UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Building a Permissionless Blockchain Based on Proof-Of-Space for Archiving the World Wide Web

Bruno Miguel Carvalho Cotrim

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor Bernardo Luís da Silva Ferreira
Prof. Doutor Miguel Ângelo Marques de Matos

2025

"The more you know, the more you realize you do not know"
Aristotle

Acknowledgments

I would like to sincerely thank my advisors, Bernardo Ferreira and Miguel Matos, for their invaluable guidance and support throughout this journey

To Bernardo Ferreira, I am deeply grateful for his mentorship, which began during my Bachelor's degree and continued during my Master's studies. His guidance not only helped me navigate the academic environment and define my path, but also enriched my knowledge in Distributed Systems, Security, and related areas. His constant support, both academically and personally, has been fundamental to my growth.

To Miguel Matos, I extend my thanks for his expertise in Proof-of-Space systems and for the many insightful discussions that greatly enhanced the quality of this thesis. Although being only accompanied by him in this thesis environment and despite being from a different institution, he was always available, willing to help, and committed to improving this work and my research approach.

Both advisors contributed significantly not only to the development of this thesis, but also to related works beyond it, such as the *Arquivo.pt Award 2025* and *Inforum – 16th National Symposium on Informatics*, which required substantial work hours and effort to deliver our best results.

I am profoundly grateful for their mentorship, encouragement, and, above all, for their unwavering belief in me every step of the way.

I would like to express my deepest gratitude to my family (especially my mom and stepdad) for their constant support, encouragement, and understanding throughout the course of this work. Their sacrifices and dedication made it possible for me to pursue my studies and follow the path I believe in. Their patience and belief in me have been truly invaluable, and I am profoundly grateful for their love and guidance.

To my girlfriend, Carolina, thank you for your patience and encouragement throughout this journey. Your support inspired me to stay motivated and give my all to this work. I'm especially grateful for offering some of your creativity, which helped me choosing the best colors and producing the designs of this work.

I would like to sincerely thank my friend André Páscoa, who guided my decision to pursue a Bachelor's in Computer Science. At a time when I was indecisive and unsure about which degree to choose, André introduced me to computer science, patiently explaining many concepts that ultimately inspired me to follow this path. His help was invaluable, particularly during the early stages of my studies, and his friendship has been a constant source of support and encouragement.

I would also like to thank Miguel Carvalho, whom I met in the second year of my Bachelor's. From that moment on, he became not only a great friend but also my main colleague in every college project. His great friendship and teamwork were essential in achieving our academic goals, and, more importantly, made the journey through college an enjoyable experience.

This work was supported by FCT through project APOSTLE - new Approaches for Secure Distributed Data Processing and Access, ref. 2023.12254.PEX, DOI 10.54499/2023.12254.PEX, and the LASIGE Research Unit, ref. UID/408/2025.

Resumo

Com o crescimento exponencial da tecnologia e conteúdo digital, arquivos web como a Wayback Machine e o Arquivo.pt enfrentam um aumento significativo no volume de dados arquivados. O Arquivo.pt, gerido pela Fundação para a Computação Científica Nacional (FCCN), armazena atualmente 1 Petabyte de dados comprimidos em 77 servidores, com 2180 vCPU, 18 TB de RAM e 1234 discos rígidos, e regista um crescimento exponencial de dados armazenados ao longo dos anos. Sistemas centralizados, embora confiáveis, limitam a escalabilidade, pois dependem de infraestruturas geridas por uma única entidade, aumentando custos e riscos de falhas concentradas. Centralização juntamente com o crescimento de dados leva a custos de armazenamento e operacionais elevados, tornando essencial uma solução escalável para mitigar estas despesas.

Este trabalho investiga como a tecnologia blockchain não permissionada pode contribuir para descentralizar e escalar o armazenamento de arquivos web. Uma blockchain é um registo distribuído que mantém, de forma imutável, segura e transparente, um histórico de transações. Estas transações podem ir desde transferências de criptomoedas até à execução de programas descentralizados através de contratos inteligentes, incluindo, no contexto relevante, armazenamento distribuído. Para tal, a blockchain resolve o clássico problema dos generais bizantinos, onde participantes honestos alcançam consenso mesmo perante maliciosos

As blockchains permissionadas oferecem ambientes controlados, onde autoridades centrais verificam a identidade dos participantes, sacrificando descentralização e transparência. As não permissionadas permitem a participação aberta de qualquer utilizador, que pode juntar-se simplesmente escutando a rede. Este modelo elimina pontos únicos de falha e reforça a descentralização. Contudo, blockchains não permissionadas estão sujeitas a ataques Sybil, em que um adversário cria múltiplas identidades falsas para obter poder de voto desproporcional, comprometendo o consenso e até o controlo total do sistema.

Para mitigar ataques Sybil, recursos não forgáveis são utilizados de forma a fazer com que a criação de identidades falsas seja dispendioso. No entanto as soluções de blockchain não permissionadas enfrentam desafios significativos relativamente à utilização ineficiente desses recursos.

A Bitcoin foi pioneira mitigando ataques Sybil ao associar o poder de voto ao poder computacional, através da prova de trabalho (*proof-of-work* — PoW). Neste modelo, participantes investem em hardware e energia para validar blocos, o que torna dispendiosa a criação de identidades falsas. Embora eficaz, este mecanismo exige equipamento especializado e provoca consumo energético massivo, levantando questões ambientais e barreiras à participação.

Como alternativas ao PoW, surgiram vários modelos. A prova de participação (*proof-of-stake* — PoS) liga o poder de voto à quantidade de criptomoeda dedicada à rede rede. Embora energeticamente eficiente, concentra riqueza nos participantes mais ricos, promovendo centralização e riscos de segurança, já que os participantes mais ricos podem adquirir a maioria do recurso e controlar o sistema.

As provas de espaço (*proof-of-space* — PoSp) utilizam espaço em disco como recurso não forjável, no qual os participantes demonstram ter reservado uma quantidade específica de espaço para a rede. Oferecem uma alternativa eficiente ao PoW, dado que o disco é mais barato e consome menos energia do que recursos computacionais. A blockchain Chia é baseada em PoSp, mas, apesar da eficiência energética, a participação em Chia exige que os participantes preencham o disco com dados aleatórios sem utilidade além do consenso da rede.

O Chia Bread Pudding estende o PoSp ao permitir que os participantes integrem dados privados locais nas provas criptográficas. Esta abordagem alcança até 50% de espaço útil. Contudo, como os participantes têm interesse natural em preservar os próprios dados corretamente, o protocolo não garante armazenamento nem integridade necessários em sistemas de armazenamento descentralizado. Além disso, necessita a resolução de questões críticas de segurança e uma adaptação costumizada ao objetivo pretendido. Foi também identificado que a utilidade do espaço e a eficiência podem ser otimizadas.

Provas de armazenamento (*proof-of-storage* — PoSt) também recorrem ao espaço em disco como recurso não forjável. Permitem incorporar dados de clientes nas provas criptográficas, reduzindo desperdício e possibilitando a verificação periódica da integridade e armazenamento desses dados. A blockchain Filecoin expande o PoSt através de provas de replicação (*proof-of-replication* — PoRep) e de espaço-tempo (*proof-of-space-time* — PoSpT), suportando uma rede descentralizada de armazenamento onde os participantes prestam serviços a clientes. No entanto, a Filecoin exige hardware especializado, o seu algoritmo de consenso levanta questões em aberto e, no nosso caso, a recuperação dos dados é lenta, podendo demorar vários minutos.

Estes desafios destacam a necessidade de um sistema que combine consenso eficiente baseado em espaço com integração de dados úteis nas provas criptográficas e garantias de armazenamento verificável contínuo. Possibilitando a construção de uma blockchain que suporte armazenamento descentralizado, seguro e eficiente, onde os participantes oferecem serviços de armazenamento e são incentivados por recompensas em criptomoeda.

Em resposta a estes desafios, este trabalho apresenta a ArchiveChain, uma nova blockchain não permissionada concebida para armazenamento distribuído, económico e verificável de arquivos web, que conta com um mecanismo de consenso inovador que utiliza recursos eficientemente. O objetivo é permitir que arquivos web escalem com o crescimento contínuo da informação, e consigam reduzir custos de armazenamento e operação.

A ArchiveChain alcança estes objetivos através de um mecanismo de consenso baseado em provas de espaço útil (*proof-of-useful-space* - PoUSp), que utiliza dados de arquivo web como recurso não forjável para atingir consenso eficiente na blockchain. Neste modelo, os participan-

tes dedicam espaço em disco à blockchain, preenchendo-o com provas geradas num processo de inicialização intencionalmente demorado, de forma a prevenir ataques Hellman, em que adversários tentam simular espaço recorrendo a computação. Durante esta fase, os dados de arquivo são processados em blocos e, para cada bloco, é construída uma árvore de Merkle que comprova o espaço alocado. Para introduzir custo temporal, o bilhete de participação no consenso é calculado a partir de uma função de hash demorada (*slow timed hash function* - SLOTH), que incentiva o armazenamento dos dados e dos bilhetes em disco a baixo custo, em comparação com o cálculo em tempo real no momento do desafio.

Periodicamente, os participantes são desafiados a provar espaço, produzindo blocos com provas PoUSp para a rede. O desafio é derivado da prova de tempo (*proof-of-time* - PoT) do bloco anterior, e cada participante procura o bilhete de melhor qualidade relativamente ao desafio, recompõe uma prova de Merkle para uma folha aleatória e gera um bloco provisório contendo a prova PoUSp (composta pela prova de Merkle e pelo bilhete SLOTH). Este bloco é então validado, garantindo que estende a cadeia mais longa, e o participante que o produziu recebe a recompensa. Para mitigar ataques de longo alcance e simulações sem custo, o protocolo alterna entre PoT e PoUSp.

O mecanismo de PoUSp apenas permite que dados de arquivo sejam utilizados como recurso não forjável, mas não garante o seu correto armazenamento. Para colmatar esta limitação, é explorado um mecanismo de prova de posse de dados (*proof-of-data-possession* - PDP), que o complementa com verificação contínua de armazenamento e integridade, permitindo que os arquivos web terceirizem de forma segura os seus dados para participantes não confiáveis da blockchain. Este mecanismo inicia-se com a fase de arquivo, na qual, para prevenir ataques de terceirização, os ficheiros armazenados por um participante são transformados em codificações exclusivas vinculadas a esse participante, utilizando cifra simétrica ou codificação SLOTH baseada em tempo. Em seguida, um contrato assinado com informações de verificação é enviado para a blockchain.

Na fase de desafio, para garantir o armazenamento correto dos ficheiros, contratos inteligentes geram janelas periódicas de desafios aleatórios, definidas por alturas de blocos da blockchain, sendo cada desafio derivado de blocos precedentes para assegurar imprevisibilidade. Durante estas janelas, os participantes devem fornecer provas de Merkle relativas a segmentos aleatórios dos ficheiros, e novas janelas são agendadas, garantindo probabilisticamente a manutenção correta do armazenamento ao longo do tempo. O cumprimento ou falha nestes desafios resulta em incentivos ou penalizações em criptomoedas, de acordo com o comportamento dos participantes. Para aceder aos dados, os utilizadores ou o arquivo web podem solicitar os dados codificados aos participantes e descodificá-los de forma eficiente. Deste modo, o acesso terceirizado aos dados reduz significativamente os custos operacionais do arquivo web.

A ArchiveChain atinge uma taxa de espaço útil de aproximadamente $\approx 97\%$ no mecanismo de consenso, superando sistemas PoSp anteriores, e garante que o poder de voto é proporcional ao espaço alocado, um fator crucial para assegurar equidade e resistência a ataques Sybil. Apresenta provas PoUSp e PDP compactas, permitindo escalabilidade, e a sua arquitetura descentralizada

proporciona velocidades de download e de renderização de páginas mais rápidas, melhorando a qualidade de serviço dos arquivos web. Esta solução oferece, assim, um caminho viável para que os arquivos web acompanhem o crescimento contínuo dos dados, reduzam custos e mantenham armazenamento seguro e verificável num ambiente de blockchain não confiável.

Palavras-chave: Blockchain, Arquivos Web, Prova-de-Espaço-Útil, Prova-de-Armazenamento, Armazenamento terceirizado

Abstract

As technology advances and digital content grows, web archives such as the Wayback Machine and Arquivo.pt face an exponential increase in the volume of websites and data archived, significantly increasing storage and operational costs. A scalable solution to mitigate these expenses is essential. This work investigates how these costs can be mitigated through an incentive-based, permissionless blockchain model. We introduce ArchiveChain, a blockchain system built through Proof-of-Useful-Space (PoUSp) and Proof-of-Time consensus mechanisms, alongside Proof-of-Data-Possession (PDP) primitives, that enable Web Archives to safely outsource data to the blockchain, providing verifiable, secure, and efficient archival of data with rapid accessibility. ArchiveChain aims to enhance energy efficiency and reduce participation costs compared to current blockchain solutions, leveraging the useful Web Archive data as a resource in its consensus rather than random, useless data approaches like those employed by Chia. By incentivizing network participants to archive data through cryptocurrency rewards, ArchiveChain provides a sustainable model that reduces the rising costs of web archiving while enabling broader applications in decentralized systems, including cryptocurrencies, smart contracts, and more. Our prototype achieves a $\approx 97\%$ useful space ratio, significantly outperforming prior Proof-of-Space systems, and ensures proportional voting power to storage allocation, which guarantees fairness and Sybil resistance. ArchiveChain's compact PoUSp and PDP proofs support scalability, while its decentralized architecture enables faster download speeds and reduced page render times, improving quality of service for web archival systems. This solution offers a viable path for Web Archives to scale with growing data demands, reduce operational costs, and maintain secure, verifiable storage in a trustless environment.

This work culminated in the publication *ArchiveChain: a Permissionless Blockchain based on Proof-of-Useful-Space for Archiving the World Wide Web* at the INForum 2025 Conference, Évora, Portugal, and was further recognized with the *Award Arquivo.pt 2025 – DNS.PT*, an award highlighting the best academic work.

Keywords: Blockchain, Web Archiving, Proof-of-Useful-Space, Proof-of-Storage, Outsourced storage

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

With technological advancements and the expansion of digital content, web archives such as the Wayback Machine [4] and Arquivo.pt [1] have experienced an exponential increase in the volume of websites and data archived. This, in turn, leads to higher storage and operational costs, creating a need for a scalable solution that can reduce rising costs effectively. To help scale Web Archives, we focus on Arquivo.pt, overseen by the Foundation for National Scientific Computing (FCCN) [2], as our test subject, to help scale it and reduce its operational costs. Current numbers indicate that Arquivo.pt is storing 1 Petabyte (PB) of compressed data (websites, images, and videos), and is using 77 servers with 2180 vCPU, 18 TB of RAM and 1234 hard drives (5.18 PB) [3] and is facing an exponential increase in the storage of archived contents over the years, as can be seen in Figure 1.1. This exponential growth of archived data, combined with the limited scalability of traditional centralized systems, results in increased storage and operational costs, thus creating a need for a more scalable solution to reduce these costs.

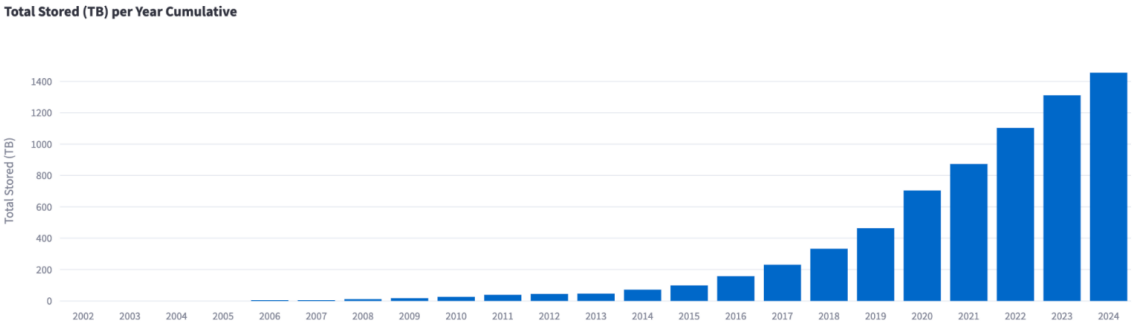

Figure 1.1: Evolution of the cumulative amount of data archived by Arquivo.pt over the years. [3]

The Blockchain technology has emerged as a prominent area of research and has undergone continuous evolution since its appearance. Unlike conventional architectures, blockchains provide scalable, decentralized, secure, and transparent infrastructures, making them a promising solution for a wide range of real-world applications, such as electronic voting, financial services, asset tokenization, and, most importantly, for the described problem, decentralized storage networks.

Despite these advantages, many blockchain based solutions still suffer from issues related to resource inefficiency, which often lead to significant environmental concerns.

Nevertheless, there remains a strong motivation to enhance the practicality and deployability of blockchain in real-world scenarios where energy efficiency and resource conservation are critical. This motivation aligns with the current growing trend toward the development of more efficient blockchain mechanisms[1], many of them applied to decentralized storage networks.

With this goal in mind, this thesis will explore the use of blockchain technologies for distributed data storage and scale centralized web archive infrastructures, and, along with a self-incentive mechanism, reduce storage and operational costs of these services.

1.2 Challenges

A blockchain is an implementation of a distributed ledger that allows for secure, immutable, transparent, and tamper-proof transactions, reaching consensus on a ledger view without the need for a central authority. These transactions can range from a simple cryptocurrency to executing programs through smart contracts and, most relevant to our case, distributed storage.

While permissioned blockchains offer controlled environments through central authorities, at the expense of decentralization and transparency, permissionless blockchains allow open participation, meaning that any participant can join simply by listening to the network. This full decentralization, in which no trusted central authority is present, establishes a trustless environment where participants are not required to place trust in one another, but rather in the combination of the protocol and the underlying cryptographic primitives called distributed trust. However, permissionless blockchains face challenges related to Sybil attacks, where adversaries create multiple fake identities to gain unfair influence that compromises the network and renders classical structured-based consensus ineffective. With enough fake identities an attacker could increase their voting power to a point where they can influence the consensus mechanism, possibly gaining control of the network and potentially modifying the blockchain, breaking its properties.

To address this issue, non-forgeable resources are often employed to make identity forgery prohibitively expensive. Nevertheless, current blockchain solutions face significant challenges due to the inefficient utilization of such resources.

