for storage security in cloud computing', *IEEE Transactions on Parallel and Distributed Systems*, 22(5), pp. 847–859. Available at: https://doi.org/10.1109/TPDS.2010.183.
23. Yang, A. *et al.* (2021) 'Lightweight and Privacy-Preserving Delegatable Proofs of Storage with Data Dynamics in Cloud Storage', *IEEE Transactions on Cloud Computing*, 9(1), pp. 212–225. Available at: https://doi.org/10.1109/TCC.2018.2851256.
24. *Zero Knowledge Proofs: Enhancing Blockchain Scalability* (2024). Available at: https://starkware.co/blog/scaling-blockchains-with-zero-knowledge-proofs/.
25. *Zero-knowledge proof - Wikipedia*. Available at: https://en.wikipedia.org/wiki/Zero-knowledge_proof.
26. *Zero-Knowledge Proofs in Blockchain: Ultimate Scalability Guide*. Available at: https://www.rapidinnovation.io/post/zero-knowledge-proofs-in-blockchain-enhancing-privacy-and-scalability.
27. *Zero-Knowledge Succinct Transparent Argument of Knowledge (zk-STARK)*. Available at: https://blog.ueex.com/crypto-terms/zero-knowledge-succinct-transparent-argument-of-knowledge-zk-stark/.
28. Zhang, Y. *et al.* (2017) 'Efficient Public Verification of Data Integrity for Cloud Storage Systems from Indistinguishability Obfuscation', *IEEE Transactions on Information Forensics and Security*, 12(3), pp. 676–688. Available at: https://doi.org/10.1109/TIFS.2016.2631951.
29. *zkVerify: Optimizing ZK Proof Verification At Scale - Delphi Digital*. Available at: https://members.delphidigital.io/reports/zkverify-optimizing-zk-proof-verification-at-scale.