Bitcoin's [59] **Proof-of-Work (PoW)** mitigates Sybil attacks by tying voting power to computational power. To increase such computational power, participants must expand their hardware resources, which in turn raises energy consumption and imposes an economic cost that prevents attackers from creating multiple fake identities. However, this Sybil-resistant mechanism requires significant energy consumption (comparable to that of small countries [41]), raising serious environmental concerns. Moreover, boosting computational resources requires significant investment in costly, powerful, specialized hardware and resulting high operational energy costs, further increasing the barriers to participation in the blockchain.

Research into alternatives to PoW has produced several promising directions.

[1] https://www.blockchain-council.org/blockchain/consensus-mechanisms-in-blockchain/

Proof-of-Stake (PoS) emerged as a promising approach that links voting power to the amount of cryptocurrency a participant stakes in the network. A successful example is Ethereum 2.0 [20, 34] that recently successfully replaced its original PoW based protocol with a PoS based one. Despite its energy efficiency advantages, PoS tends to concentrate wealth among participants who stake more coins, leading to centralization and potential security risks, as wealthier holders may acquire a majority of the network stake and effectively control the network indefinitely.

Proof-of-Space (PoSp) [31] leverages storage space as a non-forgeable resource for blockchain consensus, where participants demonstrate that they have allocated a specific amount of data to the network, providing an efficient alternative to PoW, as disk space is cheaper and less energy-intensive than computational resources. In PoSp, participants commit disk space by generating cryptographic proofs during an initialization phase, which is computationally expensive to prevent Hellman time–memory tradeoff attacks [38], making the process time-consuming. This one-time cost incentivizes storing proofs, which can be reused over time, effectively amortizing the initialization cost. The **Chia** [23] blockchain has emerged as one of the most successful blockchains based on PoSp. However, despite being more energy-efficient, participation in Chia requires farmers to fill their disk space with random data that serves no purpose other than maintaining blockchain security, resulting in wasted storage, which has grown to 18.7 EiB, and consequent eletronic waste.

Chia Bread Pudding [44] extends Chia's PoSp by embedding a user's private local data, already present on their disks, within cryptographic proofs. This approach reduces the need for storing useless data and achieves a 50% useful space ratio. However, since users are incentivized to correctly store their own data, the scheme does not provide the storage and integrity verification required for decentralized storage systems. It also needs addressing in key security concerns, and the need for its implementation to be tailored to decentralized storage use cases. Additionally, we identified that its useful space ratio and initialization efficiency can be further improved. Nonetheless, it serves as an important theoretical inspiration for our proposed consensus scheme.

Proof-of-Storage (PoSt) leverages storage space as a non-forgeable resource for blockchain consensus, similar to Chia's PoSp, but overcomes PoSp's limitation of storing random, useless data. PoSt enables the storage of meaningful data within cryptographic proofs, allowing a verifier to periodically confirm that a prover correctly maintains specific data, thereby ensuring integrity, retrievability, and accountability through mechanisms such as Proof-of-Data-Possession (PDP) [45, 67, 70]. To prevent outsourcing attacks, where provers might fetch data from others, PoSt relies on unreproducible unique encodings. These include client encodings based on secret keys (e.g., AES-256) or time-consuming verifiable delay encoders (VDE) [55], which can be quickly decoded using the corresponding verifiable delay decoder (VDD). Both approaches are costly to regenerate. This ensures continuous storage, since if a prover loses or corrupts data, they can no longer reproduce it to respond to challenges. **Filecoin** extends PoSt with Proof-of-Replication (PoRep) and Proof-of-Spacetime (PoSpT), supporting decentralized storage of client data [51]. Miners provide ongoing proofs of availability and are rewarded via Expected Consen-

sus or successful storage proofs based on their storage contribution. However, Filecoin's reliance on specialized hardware limits adoption among storage providers, and its complexity and slow retrieval speeds reduce its suitability for latency-sensitive applications, such as accessing web-archived pages.

Additionally, in PoS, PoSp, and PoSt systems, generating consensus proofs is cheap, as participants are not required to perform computationally intensive operations. However, this efficiency also exposes the network to several security threats. The most notable are costless simulation (manipulating block payloads to increase the likelihood of future rewards), long-range attacks (where entities with substantial historical resources attempt to construct a longer alternative chain), and double-dipping attacks (simultaneously extending multiple competing chains).

As highlighted by the challenges faced in previous blockchains and Sybil-proof consensus mechanisms, designing a blockchain that enables a decentralized storage network capable of securely outsourcing Web Archive data through self-incentivization is far from trivial. Beyond these consensus-related difficulties, there are also distributed storage challenges that must be addressed:

- Creating a stable blockchain network where the consensus mechanism relies on resource efficient-primitives;
- Scaling the blockchain network into a fully distributed storage system able to securely serve Web Archive data to clients, providing verifiable storage, protection against outsourcing attacks, efficient proofs and data retrievals, and offering a cost reduction;
- Definition of an incentive model that rewards farmers (i.e., ordinary people that will contribute to the network) for storing Arquivo.pt's files and maintaining the network;
- Defining the role of the Web Archive in the network and how will it interact with the system, in such a way that makes sense for them and that allows for a great cost reduction compared with Arquivo.pt's current version;
- Other distributed systems concerns that should be considered such as how to efficiently disseminate messages through the network, signature schemes and key managements, and storage schemes.

1.3 Objectives and Solution

In light of these limitations we aim to answer the following question: "*Can we design a scalable permissionless blockchain that expands Web Archives by leveraging the archival data as a non-forgeable resource for a resource-efficient consensus mechanism, while integrating storage and integrity verification primitives to ensure secure and verifiable outsourcing of data to untrusted participants, thereby achieving both useful work consensus and reliable decentralized storage?*"

Our answer to this question is *ArchiveChain*, a novel permissionless blockchain specifically designed to support distributed, cost-efficient, and verifiable storage of web archival data, while relying on a resource-efficient Sybil-proof consensus mechanism. This allows Web Archives to scale with the continuous growth of information while simultaneously reducing storage and operational costs. Network farmers are incentivized through our native cryptocurrency, *ArchiveCoin*, for providing storage resources to Web Archive services, as well as for maintaining the blockchain and contributing to consensus.

We achieve this by combining:

- Proof-of-Useful-Space (PoUSp), which avoids the high energy and wasteful data consumption of other blockchains by enabling the encoding of any meaningful data within consensus PoUSp proofs. Most importantly, it allows the system to leverage Web Archive data itself as the storage resource for maintaining the Sybil-resistant consensus mechanism.

- Proof-of-Data-Possession (PDP), which complements PoUSp by enabling continuous file storage and integrity verification, thereby providing verifiable storage. This allows Web Archives to safely outsource data to untrusted blockchain farmers, with proofs derived from the same data as PoUSp, thus extending the notion of useful consensus to our broader objective.

- Verifiable Delay Functions (VDF), which enforce real-time delays by requiring the execution of sequential, inherently unparallelizable functions. We explore the Wesolowski's [74] VDF to implement Proof-of-Time (PoT), crucial for mitigating long-range and costless simulation attacks. Additionally, we also explore VDFs to increase the initialization cost of PoUSp through the Slow-Timed Hash Function (SLOTH) [55]. An adaptation of SLOTH into a VDE will be studied, and it enables time-consuming file encoding as a deterrent against outsourcing attacks.

Our system achieves a $\approx 97\%$ useful space ratio, representing a significant improvement over previous PoSp systems. The system ensures that the amount of storage space allocated by each farmer is directly proportional to their voting power, thereby promoting fairness and security against Sybil attacks. Both PoUSp and PDP proof schemes proved to be compact and efficient, something crucial given the scalability concerns of blockchains. Additionally, ArchiveChain offers a potential improvement in the quality of service for web archives. Its decentralized architecture and support for direct file access enable faster download speeds and reduced page render times.

Most importantly, our prototype demonstrated that employing a permissionless blockchain to build a distributed storage network, enabling the safe outsourcing of data to untrusted blockchain participants, is viable and offers potential scalability, service improvements, and cost reductions compared to current centralized architectures, while also offering a decentralized currency and the possibility of running Smart Contracts on the transaction layer.

1.4 Publications and Awards

The work developed in this dissertation resulted in the following publication:

ArchiveChain: a Permissionless Blockchain based on Proof-of-Useful-Space for Archiving the Web. Bruno Cotrim, Miguel Matos, Bernardo Ferreira. Proceedings of the 16th Simpósio Nacional de Informática (INForum'25). Évora, Portugal. 2025.

This work was awarded the *Prémio Arquivo.pt 2025 – DNS.PT Honorable Mention*[2], an award recognizing the best academic contributions to the web preservation mission of Arquivo.pt.

1.5 Structure

The remainder of this work assumes the following structure:

- **Chapter 2** presents the background information about core concepts of this thesis, such as distributed ledgers and blockchains, and different types of consensus mechanisms that rely on non-forgeable resources and PoT;
- **Chapter 3** discusses the current state-of-the-art of relevant topics surrounding blockchains that use space as a non-forgeable resource;
- **Chapter 4** describes our approach for building a resource-efficient consensus via PoUSp and provide verifiable storage via PDP;
- **Chapter 5** shows implementation details that are not part of the protocol but are crucial to consider when building a permissionless blockchain;
- **Chapter 6** presents experimental results of the system evaluation;
- **Chapter 7** points to future work and concludes this thesis.

[2] `https://sobre.arquivo.pt/en/meet-the-winners-of-the-arquivo-pt-award-2025/`

Chapter 2

Background Work

In this chapter, we provide the foundational context essential for understanding our work. We begin by outlining the core principles of blockchain systems and their inherent security properties. Next, we examine various consensus mechanisms employed in permissionless blockchains, highlighting their roles in achieving decentralized agreement. We then explore PoSt, a mechanism that enables verifiable storage of useful data over time. Finally, we discuss Verifiable Delay Functions and their application in enforcing secure, provable time delays in cryptographic protocols.

2.1 Distributed Ledgers

A ledger object maintains an immutable, ordered record of transactions, ensuring the system state history is verifiable and auditable. A ledger retains all historical data, enabling transparency and traceability, and allowing the current state to be derived by traversing the entire transaction history. These transactions can encompass any type of useful information to ensure a functionality, such as the parties involved and the amount in a cryptocurrency transfer. The ledger offers two main operations: *read()*, which returns a copy of the current ledger state, and *append()*, where new transactions are appended at the end of the ledger.

In **centralized ledger** systems, a single entity maintains control over the ledger, requiring all participants to share a unified view. This centralized structure can lead to challenges in synchronization and maintenance [19]. Consequently, network participants must place trust in the central authority, which introduces ”3 Sins” to take into account: transaction forgery (commission), transaction censorship (omission), and transaction reversal (deletion) [19].

To address these challenges, **distributed ledgers** offer a robust alternative. In this architecture, the ledger is replicated, synchronized, and maintained across multiple network peers. However, this approach presents the challenge of achieving consensus on a consistent ledger state across all replicas, as peers must collectively agree on the current state to ensure the system's integrity and reliability. These properties allow distributed ledgers to be used in decentralized currencies and smart contracts, often via a blockchain, which is an implementation of a distributed ledger.

2.1.1 Game theory and consensus

Distributed ledgers must address the Byzantine Agreement problem of achieving consensus in distributed systems, requiring all honest participants to agree on a common value even in the presence of malicious or faulty nodes [53]. In the Byzantine Agreement problem a group of generals, each leading an army, must agree on whether to attack a surrounded city or retreat. The generals are physically separated from one another and can only communicate using messengers. However, both the generals and the Messengers can be traitors, meaning that they can lie or change the message about the action being executed by a general, making it difficult to reach a consensus on an action. [53, 72]. Consider the following example, where one of the honest generals decides to attack, the other honest one decides to retreat, and a third malicious one sends retreat to one of the honest generals and attacks the other. Following the majority rule, one of the honest generals will attack, while the other will retreat, causing them both to lose. Lamport et al. [53] showed that this problem can only be solved if a two-thirds majority of honest generals exist.

The Byzantine Agreement problem serves as a direct metaphor for distributed ledger consensus. Achieving consensus is not just desirable, but essential, as it enables network peers to agree on a consistent view of the system, even in the presence of Byzantine peers that may behave arbitrarily. This agreement forms the backbone of fault-tolerant blockchain systems and is fundamental to their reliability and security.

2.2 Blockchain as a distributed ledger

The blockchain was introduced by Satoshi Nakamoto as a publicly distributed ledger, serving as the backbone for the Bitcoin digital currency [36, 59]. It is composed of a list of blocks that are securely linked using cryptographic hash functions. Each block contains useful information, such as transactions, represented as a Merkle Tree[1] [56, 57], a nonce that solves a cryptographic puzzle, along with a reference to the preceding block, derived from the hash of the data contained in that block, and other important information. As an implementation of a ledger, the blockchain state is derived by traversing its immutable list of records. Once added, transactions become irreversible, and new blocks can only be appended at the end, each referencing its parent block. As discussed in Section 2.1.1, blockchain systems require peers to reach consensus on the current view of the chain. In other words, they must agree on which blocks are added to the blockchain. This agreement is essential for maintaining system stability and ensuring security against threats such as double spending (the repeated attempt to use the same coin in multiple transactions), Sybil attacks, and other vulnerabilities. Following the ledger object properties, the blockchain history cannot be tampered with, since by referencing a parent block hash, changes on earlier blocks of the chain would propagate to future blocks and be easily detected at a given height, which is crucial in fault-tolerant decentralized systems.

[1] A tree where leaves are the hash of data blocks and non-leaf nodes are the hash of their children. For example, in a 4-leaf tree, the root would be computed as $H(H(n_1, n_2), H(n_3, n_4))$

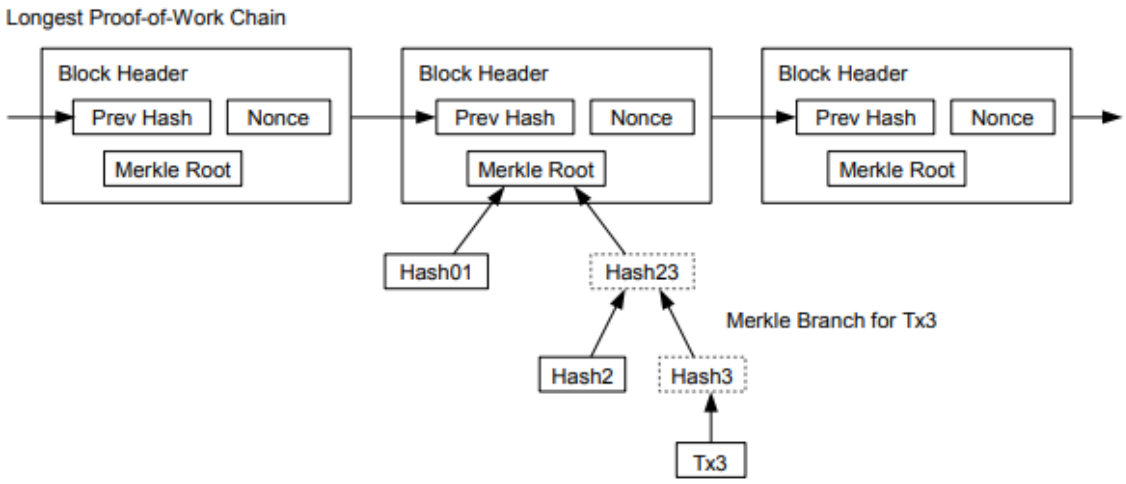

Figure 2.1: Bitcoin Blockchain Structure [59]

2.2.1 Blockchain types and security properties

In **permissioned blockchains** access to the network is restricted, this means that the identity of each participant is known and verified by a trusted party of the system. This structure enhances control and facilitates policy enforcement under a centralized governance model. In this setup, classical algorithms such as PBFT [21] are often employed, leading to more reliable and efficient systems since transactions are final when added to the system, smaller networks and the existence of a controlled environment with less exposure to unidentified access dangers, this results in simpler consensus algorithms with reduced communication overhead, making it practical for enterprise use. Quorum [26], HyperLedger Fabric [6], and Corda [18] are some examples of permissioned platforms. However, trust in a central authority is necessary, leading to weaker decentralization with less transparency and the existence of a single point of failure that limits the fault tolerance of the system.

On the other hand, **permissionless blockchains** are public networks in which no central authority governs participation, and anyone can freely join and contribute to the system. It establishes a trustless environment where participants are not required to place trust in one another, but rather in the combination of the protocol and the underlying cryptographic primitives called distributed trust. This model addresses the limitations of permissioned blockchains by enabling greater decentralization, enhancing transparency, and eliminating single points of failure. However, these advantages come with trade-offs. Transaction throughput tends to be lower, as achieving finality (the point at which transactions become irreversible) requires time for blocks to propagate across the network and the confirmation of multiple blocks. Furthermore, the absence of identity verification makes permissionless systems vulnerable to Sybil attacks, where a malicious actor can create numerous fake identities, increasing their voting power, achieving a majority quorum, and gaining control of the system. This makes it impossible to use classical consensus protocols that use a structural adversarial threshold, common in permissioned systems, instead of a computational

threshold. Preventing Sybil attacks involves implementing mechanisms that make it costly to create numerous fake identities. These mechanisms typically depend on non-forgeable, verifiable resources, such as computational power (e.g., PoW), storage capacity (e.g., PoSp), or financial stake (e.g., PoSp), which the honest majority of the system should control.

Bitcoin [59] laid the foundation for cryptocurrencies as the first digital currency based on a permissionless blockchain. Being a permissionless system, its security issues are exposed, and participants are constantly under attack, from the consensus layer to the application layer [71]. As the adoption of these technologies increases, the resulting losses from such attacks also rise. In Nakamoto's work [59], the chain progresses and remains consistent if the honest participants control the majority of CPU power. However, given the increasing attacks and losses, a more theoretical security analysis was needed to define the core properties of such a system.

Garay et al. [36] provided one of the earlier rigorous security analyses of the Bitcoin backbone protocol, and identified two main properties, that robust public transaction ledgers should follow, and states as follows:

- **Persistence with $k \in \mathbb{N}$:** Under the honest majority assumption, once a transaction is considered stable by one honest node, all other honest nodes will see the transaction in the same ledger's position and agree on the entire prefix of the ledger. A transaction is considered stable if the block that contains it is at least k blocks deep in the ledger.

- **Liveness with $u \in \mathbb{N}$:** Under the honest majority assumption, if all honest nodes in the system attempt to include a certain transaction, then after u time intervals, all honest nodes will see that transaction as stable in its blockchain.

Under the majority assumption, **persistence** and **liveness** properties can be derived from the following properties that a secure blockchain should guarantee - all with high probability [36, 46, 47, 64]:

- **Common-Prefix/Consistency with $k \in \mathbb{N}$:** At any point, the chains of two of the honest parties can only differ in K blocks. The deeper a block is the lower the probability of two chains differing considering a security parameter, i.e, mining difficulty.

- **Honest Chain Growth:** At any point, the chain of honest players grows by at least T messages in the last $\frac{T}{G}$ rounds, G being the chain growth of the system.

- **Chain Quality:** At any point, for any T consecutive blocks in an honest party chain, the fraction of honest blocks from honest parties is at least F- ideally greater than $\frac{1}{2}$- Being G and F dependent on the security parameter of the protocol, network latency and time between blocks.

2.3 Bitcoin and Proof-of-Work

PoW proves that a certain amount of computational power was expended. It was first introduced by Cynthia Dwork and Moni Naor [30] to prevent email spamming by requiring users to solve moderately hard crypto puzzles, issued and verified by Email Servers. These puzzles were based on the computationally hard yet easy to solve problem of extracting square roots $y^2 \equiv x \pmod{p}$, meaning that Spammers that sent large amounts of emails would have an increased cost of processing power thus disincentivizing email spammers.

Hashcash, initially proposed in 1997 by Adam Back and formally refined in 2002 [9, 10], was designed to mitigate email spam and denial-of-service attacks by employing cryptographic puzzles, tackling similar concepts of Dwork and Naor's [30] work. In Hashcash, users must expend computational resources to generate a token, or *nonce*, such that the hash function output $H(\textit{nonce} + \text{message})$, e.g $H = SHA - 256$, starts with n leading zeros, where n represents the computational difficulty parameter. This PoW token is embedded in the message and can be verified by the email server with minimal effort: it simply recomputes the hash and checks if the number of leading zeros matches n. Hashcash introduced key improvements over earlier work [30]. By leveraging cryptographic hash functions, the protocol became non-interactive when n was a predefined constant, eliminating the need for back-and-forth communication between participants. Furthermore, the difficulty parameter n is easily adjustable, making the system adaptable and scalable for diverse use cases. These attributes facilitated integration into existing systems, such as SMTP servers.

2.3.1 Bitcoin Protocol

Bitcoin [59], as a permissionless system, faces the Byzantine Agreement problem, making it vulnerable to Sybil attacks that challenge the secure consensus among untrusted participants. To mitigate these attacks, Bitcoin uses PoW to tie a participant's voting power to computational power, a non-forgeable resource instead of identities. This prevents Sybil Attacks by making it expensive to generate computational power, in order to increase voting power, and gain majority network control.

Bitcoin was a foundation for blockchain technologies and popularized PoW consensus in permissionless systems, most specifically Nakamoto Consensus. Nakamoto Consensus PoW is based on Adam Back's Hashcash [10] cryptopuzzle, it consists on finding a nonce n such that when block is hashed, e.g $SHA - 256$ the output of $H(p, b, n)$ starts with k leading zeros, being p the hash of the previous block, b the current block including transactions, timestamp, etc. and k the difficulty parameter. Probabilistically the work required to solve the puzzle increases exponentially with the number of zero bits and can be easily verified by executing a single hash. To compensate for hardware speed, for every 2016 blocks, this k is adjusted by the moving average of the number of blocks per hour, and a target block generation time of 10 minutes.

After solving the cryptographic puzzle, a participant (called a miner) broadcasts the newly mined block to the network. Other participants then add the block to their local blockchains.

However, it is possible for multiple miners to mine blocks simultaneously, resulting in different valid chains being observed by participants, a situation known as a fork. To resolve these conflicts and maintain consistency, Bitcoin uses a policy known as the longest chain rule. The longest chain rule complements PoW by dictating that nodes should always extend the chain with the highest cumulative computational work. When forks occur, nodes initially extend the first valid block they receive at a given height. If a longer chain later appears, nodes switch to the new longest chain, ensuring eventual convergence across the network. This mechanism achieves consensus by encouraging all nodes to adopt a consistent view of the blockchain as it grows, thus providing finality over time.

In Nakamoto Consensus, a participant's voting power is tied to their computational resources, as generating a valid block requires solving a computationally expensive PoW puzzle. This cost ensures that extending the blockchain (producing a longer chain) demands significant computational effort, making it prohibitively expensive for malicious actors to outpace the honest majority with a fraudulent chain. Nakamoto [59] explains that this combination of PoW and the longest chain rule defends against Sybil and double-spending attacks. Modifying a confirmed block requires redoing the PoW for that block while surpassing the computational work of the honest chain, which becomes increasingly infeasible as more blocks are added, assuming that the majority of computational power is controlled by honest participants who outpace any competing chain. Following Game Theory, Bitcoin achieves consensus by incentivizing honest behavior, rewarding miners with coins for mining blocks, and transaction fees for extending the longest chain, while penalizing dishonest actions with high computational and energy costs. This ensures that even a rational attacker with significant power is motivated to follow the protocol, as breaking the system would invalidate their own wealth [59].

Bitcoin Analisys

As discussed in Section 2.2.1, Garay et al. [36] outlined key properties for robust transaction ledgers but concluded that the Nakamoto assumption of a majority of honest CPU power is insufficient to ensure their security. They argue that additional assumptions are critical, namely, unforgeable digital signatures and a network synchronization rate significantly faster than the PoW solution rate, accounting for network delays. Furthermore, the emergence of mining pools (groups of miners collaborating on the same cryptographic puzzle and sharing rewards) has introduced additional security concerns.

Eyal et al. [35] highlight how mining pools challenge Nakamoto's majority assumption. By withholding newly mined blocks, mining pools can grow their private chain until it surpasses the honest chain, allowing them to claim more rewards than their fair share of computational power. This strategy is known as selfish mining. Selfish mining leads to wasteful work since blocks from the honest chain are discarded when the mining pool chain appears and honest nodes are incentivized to join these pools to avoid wasteful work and earn more rewards, according to Eyal et al. the main concern is that mining pools reduce the honest threshold from 50% to around 33% lead-

ing to centralization risks. In addition to the structural risks posed by mining pools, geographical and economic factors further amplify centralization concerns due to the concentration and migration of miners to countries with cheaper electricity, which in the case of a blackout would also impact the available computational power and the overall honest threshold [66].

Another important consideration is the blockchain trilemma, which highlights the challenge of achieving high scalability, decentralization, and security simultaneously. To prioritize security and decentralization, Bitcoin sacrifices throughput scalability, resulting in an inherent trade-off among these three metrics [73]. In Bitcoin, a block is mined approximately every 10 minutes, is deemed stable after six confirmations, and has a maximum size of 1 MB. This design leads to an average transaction confirmation time of about 60 minutes and a maximum throughput of approximately 7 transactions per second, significantly lower than centralized payment systems like Visa, which can handle peak throughputs of up to 56,000 transactions per second, but sacrificing decentralization [27].

Energy consumption is a major concern in Bitcoin mining. As of October 2024, Bitcoin's annual energy usage is estimated to be between 155 and 172 TWh, comparable to the energy consumption of countries like Poland [41]. Some argue that this energy is a necessary cost for maintaining a secure and decentralized monetary system, noting that conventional financial systems also incur significant collateral costs [15]. Critics, however, contest that this energy use provides limited societal value and highlight its environmental impact, particularly its carbon footprint. Proposed solutions include optimizing PoW protocols and adopting classical Byzantine Fault Tolerance systems. In the following sections, we focus on alternative non-forgeable resources.

2.4 Proof-of-Stake

PoS emerged as a primary alternative to PoW, initially theorized in the Bitcointalk Forums [25] in 2011, with the first operational implementation in Peercoin [48] in 2012. However, Peercoin employed a hybrid model, utilizing PoW to initialize the network and mint coins (process of generating coins), with PoW minting designed to be less competitive, thereby promoting PoS minting in the long run. The first purely PoS based system was introduced by Nxt [43]. PoS addresses the primary limitations of PoW, namely throughput scalability, and energy efficiency, by leveraging cryptocurrency as a non-forgeable resource instead of computational power to solve complex cryptographic puzzles.

In PoS, validators are selected to propose and verify new blocks through a weighted lottery, where the probability of selection is proportional to a participant's stake (voting power) represented by their collateral in the system. This mechanism enables faster block times, enhancing throughput and significantly improving energy efficiency.

From the foundational PoS system PeerCoin to modern implementations, several PoS protocols have emerged, including Cardano's Ouroboros protocol [42, 47] and Solana [75]. Notably, Ethereum [20, 34] transitioned from PoW to PoS with the launch of Ethereum 2.0 in "The merge" in September 2022, seamlessly updating its consensus mechanism without impacting token hold-

ers. Current estimates indicate Ethereum's 2.0 energy consumption is approximately 0.0026 TWh, a significant reduction compared to Bitcoin's PoW consumption of 155 TWh [34]. Additionally, it increased throughput to a range of 16 to 30 transactions per second.

However, the lack of computationally expensive operations also introduces vulnerabilities in the system [29], being the most important:

- **Costless Simulation Attack/Nothing-at-stake:** Unlike PoW systems, where mining blocks requires substantial computational effort, in PoS systems, there is no cost to validators. This costless approach enables validators to sign multiple blockchain forks simultaneously without incurring penalties or altering block contents, such as timestamps, to gain a future advantage and increase their rewards. Intuitively, a simple method to mitigate this attack is adding some cost to sign multiple chains or use checkpoints, Ethereum 2.0 [20] achieves this by burning a portion of a validator staked coins and removing him from the validator pool when caught signing or participating dishonestly in multiple forks along with the use of justified (2/3 stake votes) checkpoints, thus creating a strong economic disincentive for attackers.

- **Long-Range Attacks:** Long-range attacks occur when an adversary uses old or compromised validator keys to create an alternative blockchain history starting from a past point, potentially as far back as the genesis block. These keys, representing a high stake in the past, enable the attacker to create a seemingly legitimate fork. In Ethereum 2.0 [20], online nodes are safeguarded by the Casper Finality Gadget, which establishes finalized checkpoints through supermajority validator attestations, and the GHOST fork-choice rule, which prioritizes the chain with the most recent justified checkpoints and stake. Consequently, online nodes reject conflicting long-range forks lacking recent checkpoints. However, nodes syncing from scratch or offline for extended periods remain vulnerable. Such nodes may accept an alternative chain because they lack the knowledge about recent finalized checkpoints, exploiting the inherent "weak subjectivity" of this PoS protocol. To counter this, Ethereum 2.0 employs weak subjectivity checkpoints (trusted, periodically updated states that nodes must sync from or verify) to prevent acceptance of invalid alternative histories and ensure consensus on the canonical chain [32].

Another concern in PoS systems is the concept of wealth concentration. The more coins a participant stakes, the more likely they are to be selected as validators, consequently earning more rewards and amplifying the effect. This causes a centralization risk to security because richer holders have more power on the network and if sufficient holders can gain the majority of the stake, they can forever control the network.

2.5 Proof-of-Space

PoSp was introduced by Dziembowski et al. [31] as an energy efficient alternative to PoW that relies on storage space instead of computational power as a non-forgeable resource to achieve

consensus on a blockchain. In PoSp systems participants prove that they allocated some specific amount of data to the network. It consists of a two-phase communication protocol composed of the following:

Initialization Phase: Prover P allocates a certain amount of disk space by pre-computing and storing cryptographic data structures, often called plots [63]. This process of creating plots is computational/time consuming, but once created, they remain in disk, being a one-time cost, amortized over the system lifetime since the stored data can be reused to continuously produce proofs.

Challenge Phase: Verifier V issues challenges that require P to demonstrate the possession of the pre-computed data. When challenged, P derives the proof of the stored data.

Verification Phase: P submits the proof to V that should efficiently verify it with minimal computation.

This initial PoSp [31] construction was based on the computation of hard-to-pebble graphs. This type of graph is designed to explore time-memory tradeoffs, meaning that to recompute them, we always need a balance between time and memory/storage, and the less storage used, the more time is needed to recompute it.

In this approach, V sends a description of a hash function to P. An initialization phase follows, based on Directed Acyclic Graphs (DAG). In this phase, P computes a DAG where each vertex is labeled as $hash(\mu, i, p_1, \ldots, p_x)$, where μ is a nonce that uniquely identifies the storage commitment, i is the vertex index, and $p_1, \ldots, p_x$ are the labels of the parent vertices. The total number of vertices, denoted by n, determines the scale or size of the storage commitment. After computing all the labels, P constructs a Merkle tree [57] from these labels and sends the Merkle root as a commitment to V. During the execution phase, V selects a subset of labels to be opened, with size $k_p = \lambda \cdot \log(n) + 1$, where a larger λ increases the number of nodes opened in the proof phase, which indirectly ensures that the initialization phase produces a robust commitment. For each selected label, P constructs a Merkle proof[2] and sends it to V. Finally, V verifies each proof by easily recomputing the corresponding Merkle root and checking whether it matches the originally received commitment.

Probabilistically provers are incentivized to store all DAG labels due to the time/computation needed to recompute labels if some parent labels are missing from memory and the fast and cheap memory accesses to disk.

A main problem in PoSp is that an attacker could simply avoid storing proofs in disk and re-initialize them during the execution phase, this way swapping memory storage over time by CPU work when challenged. Dziembowski et al. [31] designed the protocol to make cheating inefficient since, to pass the verification, one needs to either dedicate N bits of memory, resultant from $\Theta(N)$ initialization steps, or recompute $\Theta(N)$ computation steps in each execution phase step. A rational participant should prefer to run a one-time CPU cost initialization phase, storing proofs on the disk at a minimal cost that is amortized over time, rather than being exposed to

[2] A Merkle proof consists of the target leaf and a sequence of sibling hashes that, when combined sequentially, allow recomputation of the root.

the CPU cost every time they are challenged, which, depending on the challenge configuration, can be exponentially hard. To ensure security against cheating (re-initialization on challenge), a PoSp system's honest nodes should possess the majority of the combined resources, including CPU and storage space [58]. Essentially PoSp relies on a time-memory tradeoff where rational participants should be incentivized to dedicate more storage rather than CPU time to efficiently produce proofs. This first approach had large proof sizes (several MB instead of a few bytes as in PoS and PoW) and proofs are interactive (the Merkle root commitment had to be sent to verifiers before being challenged), meaning in a Blockchain scenario, a participant cannot simply join the network without sending a transaction/message to the network. This means that if all nodes stop accepting these transactions, no new nodes can join the system.

The approach described above was implemented in SpaceMint [63] as a prototype of academic research nature, demonstrating that PoSp can be used to secure blockchains. However, it was never deployed in practice and remains a proof of concept.

Many simpler and efficient schemes based on inverting functions have been proposed, however, without guarantees to Hellman time-memory trade-offs, also known as Hellman attacks [38]. Hellman showed that a domain of size N can be inverted in T time when provided S bits, whenever $S \cdot T \approx N \quad$ (e.g., $S = T \approx N^{1/2}$). In other words, an attacker would only need $\sqrt{N}$ space and time to construct a proof (inversion of permutation). For functions a weaker attack takes place where $S^2 \cdot T \approx N^2 \quad$ (e.g., $S = T \approx N^{2/3}$) meaning $N^{2/3}$ space is needed. Essentially, inverting a function table involves a trade-off between space and time based on the attacker's resources. With limited time, more space is required and vice versa, meaning, in a PoS scheme, it is possible to reduce the storage needed to produce PoS proofs by increasing computational effort, simulating a larger committed storage capacity. However, Abusalah et al. [5] explore a loophole in Hellman attacks that states "For Hellman's Attack to work, the function needs to be easy to evaluate in the forward direction and for PoSp usefulness its sufficient that the function table is computable in linear/quasilinear time", given this, their construction cannot be efficiently computed in the forward direction, thus respecting Hellman time-memory tradeoffs, while also being non-interactive since no commitments need to be sent to the verifiers.

The most basic construction presented by Abusalah et al. is a function $g_f : [N] \to [N]$, which is built from a function $g : [N] \times [N] \to [N]$ and a permutation $f : [N] \to [N]$. Both g and f are modeled as truly random, and f is also modeled as a permutation for cleaner analysis, however, a random function (e.g., instantiated with a cryptographic hash like SHA-3) would suffice in practice. The function gf is defined as $g_f(x) = g(x, x')$, where $f(x) = \pi(f(x'))$ and π is an involution without fixed points (function equal to its inverse), such as a bit-flipping operation denoted $f(x) = \overline{f(x')}$. In case f is a function, π is not needed and f can simply be defined as $f(x) = f(x')$. In the initialization phase, the verifier sends the definition of $g_f : [N] \to [N]$ to the prover, which will then compute the entire function table for f along with the corresponding table g_f, in other words, the prover finds pairs (x, x') such that $f(x) = f(x')$ and the corresponding $g_f(x) = g(x, x')$ obtained from the pair. During the execution phase, the verifier sends a

random challenge $y \in [N]$, and the prover responds with a tuple (x, x') such that $f(x) = f(x')$ and $g(x, x') = y$. Then, the verifier efficiently checks these conditions. The search for pairs is computationally intensive during initialization, but once calculated, these pairs and their corresponding g_f values can be stored on disk, deterring Hellman attacks by imposing a significant storage commitment and discouraging repeated computationally expensive initialization costs.

Because PoSp farmers/miners are not required to expend significant computational effort to generate blocks, as creating this type of proof only requires simple disk lookups and reads. Consequently, it is vulnerable to long-range attacks and costless simulation [23], as discussed in Section 2.4.

2.6 Proof-of-Storage

PoSp schemes, like the one used in Chia Blockchain [23], often rely on random data as a non-forgeable resource to demonstrate storage commitment. While effective for ensuring decentralized consensus, the allocated storage is underutilized, as the random data serves no purpose beyond the consensus protocol. Proof-of-Storage (PoSt) provides a way of storing meaningful data while ensuring it can be cryptographically verified. While PoSt itself does not focus on consensus, it can complement PoSp by enabling the storage of meaningful data within the space that PoSp would otherwise leave unused.

PoSt schemes provide cryptographic guarantees that enable a verifier v to outsource the storage of specific data D (e.g. Files) to a provider P. Unlike PoSp, which uses random data, PoSt ensures that P maintains and correctly stores the meaningful data D, while allowing V to periodically verify P's commitment to storing D ensuring data integrity, retrievability, and accountability. To note that PoSpT only guarantees P is storing D **at the time of the challenge**, so periodical proofs are needed to guarantee D was stored throughout a period in time.

PoSt can also be used independently from consensus protocols, thus the use in Cloud Services [13] and L2 (built on top of Layer 1 networks usually via transaction) Blockchain implementations to provide storage guarantees, e.g Storacha [68] and SiaCoin [69]. Three approaches have been proposed:

Proof-of-Data Possession (PDP): PDP [7, 70] is a cryptographic protocol that allows a client to verify that a file stored on an untrusted server remains intact and unaltered without downloading the entire file. The client pre-processes the file, locally stores cryptographic metadata, uploads it to the server, and then safely deletes the local copy. Periodically, the client issues challenges to the server, which must compute proofs and send them back. The client verifies these proofs using the locally stored metadata. While PDP ensures the integrity of the file on the server, it does not address the potential corruption of individual file blocks.

Proof-of-Retrievability (POR): POR [45, 67, 70] extends the idea of PDP by verifying the integrity of the data but also ensuring that the entire file can be retrieved from the server even in partial corruption of data blocks. POR achieves this by using techniques such as erasure codes and aims to detect and possibly correct them if the corruption is under a certain threshold. The

process works similarly to PDP but also adds cost of pre-processing erasure codes to add that extra functionalities.

Proof-of-Replication: PoRep [13] is a protocol that proves data D has been replicated to its own uniquely dedicated physical storage. By enforcing unique physical copies, PoRep allows a verifier to confirm that a prover is not deduplicating multiple copies of D in the same storage space.

When clients replicate data across multiple storage providers, the above ideas are often combined with client side encryption. When the client is pre-processing the file, it should generate a unique encoding using a SecretKey Sk that should be unreproducible by any other entity, this way making it impossible for stores to share the same disk space (e.g. Replay Attacks) since each of them is associated to a unique encoding of the replicated file, apart from this pre-processing step, challenge and verification phase are specific to the constructed protocol [13, 69].

2.7 Proof of Time

PoT demonstrates that a function has been executed sequentially T times/iterations, with sequential execution being its critical feature. This implies that the function is inherently non-parallelizable, meaning that the prover cannot accelerate execution by deploying additional computational resources [61]. PoT is implemented through VDFs [65, 74], which are deterministic functions designed to require a quantifiable amount of real (wall-clock) time for computation [16], derived from the parameter T. These functions produce a unique output accompanied by a compact proof that enables efficient public verification. These designs can be useful to mitigate risks such as costless simulation and long-range attacks, as well as making functions hard to evaluate in the forward direction, both of which will be explored in our work PoUSp consensus scheme.

2.7.1 Wesolowski Verifiable Delay Function Scheme

Wesolowski's [74] VDF is based on squaring in a group of unknown order $G \in \mathbb{Z}_N^*$, where $N = p \cdot q$, being p and q the product of two distinct prime numbers and the size of N dependent on a security parameter K. The scheme is specified by the following two algorithms:

- **VDF.solve(*c,t*)** $\rightarrow \tau = (y, \pi, c, t)$

  Being the input c a challenge $c \in \{0,1\}^n$, and t time parameter/iterations. The output τ consists on a tuple containing the input (for convenience) along with an output of the function y, and a proof π that y has been correctly computed. This output can be computed given the equation $y = c^{2^t} \mod N$, this can be calculated by squaring c sequentially t times, e.g. $c \rightarrow c^2 \rightarrow c^{2^2} \rightarrow \cdots \rightarrow c^{2^t}$, an important aspect is that there is no shortcut to this computation without knowing the factorization $N = p \cdot q$, thus taking T exponentiations.

- **VDF.verify**$(\tau) \in \{\text{reject}, \text{accept}\}$. This method returns either `true` or `false` based on whether it successfully verifies the output. It ensures that the output was indeed calculated

with the specified delay T in the group G. The verification process involves verifying $y \equiv \pi^q \cdot c^r \mod N$, where $q = H_{\text{prime}}(c \parallel y)$ and $r = 2^t \mod q$, which is much more efficient than the computation, requiring only $\log T$ multiplications.

Along with these two specified algorithms, Wesolowski [74] and Bonet et al. [16] specify three properties a VDF should satisfy:

1. **Uniqueness:** It is hard to create a false output y and still have a valid proof π. Meaning, for the same input, the output y will always be unique but the proof π can change.

2. **Sequentialitiy:** An attacker that makes less than T sequential steps will not find an accepting proof on c. The number of sequential steps bounds the attacker and, even using high parallelism, the VDF output cannot be computed faster than the speed of a single CPU core computing a step of the VDF computation.

3. **Efficiently verifiable:** To ensure practicality for honest parties, the verification procedure is designed to be highly efficient, with a total runtime bounded by $O(\text{polylog}(t))$.

An important aspect of this scheme is that being a trapdoor VDF, using an RSA group as above, the security depends on the secrecy of the factorization $N = p \cdot q$, if this factorization is revealed, a shortcut to the computation can take place, breaking the scheme. To deal with this, imaginary quadratic fields as the group of unknown order can be used, which makes it possible to sample a group without revealing its order given a random number [17, 74].

2.7.2 Slow-Timed Hash Function

A Slow-Timed Hash Function (SLOTH), as the name indicates, was proposed by Arjen Lenstra and Benjamin Wesolowski [55] to be a hash function where its computation is designed to be as slow as desired, following two design criteria:

- It must be possible to choose parameters, such that, computing SLOTH takes at least ω seconds, independently of the computing resources available.

- The hash verification process should be modest compared to the SLOTH computation time ω.

This scheme is based on modular square roots. It begins with the selection of a prime number p such that $p \equiv 3 \mod 4$ and $p \geq 2^{2k}$, where k is a security parameter related to the bit length of p, and an input x.

The SLOTH computation is centered on modular square roots, specifically using the transformation $y = x^{\frac{p+1}{4}} \mod p$ along with an invertible permutation σ to ensure that the output cannot be expressed as a simple algebraic function of the input. This construction enforces sequential evaluation of each modular square root, thereby providing time-delay security.

Verification, by contrast, relies on modular squaring, which is significantly less computationally intensive than modular exponentiation. The basic verification step is given by $x = y^2 \mod p$, followed by the application of the inverse permutation σ^{-1}, where, at the end, the computed x' should match the input x. Both the computation and verification procedures consist of T iterations, where T is a configurable time parameter that can be calibrated to match a desired wall-clock delay.

The forward (computation) and reverse (verification) processes are defined as follows:

SLOTH.Computation (Forward Iteration):

$$w_i = \rho(\sigma(w_{i-1})), \quad \text{for } i = 1, \ldots, T$$

where:

$$\sigma(x) = \begin{cases} \sqrt{x} & \text{if } x \text{ is a quadratic residue,} \\ \sqrt{-x} & \text{otherwise} \end{cases}$$

$$\rho(x) = \begin{cases} x^{\frac{p+1}{4}} \mod p & \text{if } x^{\frac{p+1}{4}} \bmod 2 = 0, \\ p - x^{\frac{p+1}{4}} \mod p & \text{otherwise} \end{cases}$$

SLOTH.Verification (Reverse Iteration):

$$w_{i-1} = \sigma^{-1}(\rho^{-1}(w_i)), \quad \text{for } i = T, \ldots, 1$$

where:

$$\rho^{-1}(y) = y^2 \mod p$$

$$\sigma^{-1}(y) = \begin{cases} y & \text{if } y \bmod 2 = 0, \\ p - y \mod p & \text{if } y \bmod 2 = 1 \end{cases}$$

An important observation is that the scheme is not asymptotically efficiently verifiable. The total computation time is $O(T \cdot \log p \cdot M(p))$, while verification takes $O(\log p \cdot M(p))$, where $M(p)$ denotes the cost of modular multiplication for modulus p. Thus, verification is only a factor of T faster than computation, meaning that verification time still grows with the size of p and T, rather than achieving a polynomial or greater speedup [74]. However, in our application, we target short time delays of about 1 to 2 seconds, making the verification cost negligible in practice. Since this scheme is not a trapdoor function and requires no trusted setup, a verifiable slow hash function like SLOTH is ideally suited to our construction. SLOTH can also be adapted as a verifiable delay encoder (VDE), enabling data to be encoded in a deliberately time-consuming manner, while allowing for rapid decoding using the corresponding verifiable delay decoder (VDD). This approach, exemplified in the Subspace Network Blockchain SLOTH implementation [8]. Such a mechanism shifts from a hash function to a encoder/decoder that can be particularly useful in our time-based PDP encoding protocol.

Chapter 3

Related Work

In this chapter, we will explore key decentralized systems and blockchains. We will start with the Chia blockchain and its PoSp and PoT alternation mechanisms, including its security against attacks such as long-range, grinding, and double-dipping. We will also examine the Chia Bread Pudding protocol [44], which improves storage efficiency by embedding useful data in proofs. In addition, we will review Filecoin's [51] decentralized storage network, which uses PoRep and PoSpT. This analysis will highlight their contributions, limitations, and relevance to our proposed PoUSp scheme. Each analysis will be detailed, as our work shares many of the same challenges as the following studies due to similar objectives and nature. This comparison will help guide the development of our solution.

3.1 Chia Blockchain

Chia [23] is a cryptocurrency implemented on a blockchain system that uses a combination of PoSp and PoT for its consensus mechanism. It aimed mainly to enhance PoW sustainability problem, by using disk space as non-forgeable resource instead of computational power, and the interactivity problems of previous systems (e.g., SpaceMint [63]) based on the Dziembowski et. al. [31] PoSp scheme, where provers need to first send a commitment as a blockchain transaction before being able to join the network, meaning they cannot simply participate by listening to the network and if miners do not accept the transaction, one might never be able to participate. To solve this, Chia PoSp is based on the Abusalah et al. [5] approach, which is both proof-size efficient and non-interactive.

Chia's PoSp [24, 60] implementation nests 6 times (7 tables) the Abusalah et al. [5] structure design, which increases exponentially the computational difficulty of solving the problem, in other words, it makes the scheme exponentially more resistant to Hellman Attacks [38]. This construction also has different heuristics to make it more practical (e.g., a Matching Function M, a collation Function C, etc.). Each table has 2^K entries, being K a security parameter with a minimum value of 32, that generates a table with around 101.4 GiB. Each entry in a table points to entries in the previous table, except for the first table, which contains a pair of integers called "x-values."A PoSp consists of 64 K-bit long x-values that satisfy a specific mathematical relation-

ship. This approach significantly reduces the proof size compared to earlier schemes, shrinking it from several megabytes to approximately 256 bytes when $K = 32$. The computational complexity of generating this structure incentivizes honest participants to store proofs in disk instead of computing them on the fly, since probabilistically they could not produce them constantly, either fast enough or without using an immense amount of resources while honest participants have a one-time cost of generating proofs and an extremely small cost of storing and looking for them on disk over time.

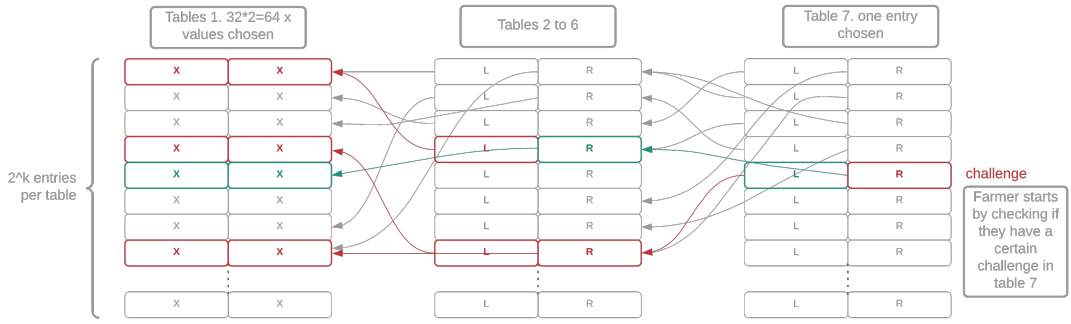

Figure 3.1: Structure of a plot file. The 64 red x-values represent the proof, the 2 green x-values represent the quality. [40]

Chia's PoT [24] is based on the Wesolowski Scheme [74], utilizing repeated squaring in classgroups of unknown order, as mentioned in Section 2.7.1. Unlike RSA-based groups, classgroups with large prime discriminants eliminate the need for trusted setup (e.g., via Multi-party Computation) but are less optimized and tested in practice. Chia's PoT generates 1024-bit prime discriminants and proofs derived from blockchain randomness. A critical feature of this PoT is its sequential nature, prohibiting parallelization and tying computation speed to a single CPU core, thus limiting the advantage to current hardware, which is limited.

3.1.1 Chia's Consensus Algorithm

There are 2 parties involved in Chia consensus algorithm: a farmer is a participant who contributes with disk space to secure the blockchain and produces PoSp proofs, and Timelords are responsible for creating PoT proofs and ensuring that a real-time delay between blocks occurs. The combination of both will be crucial to maintaining the security of the blockchain.

The **Farmer algorithm** operates in two phases: *plotting* and *farming*. In the plotting phase, storage is allocated to create reusable PoSp plots. During farming, the farmer responds to network challenges (derived from the latest PoT output) by efficiently searching his disk for the highest-quality proof based on metrics like bitwise difference or closest value. Upon finding the best PoSp, the farmer produces and broadcasts a non-finalized block. If this block has the best quality, it extends the blockchain, earning the farmer rewards and transaction fees. This process helps defend the network against Sybil attacks.

The **Timelord algorithm** finalizes non-finalized blocks by computing a PoT VDF, using the PoSp present in those blocks as input. This introduces a sequential delay that secures the chain and prevents Costless Simulation and Long-Range attacks. Timelords operate on the chain with the highest accumulated space, computing PoT for signage points and infusion of blocks when their challenge point is reached. They run three parallel VDF chains, requiring at least three fast CPU cores to advance the chain at an efficient rate, while the rest of the cores perform other tasks but don not need to be as fast. Challenges or blocks with insufficient weight or outdated challenge points are ignored to prevent withholding attacks. The Timelord's main job is to cache future blocks for infusion and broadcast finalized blocks and challenge points to the network as they are processed, farmers receive them and the above processes repeat. In each slot (approximately 10 minutes), Timelords produce 64 sub-slots of PoT challenges, achieving a block time of 19 seconds and a confirmation time of 10 minutes. This enables faster transaction processing compared to Bitcoin, where only one block is confirmed every 10 minutes. Timelords receive no rewards, as their main incentive is decentralization and contributing to a safe network evolution.

In Chia, farmers and Timelords work together to extend the chain with the highest cumulative space, as determined by the quality of PoSp. Each non-finalized block has its PoSp quality evaluated, and the time required to compute the VDF is inversely proportional to this quality. This means that higher-quality blocks are extended faster by Timelords because they require less time for VDF computation. To maximize their chances of winning a block, farmers are incentivized not only to allocate more space but also to disseminate their proofs quickly. The process relies on a combination of selecting blocks with the highest quality and ensuring they are broadcast efficiently to extend the honest chain with little delay. This dynamic helps mitigate withholding attacks, as timely dissemination ensures that only the best-quality proofs are used to finalize blocks. The probability of winning a block is directly proportional to the amount of space allocated by a farmer. Allocating more space allows the farmer to compute more proofs, increasing the likelihood of finding a proof with a quality value close to the challenge.

3.1.2 Chia's Relevant Attacks and Analisys

Long-range Attacks: As discussed in Section 2.5, the security of Bitcoin relies on an attacker requiring computational power exceeding the historical average to grow a longer chain, making such attacks impractical. In contrast, Chia's PoSp can be computed rapidly, potentially allowing an attacker with sufficient space to bootstrap a heavier chain in a short time. To address this vulnerability, Chia alternates PoSp with PoT. When a farm generates a PoSp, it serves as input to the timelord's PoT computation. The output of this PoT is then used as input for the next PoSp, and the process repeats itself. This mechanism introduces an enforced real-time delay between blocks, similar to Bitcoin's mining process. The attacker's chain growth is bound to the single-core hardware speed, which, even with the fastest hardware available today, is a relatively small advantage. This bound delay/growth rate ensures that to outpace the honest network, an attacker would need to expend real-time between blocks, which effectively limits their ability to grow

a heavier chain unless they possess both significant storage space and faster timelord hardware, which is highly impractical.

Grinding Attacks: In Bitcoin, a miner can work towards solving two crypto-puzzles at the same time (by tweaking the block payload), however, this implies that they would need to split the computational resources, offering no advantage. In contrast, Chia PoSp allows malicious miners to generate multiple challenges at no cost (e.g., by tweaking a block timestamp), increasing their success rate. This leads to grinding/digging attacks, where the attacker picks the challenge that increases its odds of winning a block and/or future blocks, allowing him to deviate from honest farming. Inspired by SpaceMint [63], Chia splits the chain into a trunk chain, which contains the PoSp and PoT proofs, and the other chain is called foliage, which contains a block payload and a signature that binds both chains. This way, all the challenges come from previous trunk values, protecting against grinding attacks, since trying multiple values in the foliage changes nothing in the trunk and the odds of winning blocks.

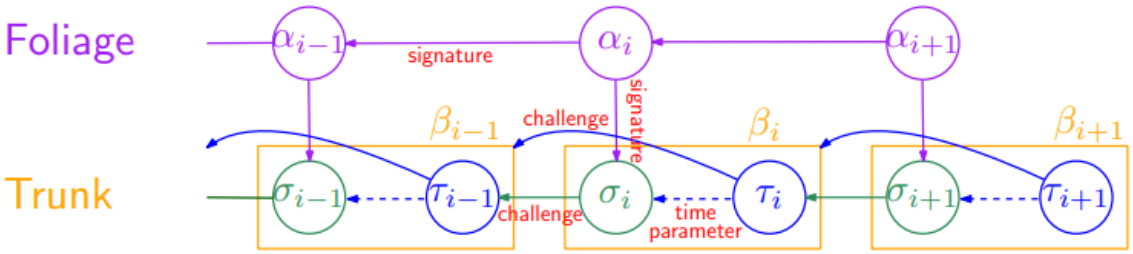

Figure 3.2: Overall View of from Chia Blockchain [23]. It contains blocks $\beta = (PoSp\,\sigma, PoT\,\tau)$ in the (ungrindable) trunk chain, and blocks α containing payload, both decoupled but linked by signatures.

Double Dipping Attacks: In double dipping, instead of trying to extend some block in many ways, an attacker tries to extend multiple blocks, posing the same dangers as the previous attack. Penalties have been suggested by Spacemint [63] to address this issue, Chia states that this is not a solution, only a deterrent. Instead of completely mitigating this, Chia makes this part of the protocol. Farmers and Timelords work on extending the first K blocks at every depth. Cohen et. al. [23] analysis concluded that setting this security parameter to 1 would require the 73% of the network to be honest, which is high compared to bitcoin 50%. Increasing k to 3 allows increasing this honest threshold to $\approx 61.5\%$, at the cost of more effort from participants and increased time to achieve consensus.

One of the main drawbacks of Chia is the weaker assumption of $\approx 61.5\%$ honest participants when compared to Bitcoin, this concern is amplified by the nature of the VDF and PoSp combination. VDFs and classgroups of unknown order with large prime discriminants d are not well tested or optimized in the real world and need further investigation. Combining PoSp with PoT to achieve consensus makes an attacker with a faster VDF "multiply" its space [24].

Chia aims to be a more sustainable solution than Bitcoin, achieving 0.13 TWh energy use,

$\approx 99.9\%$ of Bitcoins at the time of this measurement. However, energy efficiency is not the only requirement of sustainability. Space also poses no other use than to store proofs required for network security, by analyzing Chia dashboard [39], we can notice 18.7 EiB (21 Million Tb) are being used to farm plots. This lack of data utility leads to under-utilization/inefficient usage of Disk Space.

The work in the next Section aim to improve this storage inefficiency by allowing the storage of useful data within the cryptographic proofs, allowing for more general purpose and more storage efficient systems.

3.2 Chia Bread Pudding

Chia Bread Pudding was introduced by Mónica Jin [44], aiming to reduce disk waste resulting from the PoSp schemes such as the one used in Chia [23], by allowing the usage of pre-existing data (e.g., user files) within the proof mechanism. In this way, storing PoSp proofs and useful data simultaneously provides better utilization of space resources.

Being an improved adaptation of the Chia blockchain, it maintains some of its main properties. In this protocol, farmers undergo an initialization process to generate proofs before participating in the system. Challenge generation works similarly to Chia in the sense that challenges are derived from the previous block thus benefiting from the blockchain randomness and ensuring precedence relations. While adapting Chia PoSp, Costless Simulation attacks, and Long-Range attacks were also a concern of this work due to the cheap process of proof initialization.

The protocol remained non-interactive, meaning farmers can use their local data for the initialization step and are not required to send a commitment to the blockchain, being able to simply join the network by listening to the system.

To determine the winner, the protocol specifies a quality function for each proof that reflects the amount of space a farmer is allocating to the network, meaning the probability of a farmer extending a chain is proportional to the amount of space allocated. Similarly to Chia, farmers follow the chain with the most accumulated space/quality, thus, the block with the highest quality at a certain depth is chosen as an extension of the chain.

The costless simulation attack approach from Chia was not modified, the proofs of a block are decoupled from the payload into two separate chains.

The primary distinction of this protocol from Chia's PoSp lies in its approach to Long-Range Attacks and the underlying PoSp proof scheme. To mitigate long-range attacks, rather than alternating between Wesolowski's PoT and PoSp as Chia does, this protocol leverages VDE/VDD to compute PoT, introducing enforced delay between blocks and thereby making block generation time-consuming. The PoT is alternated with Chia's Bread Pudding PoRep adaptation. Unlike Chia, which uses Abusalah et al. [5] nested tables, this protocol employs Merkle Trees for PoRep generation, with data initially encoded via VDE. This increases the computational cost of the initial setup, deters Hellman Attacks, and enables the embedding of useful data within proofs.

The modified PoRep algorithm is presented below:

Initialization: The first step of this phase is the encryption of the user data using Cypher Block Chaining (CBC), which helps guarantee the user's data privacy and increases the entropy of the proofs. This protocol uses a Vector Commitment (VC) Scheme, in particular, a Merkle Tree [56] to prove individual elements of a vector are part of it without revealing the whole vector, and to build such a proof, the prover needs to correctly store all elements of the vector. A Merkle Tree is built using the encrypted blocks, where the Hash of these encrypted blocks becomes the tree leaves, and iteratively, non-leaf nodes are the hash of their children, producing a Merkle Root. To finish this phase every Merkle node is encoded using a VDE. Being a VDF, the VDE encoding computation is sequential and the encoding process is time-consuming, while decoding is fast. By making the initialization step time-consuming, honest farmers are incentivized to store proofs on disk and reutilize them, having a one-time generation cost, instead of running the re-initialization during the challenge phase, making it impractical to perform a time-memory tradeoff to generate proofs on the go.

Proof Generation: In this phase, farmers receive challenges from the network (e.g., derived from the proof of the previous block). Upon receiving a PoRep challenge (derived from the previous block PoT), farmers search their disk for the Merkle root with the highest quality relative to the challenge. Once the best Merkle tree is identified, the farmer generates a cryptographic proof (e.g., $proof = E3||E4||E12||E5678||E12345678$) composed by a leaf ($E3$) and the corresponding Merkle Path, which includes is a set of intermediate nodes needed to compute the tree's root hash. This path can only be efficiently constructed if all leaf nodes are stored on disk, proving that the farmer is genuinely storing the data. However, a malicious farmer might attempt to store only a single path, falsely claiming to store all leaf nodes. To counter this, the protocol makes use of the blockchain randomness to randomly target a specific leaf, determined by $challenge \mod total\_leaves$. This ensures that different portions of the Merkle trees are probabilistically selected over time, making it impractical for attackers to discard nodes and incentivizing honest storage. Once the proof is generated, the farmer extends the blockchain. This proof generation process is efficient for honest miners, as proofs can be reused over the blockchain's lifespan, effectively amortizing the initialization cost.

Proof Verification: This phase mainly consists of 2 steps, decoding and validation. First, if a block's proof has a quality below a certain difficulty threshold, the block is discarded. If the block passes this filter, it enters the decoding phase where a VDD applies the inverse operation of a VDE and decodes the different nodes in the Merkle Tree, obtaining the original Merkle Path that corresponds to the proof's Merkle Root. Finally, in the validation phase, the Merkle Root resultant from iterative hashing of these decoded nodes matches the Merkle Root present in the proof. If it does, the block is deemed valid. To complete this phase, the challenge from the block should belong to the three most recent PoT proofs, and miners are given a short time frame (9.375 to 28.125 seconds) to generate and broadcast a valid proof for each challenge, thus adhering to real-time delays as present in Chia, preventing long-range attacks. This limited time window also discourages miners from generating data on demand, making this process expensive, thus

incentivizing the initialization and reuse of proofs on disk.

The main part of this protocol is that the user's data encoded within the Merkle Trees can simply decode and traverse all Merkle Leaves and reconstruct the original data, thus allowing for a useful storage scheme. User's data modification is also possible, however, it is required to run the initialization phase over that data again.

There are several parameters to be adjusted, n number of leaves of the Merkle tree, f the fanout of the Merkle tree, m node size, and t VDE execution time, all affecting proof sizes, useful space ratio and initialization time.

This work successfully shows that it's possible to achieve 50% of useful data while maintaining most of the security properties present in Chia, thus helping reduce storage waste.

One of the main limitations identified by the author is the absence of a proper VDE/VDD. Our proposed PoUSp scheme aims to achieve a Chia Bread Pudding [44] inspired practical implementation, addressing important security considerations and tailoring implementation aspects to suit our specific requirements. Furthermore, our analysis reveals that the useful space ratio can be improved beyond the original 50% by minimizing the amount of useless data to the size of a single VDE proof, rather than storing the entire encoded tree, further reducing the storage waste. While Chia Bread Pudding was originally designed to embed useful local user data within cryptographic proofs, it lacks built-in mechanisms for storage and integrity verification, features essential for its application in decentralized storage systems. Nevertheless, it serves as a foundational theoretical foundation for our approach, and all improvements and modifications will be addressed throughout this work.

3.3 Filecoin Blockchain

Filecoin [51] is a decentralized storage network that turns traditional cloud storage into an algorithmic market. Built on top of the Interplanetary File System [12], it enables participants to rent out unused space, and clients can choose storage providers based on cost, redundancy, and location, making it highly flexible and user-oriented. This market operates on a blockchain with a native token, Filecoin (FIL), which incentivizes storage providers to participate and maintain a quality service actively. Filecoin is essentially an economic layer on top of InterPlanetary File System (IPFS), a perfect infrastructure for decentralizing data through content addressing and implementing smart contracts.

There are three main participants in the Filecoin Decentralized Storage Network. *Clients* pay to store and retrieve data. *Storage miners* provide storage by pledging collateral proportional to the amount of storage they offer. To utilize this storage, they respond to *PUT* requests by committing to store client data for a specified duration. They must continuously generate and submit PoSpT to the blockchain to prove that they are fulfilling their storage commitments. Failure to submit valid proofs results in penalties, including the loss of part of their collateral. Storage miners are also responsible for block production, earning block rewards, and transaction fees. There are also *retrieval miners* that serve as intermediaries, assisting with the initial storage setup and responding

to *GET* requests by delivering requested data to clients. In many cases, storage miners also serve as retrieval miners, as their roles often overlap in practice.

Filecoin supports two types of verifiable exchange markets: the Storage Market and the Retrieval Market. The Storage Market operates on-chain, where clients request data storage and storage providers offer their services. The Retrieval Market functions off-chain, where clients request specific data pieces, which retrieval miners then fetch and deliver.

The Filecoin consensus relies on two types of proofs: PoRep, mentioned in Section 2.6, and PoSpT, where miners publicly demonstrate that a specific data encoding is continuously stored over time.

PoRep is a protocol in which a miner allocates storage space (typically in 32 or 64 GiB sectors) and negotiates a storage deal with a client through the Filecoin Storage Market. Once a deal is established, the client's data is stored in a miner's sector, initiating the PoRep lifecycle. This sector then undergoes a time-consuming, sequential sealing process, which encodes the original data into a unique replica D. To prove the existence of this replica without revealing its content, the miner generates a cryptographic proof. This proof is compressed using Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARK) [11, 14, 37], which allows the validity of the proof to be verified while preserving privacy and minimizing proof size. The final proof is submitted to the Filecoin network and serves as a storage commitment that attests the miner has correctly generated and stored the unique replica D.

Each resulting **sealed sector** is uniquely bound to a specific miner, ensuring that even identical unsealed data results in distinct sealed sectors. Sealing is computationally intensive and deliberately slower than proof generation and verification, taking approximately 4 to 24 hours, depending on sector size (e.g., 32 GiB or 64 GiB) and hardware capabilities. This prevents malicious miners from pretending to have two replicas of a sector without dedicating twice the size (faking storage), or two malicious miners pretending to store the same sector and only one stores the copy (collusion), meaning that only the miner that produced a replica can use it to produce its own proofs. The costly and time-consuming nature of sealing ensures that only the miner who generated and stores a sealed sector can produce valid proofs of storage within the 30-minute challenge window or block epoch, as resealing from unsealed data is infeasible in this time frame. This prevents miners from faking storage of a large number of sectors or reproducing the sealed replica from the unsealed data, either present in the miner's disk or in another farmer's. Consequently, miners are incentivized to maintain the proper storage of each sealed sector to avoid penalties.

To address the high communication complexity of repeated PoRep challenges to verify storage over time, Filecoin introduces **PoSpT**. In this mechanism, provers recursively produce a sequence of PoRep proofs over a fixed number of iterations t. This creates a composition of compact, chained PoRep proofs, significantly reducing communication overhead by minimizing data exchanged between prover and verifier while ensuring data integrity and availability over time.

Filecoin employs two types of PoSpT, **WinningPoSpT** and **WindowPoSpT**:

WinningPoSpT is related to the consensus process, in which elected miners, at specific

epochs, prove that they are storing designated sectors to mine blocks and earn rewards, including transaction fees. This is done by submitting 66 Merkle Paths for leaves of a specific sector, proving a miner has a replica of the data at the specific time when they were asked. The answer to the WinningPoSt challenge must be submitted within the short, elected block epoch, making it impossible for the miner to seal and find the answer to the challenge on demand. This guarantees that at the time of the challenge, the miner maintains a copy of the data. Missing a WinningPoSpT window incurs no penalties but results in forfeited rewards.

In contrast, **WindowPoSpT** ensures all partitions are proven to be stored at least once every 24 hours across all miners, proving that a copy of the data has been continuously maintained over time. The 24-hour proving period is divided into 48 non-overlapping 30-minute windows, each corresponding to a deadline for proving partitions (subsets of up to 2,349 sectors per partition for 32 GiB). The more storage a miner pledges, the more partitions or sectors they must prove. Within each 30-minute window, miners generate a PoSpT proof, leveraging the recursive PoRep process with t iterations (currently $t = 10$ recursive challenges) to produce a single, compressed zk-SNARK proof for the entire partition. This probabilistically proves that the miner is storing the sector over the 30-minute window. The first challenge is generated using randomness from a beacon available 20 epochs ($\sim$10 minutes) before the window opens. Each subsequent PoRep output serves as the challenge for the next iteration over t iterations. Each proof must be submitted to the Filecoin blockchain as a message before the 30-minute deadline expires. It is economically irrational for a miner to discard sealed copies of data, as re-sealing a sector for each WindowPoSt challenge is significantly more resource-intensive than maintaining it continuously. Failure to submit a valid proof within the deadline results in penalties, including loss of collateral and reduced storage power, ensuring accountability for storage commitments.

Filecoin uses a useful work consensus called **Expected Consensus**, and instead of wasteful PoW, the work that miners do generating PoSpT is what allows them to participate in the consensus, meaning that the outcome of the computation is valuable to the network (proving sector storage), beyond securing the blockchain.

In this protocol, the probability of a miner being elected to create a new block is proportional to the miner's storage in use in relation to the rest of the network. The protocol is designed such that miners would rather invest in storage than in computing power to parallelize the mining computation, since miners offer storage and reuse the proof that data is being stored to participate in the consensus.

The Filecoin consensus protocol extends existing PoS protocols, which can evolve with their future improvements as well, but replacing the stake with the assigned miner's storage. That is, the voting power is not tied to a cryptocurrency stake in the system but instead to the amount of sectors currently being stored by the miner. The power of a miner is publicly visible because all storage assignments are recorded on the blockchain, cryptographically verifiable through mandatory PoSpT that prove that the miner is genuinely storing the data they claim, and variable over time, since miners can increase or reduce their power by adding or removing storage sectors as

they participate in the network

Expected Consensus, is a probabilistic consensus protocol designed to deterministically, unpredictably, and secretly elect a small set of leaders in each epoch. The expected number of leaders per epoch is approximately one. However, due to the probabilistic nature of the election process, there may be zero or multiple leaders. If no leader is elected by the end of an epoch, an empty block is added to the blockchain. When multiple leaders are elected, their proposed blocks are grouped into a structure called a tipset. The blockchain is organized as a Directed Acyclic Graph, which allows multiple forks to occur and potentially delays convergence. Miners follow the heaviest chain, defined as the chain with the highest accumulated storage power. As a probabilistic consensus mechanism, the protocol becomes increasingly stable over time. As more blocks are added, the probability of earlier blocks being final increases, thus strengthening their confirmation guarantees.

A miner M_i is a leader at epoch t if the following condition is met:

$$\frac{H\left(\langle t \| \mathrm{SIG}_{M_i}(H(t \| \mathrm{rand}(t)))\rangle\right)}{2^L} \leq \frac{p_i^t}{\sum_j p_j^t}$$

where $\mathrm{rand}(t)$ is a public randomness available that can be extracted from the blockchain at epoch t, $\mathrm{SIG}_{M_i}(\cdot)$ denotes the signature by miner M_i, H is a secure cryptographic hash function with output size L, and p_i^t is the power of miner M_i at time t.

Questions have been raised about the complexity of PoRep and PoSpT, as well as the expected consensus protocols in Filecoin; however, other limitations include its slow data retrieval process. This is due to the time-consuming sealing and unsealing procedures, that can take up to 5–10 minutes to retrieve a 1 MiB file [52]. As a result, Filecoin is better suited for cold storage scenarios where data retrieval is infrequent or does not require high speed. The complex sealing process also requires more powerful and expensive hardware. This high initial cost deters casual/small miners from joining the network, which reduces decentralization and causes storage power to concentrate in a few powerful identities (e.g., mining pools), potentially harming network security and allowing for collusion. This computationally expensive sealing process also deters miners from adding storage capacity without proportionally increasing computing, further increasing costs and limiting storage and network scalability.

3.4 Discussion

In this section, we will evaluate the previously presented blockchains and protocols in Sections 2 and 3. Most importantly, we will compare the multiple approaches with their best advantages and drawbacks, as well as how their ideas can inspire our work. Table 3.1 shows an overview of the comparison between all analyzed systems.

Bitcoin's PoW has proven to be a foundational mechanism in Sybil Attack prevention, tying voting power to computational power, mainly due to it being the first to appear and demonstrating long-term reliability and robustness, making it one of the most secure systems to remain successful

since its creation. This system is trustless and completely decentralized, and in theory, anyone with computational resources can join. However, as the system scales, the mining difficulty increases, requiring more computational power to solve cryptographic puzzles. This increases hardware complexity (e.g, ASICs) and infrastructure costs, creating significant barriers to entry for new participants. Most importantly, the network's energy usage also scales with the total computational power, reaching levels comparable to those of small countries. This raises serious concerns about sustainability and environmental impact. Energy needs also increase the concentration of participants in cheap electricity zones, which, together with the hardware entry barrier, increases the centralization of computational power in a few participants, decreasing decentralization and consequent system security.

Bitcoin's popularity and drawbacks fueled research around improving the Sybil proof mechanism around the usage of alternative non-forgeable resources.

PoS popularized by **Ethereum 2.0** emerged as an alternative, offering a more energy-efficient consensus mechanism. Instead of relying on energy-intensive computations, PoS uses a cryptocurrency stake as a non-forgeable resource. The more stake a participant provides, the more likely they are to extend the blockchain and earn rewards. If participants misbehave or attempt to violate system rules, their stake is slashed. This approach improves Bitcoin's energy efficiency and throughput, and, in theory, also promises a lower entry barrier and better scalability, since joining the network only requires a cryptocurrency stake. However, due to the risk of slashing (e.g., from network downtime or latency) and the high initial participation cost of 32 ETH is prohibitively high compared to many fiat currencies, users are incentivized to join staking pools, reinforcing wealth concentration, where the more stake a validator possesses, the more stake they will further accumulate. Currently, a small number of pools control a significant portion of the total stake, leading to increased centralization. This centralization poses threats to the system's correct functioning (e.g., transaction censorship, collusion, etc.).

PoSp emerged as an alternative consensus mechanism that utilizes physical resources, particularly disk space, as a non-forgeable resource, and it has been proven to be much more energy efficient when compared to Bitcoin. Earlier schemes discussed, were introduced as a promising approach. However, their reliance on large proof sizes and interactive protocols (where nodes must first send a space commitment to join the system) introduces practical limitations. This dependency on active participants and the inefficiency of large proofs represent key drawbacks, highlighting the need for more practical and scalable solutions. **Chia's PoSp** improved upon the initial schemes and introduced a Nested beyond Hellman tables [5] that successfully deters Hellman Attacks while having a compact proof size (from several MB down to 256 bytes) while achieving non-interactivity, meaning a participant can fill its disk space and actively participate in the network simply by listening to it. Chia also needs a greater threshold of $\approx 61.5\%$ honest participants, meaning that to achieve their goals of having a more energy-efficient, compact, and non-interactive scheme, they had to sacrifice this security threshold. Most importantly, despite achieving energy efficiency, Chia fills disk space with random data that serves no other purpose

than securely maintaining the blockchain, currently reaching 18.7 EiB (21 Million Tb). This is an inefficient use of storage resources that leads to increased electronic and storage waste.

Even though PoS and PoSp both achieve energy efficiency, their mining process is cheap, since no computation takes place when mining blocks. This exposes these schemes to **costless simulation** attacks that threaten the blockchain security and fairness. Ethereum 2.0 and other blockchains rely on cryptocurrency penalties to discourage miners from executing attacks. However, Chia's authors argue that penalties are not a definitive solution to this issue. Instead, Chia introduces an innovative approach by alternating PoT with PoSp, with specific mathematical research on VDFs and class groups of unknown order.

Our work will focus on useful work consensus schemes, addressing the useless space problem from Chia, and enabling the storage of web archive files within the consensus proofs. Giving dual purpose to the resources, supporting both a Sybil-proof mechanism and an archival utility, while achieving both, energy and disk space efficiency. Additionally, we want periodic storage and integrity proofs that ensure files are being correctly stored in order to build a decentralized storage network.

Chia Bread Pudding improves upon Chia by enabling the encoding of local user data within cryptographic proofs, increasing the previously nonexistent useful space in Chia to around 50%, while maintaining Chia's original properties and energy efficiency. However, it cannot be directly integrated into our system and will instead serve as a theoretical foundation. This is primarily because it lacks a proper implementation of a VDE/VDD, which is crucial for the correct operation of the scheme. In addition, it needs to address key security concerns and adapt implementation aspects to our specific requirements. Our analysis also identified opportunities to improve the useful space ratio even further, while still preserving the scheme properties. Most importantly, the current design only gives the miner the option to store useful data within the proofs, without any control or guarantees over the nature of that data (it can be either useful or useless), benefiting only the farmer. To build a decentralized storage system capable of storing web archival data, this scheme must be complemented with mechanisms for storage and integrity verification, ensuring that files are stored correctly and can be reliably validated.

PoSt algorithms are a way of complementing Chia Bread Pudding with storage and integrity verification. This type of scheme allows a prover to demonstrate that they are correctly storing specific data at the time of the challenge. **Filecoin** achieves this through PoRep and PoSpT to also guarantee storage over a time period, but most importantly, following the useful work idea, it does so in a way that the proofs used to verify the storage and integrity of the data are also the same proofs used in the Sybil-proof consensus, giving dual purpose to the data. In this way, Filecoin also achieves the same useful space idea from Chia Bread Pudding while additionally allowing the construction of a decentralized storage network.

One could argue that existing decentralized storage solutions like Filecoin could be used for our goal, but they fall short in key areas. Web archives not only archive files, but they also serve their contents to the public. Therefore, any viable solution must be transparent to users, maintain-

ing the Quality of Service, ensuring fast, reliable retrieval of stored content. Filecoin's retrieval times (often ranging from 5 to 10 minutes) are incompatible with the performance requirements of such services, leading to a poorer user experience and making the system impractical for real-time access. In addition to improving performance, our design seeks to minimize the dependence on specialized hardware for serving files and generating proofs. This increases participation, enabling a broader range of users to contribute resources and thus enhancing the resilience and decentralization of the network.

Nonetheless, our work will focus on Chia Bread Pudding as a theoretical foundation for our consensus layer, enabling the encoding of useful data within cryptographic proofs, and then we will be inspired by Filecoin PoRep and PoSpT storage and integrity verification over time, using random challenges and proving windows. With all these primitives, we can build a solid scheme that will allow us to build a decentralized storage for web archival.

System	Consensus Proof	Non-forgeable Res.	Useful Work
Bitcoin [59]	PoW	Computation	None
Ethereum [20, 34]	PoS	Stake	None
Chia [23]	PoSp & PoT	Space & Time	None
Chia Bread Pudding [44]	PoRep* & PoT	Space	Storage of local data
Filecoin [51]	PoRep & PoSpT	Space	Verifiable storage
ArchiveChain	**PoUSp & PoT & PDP**	**Space & Time**	**Verifiable storage**

Table 3.1: Comparison of systems by consensus proof type, non-forgeable resource, and useful work

Chapter 4

ArchiveChain

4.1 Motivation

The goal of this work is to design a permissionless blockchain system that can provide distributed, cost-effective, and verifiable storage of web archival data.

A key requirement for such a permissionless blockchain is a Sybil-resistant consensus mechanism. Our goal is to develop a resource-efficient protocol that avoids the massive energy consumption of PoW systems and the storage inefficiency found in more energy-efficient PoSp systems, such as the one used by Chia. Our consensus protocol builds upon Chia's Bread Pudding construction, which reduces storage waste (minimizes the amount of random data stored) in PoSp protocols by enabling miners (or "farmers") to encode local user data within their consensus proofs. This protocol follows the notion of a useful work mining process, where the stored data serves the dual purpose of participating in consensus and user data storage utility, thus not wasting any resources.

However, this approach is currently limited to the miner's own local data, and it only allows farmers to store their local files (whether random data or private documents), there is no need for storage or integrity verification mechanisms, as farmers are naturally incentivized to preserve their own data. To support the outsourcing of web archive data, we need to generalize the notion of useful work to include the storage of third-party (i.e., web archive data) outsourced data.

Inspired by Filecoin, we propose to extend our consensus protocol with periodic storage and integrity verification through a PDP mechanism. In this model, farmers prove they are correctly storing outsourced archival data, and these proofs are integrated with the underlying consensus mechanism. Specifically, the same web archive data embedded in the consensus PoSp proofs will also serve as the basis for verifiable storage proofs, thereby ensuring data storage integrity while preserving resource efficiency.

In addition to preserving data, web archives must provide fast and efficient access to their contents. Therefore, our system design must maintain the existing end-user experience, enabling seamless and transparent access to archived files, with minimal cost to the web archive service.

Our consensus mechanism retains the essential properties of Chia's PoSp protocol, where the more storage a miner has committed, the higher the probability of extending the blockchain. However, rather than leveraging useless or arbitrary data, our approach utilizes real, valuable

archival content to prove space allocation. Our protocol should allow us to simultaneosly achieve consensus while providing means of verifiable storage.

Furthermore, the incentive mechanism in our design rewards farmers proportionally based on the amount of dedicated storage and the correctness of their provided storage proofs. Farmers receive cryptocurrency for both storing data and proving its integrity. To discourage misbehavior (such as data loss or corruption) our protocol also introduces slashing penalties, thereby promoting honest participation and reliable archival storage.

4.2 Challenges and Solutions

Building a permissionless blockchain capable of providing distributed storage for the secure outsourcing of web archive files is a complex and demanding task. Systems with these characteristics are inherently large-scale and intricate, presenting numerous technical and architectural challenges. We now outline some of the challenges encountered to achieve the goals described in Section 4.1, along with the solutions we propose to address them:

- **Creating a stable blockchain network where the consensus mechanism relies on resource efficient-primitives.**

  In theory, a permissionless blockchain is not required to create a distributed version of Web Archives's storage systems. However, without it, the system would rely on altruistic participants who contribute resources purely out of goodwill. This approach significantly limits participation, security, and the network's scalability. A blockchain serves as the incentive layer, encouraging participants to dedicate storage by rewarding them with a native cryptocurrency. To build and maintain this permissionless blockchain while addressing the concerns mentioned in Section 2.1, we require a sybil-proof consensus mechanism built using resource-efficient primitives.

  For this, we introduce a PoUSp consensus protocol, where storage space is used as a non-forgeable resource tied to voting power, instead of computational power. As a result, in this protocol, a farmer can prove that they allocated some specific amount of data to the network. In this protocol, farmers have an initial computational cost of producing proofs that can be stored on disk at a negligible energy cost over time, amortizing initialization costs.

  Our design builds upon Chia's Bread Pudding, but introduces several improvements aligned with our specific goals:

  - **Verifiable Delay Function (VDF)**: Chia Bread Pudding lacks a proper VDE/VDD implementation. We will achieve similar objectives by implementing a VDF based on the SLOTH function (detailed in Section 2.7.2) to enforce a time-consuming proof generation and prevent time-memory tradeoffs.
  - **Challenge Leaf Verification**: In the Chia Bread Pudding protocol, when providing proofs for a target leaf, provers could present any leaf and Merkle Path, as there was

no way to verify its correctness. Our design introduces a way of validating randomly selected Merkle tree leaves by encoding the tree structure within the Merkle nodes, preventing second-preimage attacks, and ensuring that responses to challenges are completely correct.

- **Storage Efficiency**: We further improve the useful space ratio by encoding only to the Merkle Root commitment, using the aforementioned VDF, instead of storing all encoded tree nodes, which causes a larger overhead, while maintaining the time-consuming initialization process needed for the protocol's security.
- **Proof Binding**: In the original protocol, PoUSp proofs could be stolen by malicious farmers from honest ones, undermining the protocol's security. We address this by binding each proof to its rightful owner through the secure inclusion of the farmer's identity information within the cryptographic proof.
- **Initialization Performance**: The plotting process is optimized to fully utilize CPU resources, reducing setup time and improving accessibility for participants with consumer-grade hardware.
- **Chain selection/extension rules**: The original Bread Pudding design did not implement a functioning blockchain, it only addressed space commitment initialization, challenge, and verification. We extend it by implementing the full consensus protocol, including chain selection and extension logic. This allows nodes to achieve consensus, resolve forks, and maintain a coherent and secure ledger, which will allow us to build a functioning blockchain.

Opposite to Bitcoin, where energy demand (due to PoW difficulty increase) scales with the number of participants, our protocol's storage requirements remain constant. These properties allow for a more energy-efficient system compared to Bitcoin.

Chia already introduced a PoSp protocol that enhances energy efficiency. However, its main downside is that it fills the storage space with random data, which serves no purpose other than to maintain the blockchain consensus. In contrast, our PoUSp enables the encoding of web archive data within the stored proofs, leading to more efficient and purposeful utilization of space resources.

- **Scaling the blockchain network into a fully distributed storage system able to securely serve Web Archive data to clients.**

  To enable web archives to outsource their data to farmers, we must ensure that files are stored correctly and securely. While Chia Bread Pudding inherently relies on local user files (either random or user-useful), and users are naturally incentivized to store them properly, it does not independently provide storage integrity or verification. Therefore, we must complement this protocol with a robust storage and integrity verification mechanism to support a decentralized storage system that securely handles outsourced web archive data. To this end, we propose a PDP protocol that enhances PoUSp by enabling verifiable storage.

In our system, to prevent outsourcing attacks, files are uniquely encoded and bound to the specific miner responsible for storing them (Step 1 in Figure 4.1). To accomplish this, we introduce two types of encoding mechanisms: one based on symmetric encryption (e.g., AES-256), which relies on the secrecy of a key managed by the web archive system to produce file encodings; and a second using a VDE encoder derived from a SLOTH-based protocol, and its security relies on a large enforced encoding time that is bigger than the challenge window, while being fast to decode when being delivered to the end user. After the encoding is produced, PoUSp proofs are extracted from the file (Step 2 in Figure 4.1). Finally, a contract with important information about the agreement, metadata for verification (e.g., Merkle Root), and file information is signed and submitted to the blockchain (Step 3 in Figure 4.1). We apply Merkle Tree commitments to the files, and smart contracts periodically generate random challenge windows (PDP challenges) using randomness derived from previous blockchain blocks (Step 4 in Figure 4.1). Within these windows, farmers must demonstrate possession of the correct file by providing the challenged Merkle tree leaves and corresponding paths (PDP proofs). This verifies the integrity and availability of the data at the time of the challenge and ensures, through repeated challenges, that files are being probabilistically stored over time.

Importantly, PoUSp proofs are derived from files in the same manner as PDP proofs. This dual-purpose design supports our goal of building a decentralized storage consensus, as the computational resources used to generate PoUSp proofs also serve to verify storage integrity. Consequently, the encoded web archive data is directly accessible by farmers, allowing it to be quickly and efficiently retrieved and decoded when requested by web archive users.

Together, the PoUSp and PDP protocols enable the creation of a distributed storage network, ensuring that web archive data is safely stored with verifiable integrity, while simultaneously serving as a non-forgeable resource for consensus.

- **Defining an incentive model that rewards farmers.**

  The security and sustainability of our system depends on the incentives and penalties provided to farmers for offering storage services to the network. Our approach introduces ArchiveCoin, a native cryptocurrency designed to reward farmers. Farmers can earn rewards in two primary ways: by maintaining the blockchain, extending blocks with PoUSp proofs, demonstrating their committed storage to the network (which gives them a chance to win a block and earn a reward), and by collecting transaction fees from transactions included in those blocks. Additionally, farmers can earn ArchiveCoins by correctly storing web archive files, verified through PDP proofs submitted on the blockchain, which probabilistically confirm proper file storage, granting farmers a reward agreed upon at the time of the contract.

  However, rewards alone are insufficient to ensure secure file storage, as farmers might only prove portions of files, potentially discarding or corrupting parts of them, or they could lose

files or cease serving them. To address these risks, we implement penalties where farmers who fail PDP proofs or serve file retrieval requests incorrectly face significant cryptocurrency slashing, incentivizing correct storage of archived files. Additionally, misbehaviour can be registered by the Web Archive and may result in blacklisting and termination of contracts, allowing them to only participate in the public consensus.

- **Defining the role of the web archive and how it interacts with the system**

  The web archive serves two primary roles within the ArchiveChain system.

  First, the web archive acts as the primary source of archival files and the beneficiary of their outsourced storage. It is responsible for distributing files across the network farmers and ensuring that unique encodings are generated correctly (Step 1 in Figure 4.1). Two encoding methods are employed: symmetric encryption (e.g., AES-256, using a secret key managed by the web archive) or VDE encoding, which is computationally intensive and performed by the farmer. After encoding, the web archive validates the integrity of the encoded files. Subsequently, the web archive and the farmer agree on a smart contract, which is submitted to the blockchain to formalize the storage agreement, specifying cryptographic details, file metadata, and incentives (Step 3 in Figure 4.1).

  Secondly, the web archive serves as the initial point of contact for users requesting archived web pages (Step 5 in Figure 4.1). Operating as a proxy, it facilitates file retrieval based on the encoding method used (Step 6 in Figure 4.1). For files encoded with symmetric encryption, the web archive retrieves the encoded file from the farmer, decodes it using its secret key, and delivers the decoded file to the user. For VDE-encoded files, the web archive provides the user with the file hash and the identity of the farmer storing the file. The user can then directly retrieve the VDE-encoded file from the farmer and decode it locally, leveraging the fast decoding property of VDE. In both cases, the web archive maintains a record of file locations, contracts, and farmer reputation.

  The costs associated with the web archive's roles are minimal and amortized over time. File distribution and encoding validation occur only during the initial archival phase, after which no further processing is required for those files. The proxy role incurs negligible computational overhead, particularly with modern hardware optimized for AES decryption or the lightweight task of forwarding VDE-encoded file requests, outsourcing the work to users. Compared to centralized web archiving solutions, this approach significantly reduces storage and operational costs, offering sustainable long-term savings while maintaining high reliability and accessibility.

4.2.1 System and Adversary Model

We follow the standard assumptions in the space: Byzantine processes can behave arbitrarily, but they cannot subvert the cryptographic primitives and control at most 38,49% of the disk space [23]. Processes communicate through authenticated perfect point-to-point links.

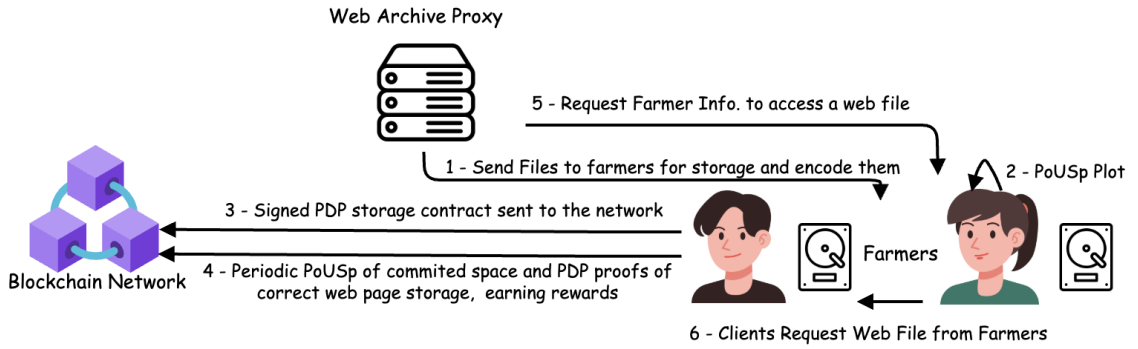

Figure 4.1: Overview of ArchiveChain's system flow, using Arquivo.pt's archival data.

4.3 Proof-of-Useful-Space consensus

Recalling Section 2.5, a PoSp protocol aims to use storage space as a nonforgeable resource to achieve consensus, where a farmer proves to the network that they are storing a specific amount of data, and the network should be able to efficiently verify it. It consists of three components: plotting (initialization), where farmers dedicate a certain amount of disk space to the network by pre-computing proofs, in a time-consuming process, and storing them in disk; challenge/farming where farmers prove their dedicated storage by producing blocks containing proofs and trying to extend the blockchain; and verification where farmers validate the proofs received from blocks and select the ones with the best quality.

As discussed in Section 3.2 and Section 3.4, our PoUSp consensus layer is built upon the theoretical foundation of Chia Bread Pudding [44]. In this section, we present its practical implementation, incorporating the improvements outlined in Section 4.2, which will be addressed in detail throughout the following discussion.

This protocol tackles two significant issues simultaneously. First, it mitigates the storage inefficiency associated with Chia by enabling the integration of useful data into the proofs required for maintaining blockchain consensus. Second, by solving the first issue, it facilitates the storage of Web Archive files within these proofs, allowing farmers to store meaningful data as part of the consensus mechanism. However, it is crucial to note that in this protocol, farmers can use any type of useful data, but it does not inherently ensure data integrity or verifiable storage. Later, in Section 4.4, we how to enhance this protocol to allow for verifiable storage of Web Archive files.

4.3.1 Plotting/Initialization Phase

Context

In this process, a farmer allocates a specific amount of storage space and fills it with proofs/plots. As discussed in Section 2.5, producing PoUSp essentially corresponds to inverting a function table. In our protocol, we aim to use dedicated commit space as a non-forgeable resource that represents voting power (identities), thereby achieving Sybil resistance. However, if the function is easy to evaluate in the forward direction, an attacker could trade-off disk space for computation and gen-

erate PoUSp proofs on the fly when challenged. This would allow them to simulate large amounts of storage, increasing their probability of winning blocks, creating longer dishonest chains, or engaging in other malicious behaviors. In such a scenario, the attacker could acquire disproportionate voting power without actually dedicating the required storage, undermining the security of the consensus protocol and enabling a range of attacks. This form of time–memory tradeoff is known as a Hellman Attack [38]. To mitigate Hellman time-memory tradeoffs, Section 2.5 showed that the underlying function to be inverted must not be easy to evaluate in the forward direction. Instead, plotting should be an intentionally expensive and time-consuming process, which incentivizes rational farmers to initialize and store proofs on disk and amortize the initialization cost over time, rather than attempting to generate proofs on demand, because probabilistically, they could not generate them fast enough or without spending immense amount of resources. Moreover, the probability of having a winning proof should be proportional to the amount of dedicated space. In an ideal protocol, no time–memory tradeoff would be possible, and an attacker could only produce PoUSp proofs if the corresponding space were genuinely allocated. Although this cannot be perfectly achieved in practice, the initialization process can be prohibitively costly, making it computationally expensive and effectively discouraging such attacks by making it impossible to generate proofs within the challenge time.

Merkle Tree Storage Commitments

In the initialization phase of our PoUSp protocol, we first require a storage commitment that enables farmers to prove they are genuinely storing a piece of data, building on the Chia Bread Pudding foundations, we employ a Vector Commitment (VC) scheme [22] that enables farmers to commit to an entire vector (a set of data blocks) while allowing for later revealing/opening of individual values (specific blocks) along with cryptographic proofs that these values are part of the originally committed vector, if the prover maintains knowledge about the elements of the entire vector. Such a scheme offers efficiency in both the commitment and opening phases.

Specifically, we adopt the**Merkle Tree [56, 57] Commitment** as our VC storage commitment scheme. In our construction, we first split the data to be committed into a vector of blocks $[m_0, m_1, \ldots, m_{n-1}]$ of size S (e.g., 64-byte segments), which serve as the leaves of a binary Merkle tree. Each leaf is then hashed using a cryptographic hash function (e.g., SHA3-256) to produce a unique, fixed-size digest. The resulting hashes are grouped into pairs, with each pair concatenated and hashed to produce a parent node. This process of pairing and hashing is applied recursively until a single hash value remains, known as the Merkle Root, which is the commitment of the data being stored.

We then use the Merkle root commitments of the data being stored to probabilistically prove the vector's correct storage without requiring the download of the entire data. To ensure that data is correctly stored, farmers are periodically challenged to prove that they are storing a specific leaf m. When challenged, a farmer must produce an opening of the Merkle root that demonstrates the leaf is part of the committed vector of the data being stored. This opening consists of a leaf m,

along with the corresponding Merkle path, a sequence of hashes sufficient to recompute the Merkle root by sequentially hashing the leaf node with each hash along the path. During verification, if the recomputed root matches the original Merkle root commitment, it is confirmed that the leaf m is part of the committed data.

Any change in the data leaves propagates to the root, making the Merkle root a binding commitment that fixes each leaf value and ensures that the prover had access to all elements of the vector at the time of a challenge. While farmers may later discard some leaves, they cannot produce valid proofs for any omitted or modified elements without either retrieving them, storing intermediate sub-tree roots, or computing a computationally infeasible hash collision.

This protocol is also efficient, as it allows for compact proofs of size $(\log_2(n)+2)\times 32$ bytes$+$ 64 bytes), where the 32 bytes correspond to the output size of the chosen hash function (SHA3-256), and the 64 bytes correspond to the leaf size S.

Introducing Computational Delay through SLOTH

However, generating Merkle Trees alone is insufficient to prove storage, as their construction involves only simple hashes of the data blocks, which can be computed extremely quickly. This efficiency allows farmers to generate random and quickly produce the respective Merkle Root commitment without actually storing the data on disk over time. This way, when challenged, a farmer could generate an immense amount of Merkle Root commitments to invert the function table and find the one that best answers the challenge, exploiting Hellman's time-memory tradeoffs and pretending to store more data than they really reserved, breaking the PoUSp protocol.

To make Merkle Trees computationally expensive to generate and deterring Hellman Attacks, Chia Bread Pudding encodes nodes in the tree using a VDE that can be made as slow as desired to encode while still allowing fast decoding via a VDD. However, one of the main specified limitations of the original work is the lack of a concrete VDE/VDD implementation, which is critical for the correct operation of the protocol. To address this, **we implement a VDF based on the SLOTH construction** (introduced in Section 2.7.2). SLOTH functions act as a sequentially slow hash function, with a parameterizable delay that enforces computationally expensive operations while being fast to verify. Our implementation is defined as follows:

- **SLOTH.Hash(*data, t, p*)** $\rightarrow \pi$: Computes the SLOTH hash of *data* through a deliberately computationally expensive slow process. Where t specifies the number of iterations, mapped to real-world execution time, and p is a prime with K security bits. We choose $p = 2^{256} - 189$, the largest 256-bit prime satisfying the SLOTH criteria. The output π is the resulting SLOTH hash. Both t and K are security parameters of the protocol.

- **SLOTH.verify(**π***, data, t, p*)** $\in \{\text{reject}, \text{accept}\}$: Efficiently verifies that the SLOTH hash π corresponds to the input *data*, using the same parameters as in the hash computation.

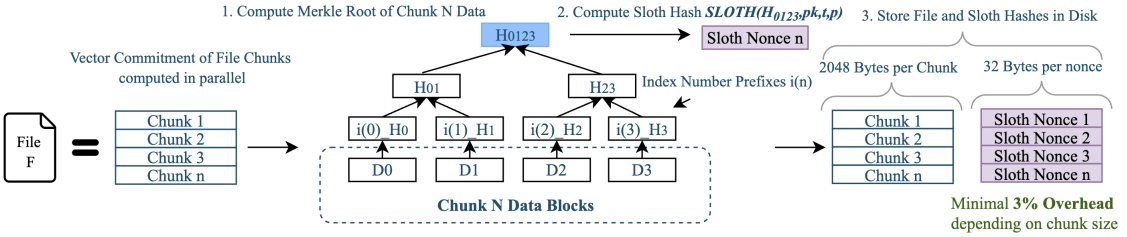

Figure 4.2: Initialization phase

Plotting Procedure

Our improved plotting algorithm consists of a parallel construction of Merkle Trees (used as space commitments) for each file chunk. Leaves are embedded with index positions, and SLOTH encoding is applied to each root to enforce time-intensive computation. The procedure is as follows:

Step 1 - Starts when file data D is received for plotting. The farmer calls **PoUSp.plotFile** specified in Algorithm 1 (see Appendix A). It starts by dividing D into chunks of size M in line 1. For each chunk, processed in parallel in the for loop that starts in line 3, the data is further split into N leaves of size S in line 4. Each leaf is prefixed with its index position (from 0 to $N-1$) in line 9, for each chunk. A Merkle Tree is then constructed by recursively hashing each pair of child nodes up to the Merkle Root in the inner loop at line 8.

Step 2 - Once the roots of all chunks are computed, a time-expensive computation is introduced in line 16. For each chunk root, compute the $SLOTH.Hash(\text{root}\|pk, T, P)$ hash, where pk is the farmer's public key (used to bind the proof to its owner), P the prime used, and T is the time parameter controlling the computational delay. Producing a SLOTH hash output.

Step 3 - After all SLOTH computations are completed, the data and their corresponding SLOTH hashes/proofs are written to disk in line 19. When challenges are received, the SLOTH hash outputs quality will be compared against the challenges.

Improvements Over Chia Bread Pudding

In our improvements, we encode the tree structure within the nodes through index position prefixes into the leaves of the Merkle tree. This allows the verifier to confirm that a challenged leaf was correctly returned and also prevents second-preimage attacks, something that was not possible in the original Chia Bread Pudding implementation, as a prover could present any leaf, and the verifier would have no information to validate that it was the targeted leaf. Index prefixes are secure because they are embedded within the hashes that generate the Merkle Tree root, ensuring that any modification to the prefixes would alter the root, rendering the proof invalid. Because indexes are only required to reconstruct the root and can be appended during Merkle Tree computation, they do not need to be stored on disk and therefore introduce no overhead.

Additionally, we introduce a proper VDE based on SLOTH encoding. Unlike the original protocol, we do not SLOTH encode every node in the tree (which requires storing SLOTH outputs for each node), only the root, since it is the minimum space needed to introduce a computational

bottleneck to mitigate Hellman time-memory tradeoff attacks. By applying SLOTH only to the root, we can store the original data (and reconstruct Merkle Trees from it when challenged) and a single SLOTH proof metadata separately. This design improves the useful space ratio to $\approx 97\%$, compared to around 50% in the Chia Bread Pudding approach. Furthermore, our implementation is optimized to utilize all available CPU cores to process the multiple data chunks in parallel, significantly enhancing initialization efficiency.

In the original Chia Bread Pudding, PoUSp were not bound to the farmer's identity. This allowed a malicious participant to receive a block containing a valid PoUSp and attempt to steal it by presenting it as their own proof. To prevent this, we include the farmer's public key as input to the SLOTH hash. This ensures that the resulting proof can later be verified against the farmer's public key, since only the corresponding private key can produce a valid block signature. The same public key, included in the SLOTH hash input, is then used to validate the block signature, thus binding the proof to the rightful farmer. Consequently, if a malicious farmer attempts to steal such a bound PoUSp, they will be unable to generate the corresponding valid block signature, since they do not possess the private key associated with that public key used in SLOTH.

The original Chia Bread Pudding encrypts data before plotting to increase entropy and privacy; in contrast, our protocol does not require such encryption since the data is a public archive without privacy concerns. Moreover, it is already pre-encoded by the PDP protocol, described in Section 4.4, which inherently provides the entropy necessary to ensure proofs are uniformly distributed across their value space. However, if farmers wish to use personal data or any other data at their discretion, they should consider the entropy and the possible need for encryption.

Considerations

This initialization step can be repeated multiple times for different data, with each file being stored alongside its corresponding PoUSp proofs in the form of SLOTH hashes derived from the file chunk's Merkle Roots. These SLOTH proofs act as the nonces or tickets required to win blocks in our protocol. Intuitively, as more space is dedicated and more files are stored, the more chunks data can be split into, and consequently, the more SLOTH proofs can be generated and the higher the likelihood of winning blocks and rewards by successfully extending the blockchain.

By making the usable nonce dependent on the SLOTH output, we limit the proof production rate to the time parameter T, making the process inherently time-consuming. This prevents Hellman attacks, as miners are incentivized to store and reuse the data and proofs generated during this phase over time, while also making replotting attacks (where farmers allocate space only when a challenge is issued) expensive. This strengthens blockchain security, as miners who have invested time in the lengthy data initialization process are motivated to protect and maintain that data. Their effort ensures that useful data remains a reliable foundation for the ongoing stability and security of the blockchain.

It is important to note that storing only the nonce is insufficient. The data and nonce are interdependent, meaning that if the data is discarded, the farmer would need to find a hash collision

in SLOTH, which is computationally infeasible. Similarly, discarding the nonces would require regenerating the corresponding data, which is also computationally infeasible. Therefore, farmers must correctly store both the data and their respective nonces.

4.3.2 Challenge Phase and Proof generation

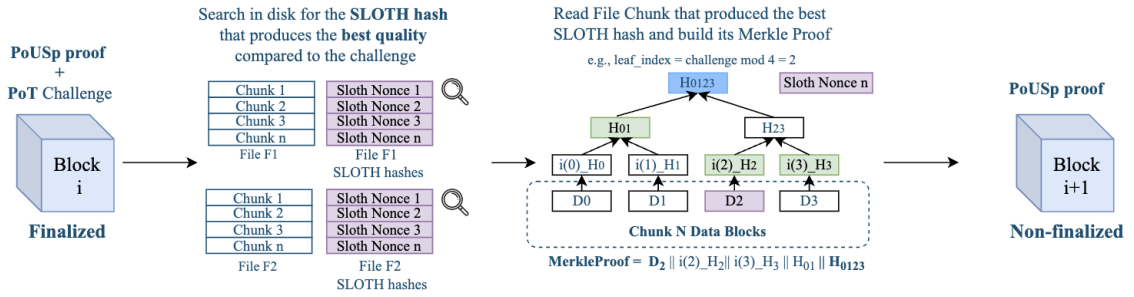

Figure 4.3: Proof Generation — The colored nodes represent the Merkle Proof within the Merkle Tree, while the purple node on the right represents the SLOTH Hash proof.

Periodically, the network challenges farmers, derived from PoT of the last block, to provide PoUSp proofs. This is an opportunity for them to prove that they are dedicating some amount of space to the network and a chance to win rewards from farming the block and processing transaction fees.

A proof consists of an opening of the VC scheme, in this case, a Merkle Root, where a subset of hashes can be used to prove a specific leaf is present in a tree. This subset is composed of the leaf itself and its corresponding Merkle Path, as explored in Section 3.2 and Section 2.5. Note that to produce a correct Merkle Proof for a specific leaf, the farmer should have knowledge of every other leaf in the Tree.

When a farmer receives a PoT challenge from the network, it searches its disk for the SLOTH hash that produces the best quality (e.g., minimal bitwise difference) relative to the hash of the PoT. Once this best SLOTH hash is identified, the Merkle tree derived from the encoded file chunk that produced the SLOTH hash is recomputed. The PoUSp protocol requires storing the whole data responsible for generating this Merkle tree and its corresponding SLOTH hash. Consequently, to prevent farmers from storing only a single Merkle Path, or from discarding or altering any leaves of the Merkle Tree, the protocol randomly selects leaves along with their corresponding Merkle paths. This probabilistic guarantees that to correctly respond to challenges, farmers must store all leaves (the entire committed vector) in their original form, as they cannot predict which leaves will be requested at each challenge. The randomness is derived from the blockchain itself, specifically from the previously introduced PoT hash, which farmers cannot anticipate or manipulate. The challenged leaf is determined using the expression $leaf\_index = challenge \bmod N$ (the index is encoded in the value of the leaf for later validation), where N denotes the total number of leaves in the tree. The resulting proof consists of the challenged leaf together with its Merkle path that allows recomputation of the Merkle Root, and the SLOTH hash of the root, which serves as the

nonce that may win the block. After producing the PoUSp, the farmer can attempt to extend the blockchain by broadcasting a **non-finalized block**, a block without the PoT computed in its contents. Timelords then receive these blocks and, if they are winning candidates, compute the corresponding PoT, thereby finalizing them.

The challenge and proof phases are designed to remain efficient for honest farmers (those who correctly initialized and maintain their data and SLOTH proofs), as they require only straightforward disk lookups. Both the data and the SLOTH proofs can be reused multiple times after the initialization phase throughout the blockchain's lifetime, amortizing the initial computational cost over time.

4.3.3 Proof Verification Phase

When farmers or a timelord receive a block, they must validate the PoUSp proof associated with a challenge to ensure that it meets the required criteria for securely extending the blockchain. This validation involves multiple checks, which can be divided into three categories: verifying block quality, validating the Merkle Proof and SLOTH proof, and verifying the challenge and blockchain extension rules.

The farmer first verifies if the public key present in the block message correctly validates the signature of the block produced by the private key of the farmer, that the quality of the block exceeds the current difficulty of the blockchain, which is dynamically adjusted to the available resources of the network, and that it represents the block with the best quality observed at that specific height. The extension rules will be discussed in detail in Section 4.3.5. The farmer then checks whether the block correctly answers the most recent PoT challenge and extends the chain with the most amount of accumulated space.

By evaluating quality first, blocks that do not meet this criterion can be immediately discarded, regardless of whether their proofs are correct or not, saving unnecessary computation for the farmer.

When verifying the cryptographic proofs, the farmer begins by validating the Merkle Tree structure provided in the Merkle Proof and ensuring it complies with the protocol's public parameters that ensure space was dedicated to generate a Merkle Tree. First, it checks the total number of leaves N present, verifying that $Merkle\_Path.size == \log_2 n\_leaves$, along with the presented targeted leaf. Next, the farmer computes the target leaf index using $leaf\_index = challenge \bmod N$ and confirms that the provided leaf contains a prefix that matches the computed prefix (and direct sibling prefix matches $leaf\_index + / - 1$), ensuring that it is indeed the correct target. The farmer also checks that the leaf has the correct size S, a public parameter of the protocol.

In the next step, the farmer recomputes the Merkle Root from the Merkle Proof and verifies that it corresponds to the expected input that generated the provided SLOTH hash nonce, which gives the chance to extend the blockchain. To recompute the root, the farmer hashes the leaf node, combines it with the first node in the Merkle Path, and iteratively hashes the result with each

subsequent node in the path until no nodes remain. The final hash should be the resulting Merkle Root. Finally, the farmer verifies that this Merkle Root correctly produced the SLOTH hash by calling ***SLOTH.verify(sloth hash, Merkle Root*** || ***FarmerPk, t, p)***, where t and p are the iteration count and prime number, public parameters that must be identical across all participants, and the FarmerPk is the public key that correctly verifies the block signature and as a result if it was not the same as the farmer that generated the PoUSp, the block will be rejected. If the verification returns true, the proof is valid, and the farmer has probabilistically demonstrated that it is storing a vector of data of size M used to generate the Merkle Tree and corresponding SLOTH proofs. In other words, the farmer proves that it has committed a certain amount of storage through the Merkle Proof, reinforced by the computationally expensive initialization introduced by SLOTH.

The following enumeration provides a more condensed version:

1. Verify that quality is bigger than the current farming *difficulty* and it's the best quality block at a specific height.
2. Validate that the block correctly extends the blockchain and answers to the most recent challenge for that specific height.
3. Verify if $Merkle\_Path.size == \log_2 n\_leaves$ and that the Merkle Proof contains the targeted leaf (Preimage Attack Prevention).
4. Verify if the encoded prefix in the leaf matches the index given by $leaf\_index = challenge \mod n\_leaves$, and direct sibling prefix matches $leaf\_index + / - 1$. (Preimage Attack Prevention).
5. Verify if the presented leaf has the expected size S.
6. Recompute the Merkle Root from the Merkle Proof and verify if it is the correct output to the SLOTH hash through ***SLOTH.verify(sloth hash, Merkle Root*** || ***FarmerPk, t, p)***, validating t and p in the process, where the FarmerPk is the farmer's public key that validates the block signature.

If all the validations are valid, the proof is considered correct. Depending on whether the verifier is a timelord or a farmer, they may then proceed with their respective protocol algorithm.

Following the original Chia blockchain consensus design, PoT challenges are issued every 9 seconds, with a maximum window of approximately 28 seconds to broadcast a non-finalized block with the respective PoUSp. This small time frame guarantees that the data generated during the computationally intensive initialization phase remains correctly stored and available throughout the blockchain's lifetime, as farmers cannot feasibly generate such data on demand at the moment of a challenge, since doing so would be prohibitively slow or require excessive computational resources. As a result, they are compelled to retain the initialization data for future proof generation.

Additionally, steps 3, 4, 5, and 6 (adapted from the original) were not present in the original Chia Bread Pudding, so they are part of the improvements and security concerns that we needed to our construction.

Quality Function

The quality function has several purposes, namely, it helps to compare proofs in a deterministic way, where better proofs correspond to a higher-quality function output. It also makes farming more efficient, since farmers only need to read the stored SLOTH hashes and compute the quality function without reading/computing the entire proof, since full proofs are only produced when the best quality SLOTH hash is found. The probability of getting matching bits follows a uniform random distribution, and every bit position has an equal likelihood of matching. This makes this function fair, deterministic, and unbiased, making the likelihood of finding a better quality increase with the amount of space allocated. This incentivizes farmers to dedicate more space to the network since the more proofs they store on disk the higher the chance of finding a proof with a quality good enough to produce a winning block. Currently, in our implementation, this quality function is given by:

$$quality = \frac{bitwise\_difference(challenge, nonce)}{max\_difference}$$

and a difficulty parameter d that works similar to a plot filter in Chia [23], where the first k bits of the challenge should match the first d bits of the proof, where d can be dynamically adjusted to moderate the amount of produced proofs and making it more difficult to perform replotting attacks.

4.3.4 Relative attacks and countermeasures

The PoUSp data structure is inspired by Chia Bread Pudding [44], however, the protocol participants can be either a Farmer or a Timelord, and their behaviors are inspired by Chia [23] and how they prevent grinding, double dipping, long-range attacks, and how they work on the longest accumulated chain space.

Long-range attacks: Similarly to Chia, in our PoUSp protocol, proofs are computed rapidly, allowing an attacker with sufficient space to bootstrap a heavier chain in a short time, opposite to Bitcoin where an attacker is required to have computational power above the historical average. To mitigate this, we also alternate between PoUSp and PoT, meaning one serves as challenge to the other alternately. By requiring PoT between PoUSp, an attacker is required to spend a real-time delay between computing the VDF, which makes it impractical to grow a chain larger than the honest chain since the attacker's chain growth rate is bound to his single-core hardware speed, meaning, an attacker would need to compute the VDF significantly faster (along with sufficient space) than the honest participants, which given today's possible hardware advantage it's not possible. To make it even more difficult the time spent in VDF is inversely proportional to the quality of the PoUSp, meaning the better the quality the faster the VDF is computed which prioritizes the chain with the most accumulated space to grow faster.

Grinding attacks: To prevent farmers from modifying the block payload to produce challenges that increase the odds of winning blocks, we follow the Chia proposal where the PoUSp and

PoT proofs are decoupled from the payload of the block, meaning that modifying the block payload does not produce changes on the proof's of a block and consequently do not affect consensus odds of winning blocks.

Double Dipping attacks: Farmers can also extend multiple blocks, again, inspired by Chia proposal, we make this part of the protocol instead of using penalties. Farmers and Timelords work on the first K blocks at every depth. Being K is a security parameter directly affecting the honest threshold, where higher values allow for higher honest thresholds at the cost of more work participants perform.

4.3.5 Timelord and Farmer Algoritmhs

Farmer Algorithm: After plotting, the farmer starts the main farming loop, where he listens to finalized blocks from the network. The algorithm begins when a farmer receives finalized block (Possesses a valid VDF and PoUSp) at depth i, it verifies if the block is valid (extends the current chain with the most accumulated space and has valid transactions), and fresh (valid but never seen), and in case of being a new heaviest chain not yet seen, it performs synchronization and chain selection rules. Then, the farmer runs Algorithm 2 (see Appendix B), where it starts by validating the block's PoT (line 4). Then, because farmers work on the first K chains they see, if they have seen more than K blocks at that depth, they ignore the block (line 7). Otherwise, it increments the counter of blocks seen at this depth (line 10). Then, the output of the VDF is signed (line 11) and the hash of this signature is used as challenge input (line 12) to PoUSp (alternating PoT and PoUSp) (line 13). Finally, the new PoUSp is computed from the local farmer plots (line 13), the new block's payload is generated (line 15), and the corresponding signature for the payload and PoUSp is generated (line 16). Finally, the unfinalized block is assembled with the previously generated proofs and data and broadcast to the network (line 17, 18). At the end, the blockchain's local view is updated with the received block and the information from the newly mined block. Blocks that are received at future depths should require chain synchronization or be cached for later processing.

Timelord Algorithm: Algorithm 3 (see Appendix B) shows what happens when a timelord receives a non-finalized block that lacks a VDF output but has a valid PoUSp. It starts by checking the block's depth. The new block is ignored if there are already K finalized blocks at the same depth i (line 4). Otherwise, the timelord computes the PoT challenged derived from the block's PoUSp (line 7, alternation between PoUSp and PoT) and calculates the required number of VDF iterations taking into account the block's PoUSp quality (line 8 and 9). The time t spent on the VDF is inversely proportional to the quality of the PoUSp and is defined as $t = \frac{T}{Q}$, where T is a constant regulating the VDF speed to maintain the desired block time taking into account the blockchain space resources, and Q is the quality of the PoUSp of the non-finalized block. Two scenarios then arise:

1. **Fewer than K finalized/running blocks at depth i** (line 10): If the number of finalized blocks and blocks with ongoing VDF computations at depth i is less than K, the timelord

begins computing the VDF for the received block (line 11) and increments the counter of blocks at this depth (line 12).

2. K **or more finalized/running blocks at depth** i (line 13): If there are already K finalized or running blocks at depth i, the timelord compares the VDF time of the slowest running thread at i with the VDF time of the new block. If the new block's VDF time is faster, the timelord aborts the slowest VDF thread (line 16) and begins computing the VDF for the new block (line 17).

By prioritizing VDF computations for blocks with faster VDF times (which correspond to higher PoUSp quality), this algorithm ensures timelords focus on blocks that disseminate quickly and have greater quality. This approach prioritizes working on the chain with the highest accumulated quality. In addition, farmers are incentivized to disseminate blocks quickly since high-quality blocks that arrive late may fail to outpace the slowest VDF thread, discouraging withholding attacks. As a result, this mechanism extends the chain with the most accumulated space and minimizes delays. Blocks that are received at future depths should require chain synchronization or be cached for later processing.

We chose to implement Wesolowski's VDF [74], as described in Section 2.7.1, to ensure a secure time delay between blocks, similarly to Chia. In theory, our SLOTH VDF implementation could also be reused in the PoT component. However, since this scheme is not asymptotically verifiable, it would still require a significant amount of time despite being considerably faster than the SLOTH hash computation. This makes it challenging to fit within the constraints of the blockchain's block time schedule.

4.4 Verifiable storage with Proof-of-Data-Possession

4.4.1 Context and Challenges

The presented PoUSp mechanism enables the construction of a secure blockchain that reaches consensus on a consistent ledger view by leveraging the space committed by farmers as a non-forgeable resource. By building on Chia Bread Pudding's primitives, our PoUSp mechanism inherits its properties, and only allows the embedding of useful data within the consensus proofs, it does not guarantee that data is being correctly stored. However, to support a decentralized storage network capable of outsourcing web archive files, it becomes necessary to introduce mechanisms that ensure both the integrity and continuous availability of outsourced archival content, which is critical for preserving the historical record of the internet. To address this, we propose integrating a PDP protocol at the application layer (blockchain transaction layer) to preserve non-interactiveness in consensus participation. This addition complements the PoUSp (consensus layer) with verifiable storage, enabling web archive files to be securely stored within consensus proofs while providing guarantees of their correct storage.

This integration also extends the concept of useful work from the consensus protocol. Although the usefulness of PoUSp was limited to reducing storage waste by allowing farmers to

store personal data in proofs, our approach extends this notion to include the verifiable storage of outsourced web archive data, making the work not only useful for the farmer but also valuable for a community goal. Following Filecoin's useful work definition (Section 3.3), the same resource/proofs used to participate in the consensus through PoUSp are used to provide PDP proofs of verifiable storage, being valuable beyond securing the blockchain, improving on PoW energy waste and Chia storage waste.

Our PDP protocol is a central component of this solution, allowing the outsourcing of web archive data and consequently reducing the web archive costs. To achieve this, the protocol addresses several key challenges:

- **File integrity and continuous storage**: To achieve our goal of building a decentralized storage network capable of allowing the outsourcing of Web Archive data to the blockchain participants, this protocol must ensure that files remain unmodified and that they are being correctly stored over a period of time.

- **Outsourcing Attacks**: The protocol must prevent farmers from faking their storage capacity by quickly fetching proofs/files from other storage providers. Such behavior leads to an unfair reward distribution and weakens the system's redundancy. For example, a file intended for independent storage across multiple farmers may be held by only one, with others relying on outsourced proofs. This undermines network reliability, as the protocol cannot verify genuine data storage, increasing the risk of data loss if the sole provider fails. It also reduces centralization, contradicting the decentralized and fault-tolerant design of the system.

- **Efficient Proofs and Data Retrieval**: To ensure long-term data integrity, files must be continuously and verifiably proven to be correctly stored, without retrieving the whole file. However, since the verification protocol operates at the blockchain's transaction layer, both the proof size and computational overhead must remain minimal to avoid overloading the network. Additionally, Web Archives are responsible not only for storing files but also for ensuring fast and reliable access for end users. Therefore, the design of this protocol must maintain a transparent transition for the end user and current retrieval systems, preserving low-latency access and maintaining a smooth user experience without compromising proof efficiency.

- **Minimal Web Archive Costs**: The main objective of this work is to reduce the increasing storage and operational costs that Web Archives face due to growing digital content. Therefore, the protocol must achieve cost savings over existing centralized solutions, ensuring the Web Archive's involvement remains minimal over time, both in storage verification and file access, with costs offloaded to incentivized blockchain participants.

4.4.2 Proof-of-Data-Possession protocol

The PDP protocol begins with the farmer allocating storage (e.g., 32GB) to store Web Archive Files. Then the **Archival Phase** takes place, where the farmer receives files from the Web Archive until the space is fully utilized. In this phase, files are encoded into unique representations of that storage replication that are bound to the farmer. Once the farmer possesses the encoded files, he extracts PoUSp proof from them and is able to participate in the consensus using that data. Following, there is the **Integrity and Storage Challenge Phase**, where smart contracts initiate periodic challenge windows during which farmers are required to provide cryptographic proofs for randomly selected segments of a stored file. This probabilistically guarantees that files are being stored correctly continuously, while also providing incentives for farmers who provide correct proofs, and penalizing farmers who do not. Finally, in the **File Access Phase**, clients can efficiently retrieve stored files, either through a Web Archive proxy or directly from the farmer.

Archival Phase

Recalling Section 2.6 and Section 3.3, in a PDP protocol, the outsourced data must be preprocessed to produce a unique cryptographic encoding and metadata for later verification. A critical requirement of this encoding is that it must be unreproducible within the challenge proving window. Otherwise, the protocol becomes vulnerable to outsourcing attacks, where farmers could regenerate the encoding by retrieving the original file (either from other farmers or by posing as end users), thus faking proper file storage and compromising storage security.

In our protocol, we use two distinct encoding methods to guarantee uniquely replicated, verifiable, and secure file storage. Each method binds the encoding to a specific farmer, ensuring that even identical raw data results in distinct encodings across different farmers. To achieve this, each encoding incorporates a public Salt/IV, derived from the hash of the farmer's public key, the file identifier, and a replication counter, according to the expression: $\underline{\text{Hash(farmerPk + fileId + counter)}}$. This ensures that two malicious farmers cannot collude by pretending to store the same file while only one stores a copy, nor can a single farmer falsely claim to store multiple replicas of the same file without dedicating the corresponding storage space. In other words, the protocol prevents **collusion and storage deduplication** (two forms of outsourcing attacks). This ensures that only the farmer who generated a unique encoding can use it to produce valid proofs of storage.

We now present the two types of encodings employed in our protocol:

- **Symmetric Encryption Encoding (AES-256)**: This encoding approach is based on symmetric encryption. Specifically, we use AES-256 in CTR mode, which is fully parallelizable, highly performant, and secure as long as the salt is never reused with the same secret key. The security of this encoding relies on the confidentiality of a secret key Sk owned by the Web Archive service. This allows us to generate unreproducible and unique file encodings bound to each farmer, by using the salt $\underline{Hash(farmerPk + fileId + counter)}$. If a farmer loses, corrupts, or attempts any malicious behavior involving the absence of their

original encoding, they will be unable to reproduce it due to lack of access to Sk. As a result, they will fail to generate valid proofs of storage, and such behavior will be detected, forcing the farmer to store his encodings correctly.

- **SLOTH-Based VDE**: This encoding method is an adaptation of the SLOTH VDF (mentioned in Section 2.7.2).[1] into a time-asymmetric permutation [8], enabling the construction of VDE and VDD with computationally intensive encoding and rapid decoding.

  The security of this approach does not rely on the confidentiality of a secret key. Instead, it depends on the significant computational delay introduced by SLOTH, which requires t seconds for encoding, and the security parameter defined by the bit size of a prime $p = 2^{256} - 189$, the largest 256-bit prime available for SLOTH criteria. If a farmer loses, corrupts, or engages in malicious behavior that compromises the original encoding, they cannot reproduce it within the challenge proving window or file retrieval threshold time, as long as the encoding time t exceeds both this time periods. Consequently, they will fail to generate valid proofs of storage in time, ensuring detection of such behavior and incentivizing proper storage of encodings.

  A limitation of the Subspace SLOTH library is its size restriction to the encoding of fixed-size 4096-byte blocks per $SLOTH_VDE.encode(piece, Salt, T)$call, where $piece$ is a 4096-byte file chunk, and T represents the number of computational iterations, directly corresponding to real-world clock time. To handle files larger than 4096 bytes, we developed a custom Cipher Block Chaining (CBC) encoding protocol using the Subspace SLOTH library as its core. This maintains the sequential computation responsible for the time-intensive process by enforcing a precedence relationship between encoded file chunks. This is done through the Salt values, where for the first block in the sequence, it matches the previously mentioned binder $Hash(farmerPk + fileId + counter)$, while for subsequent blocks it is computed through $Hash(previousBlock)$. This way ensures sequential SLOTH encoding/decoding for each block of the file

The **Archival Phase** begins when a node joins the network and allocates a specific amount of storage (e.g., 32 GB) for archiving Web Archive data. The node first communicates with the Web Archive to perform registration by sending a signed message containing its wallet address, IPv4 address, listening port, dedicated storage capacity in bytes, and its system identification in the form of a public key, which can be used to verify the signatures of its messages. After registration, the Web Archive can initiate file archiving on the farmer, initiating the encoding phase. If AES encoding is selected, the Web Archive encodes the file using its own secret key Sk and sends the encoded file to the farmer along with a pre-signed storage contract specifying cryptographic details for integrity and storage verification (Merkle Root), file metadata, and incentive values. If VDE

[1] In this work, we implemented two SLOTH-based schemes: one constructs a slow-timed hash function (VDF) to mitigate Hellman attacks in the PoUSp protocol, and the other develops a slow-timed encoder/decoder for CBC file encoding in PDP.

encoding is selected, the raw file is sent to the farmer, who performs encoding locally. Since the encoding occurs on the farmer's side, the cryptographic details for integrity and storage verification must be confirmed by the Web Archive. In this case, after completing VDE encoding, the farmer sends the VDE encoded file back to the Web Archive, which is then VDE decoded to verify that it matches the original file. If verification is successful, the Web Archive sends the pre-signed storage contract to the farmer with the relevant cryptographic details for the VDE encoding. In both encoding approaches, the farmer must verify and agree to the contract, sign it, and broadcast it as a blockchain transaction to finalize the storage deal. Finally, the Web Archive increments the file's replication counter, and the farmer extracts PoUSp proofs, using **PoUSp.plotFile**, from the stored encoding, enabling participation in consensus. This ensures that Web Archive data is used for both PDP proofs of storage and PoUSp proofs securing the blockchain, aligning with our notion of useful work consensus.

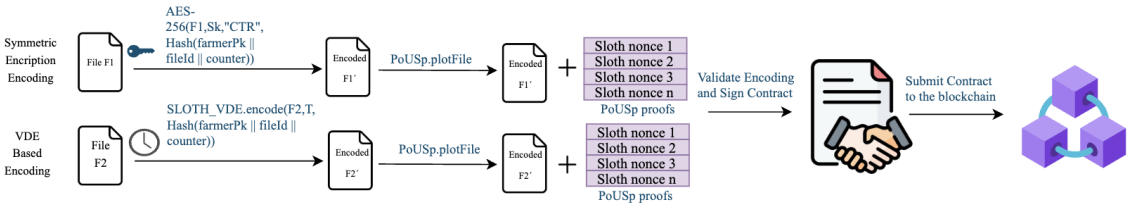

Figure 4.4: Illustration of both encoding types: the top shows Symmetric Encryption encoding, while the bottom shows SLOTH VDE based encoding. In both cases, the process results in an encoded file stored on the farmer's disk, along with PoUSp proofs derived from that encoding.

It is important to note that the VDE approach increases the farmer's computational overhead during both the initialization process of filling disk space, as the encoding time t must exceed the proving and retrieval windows. For small files, a setting such as $T = 1$ minute would impose excessive overhead, making it impractical to fully utilize storage capacity and participate in the network, disincentivizing adherence to this approach. Consequently, the VDE approach is better suited for large files. To choose between VDE and AES encoding, we define a file size threshold[2] above which VDE's encoding time reaches at least t, a security parameter of the VDE method. Careful tuning of T and incentive structures is necessary to ensure farmers adopt VDE encoding, as it most effectively shifts operational costs from the Web Archive to network participants (discussed later in this work).

The Algoritmhs present in the Appendix C illustrate the Archival Phase described above. Algorithm 4 details the process from the farmer's perspective, when receiving files for archival, while Algorithm 5 describes the process from the Web Archive's perspective, when archiving a file on a farmer:

This phase represents the most resource-intensive task for the Web Archive, as it must distribute files, potentially perform and validate decodings, and validate contracts to establish storage

[2]This is because for large inputs, VDE encoding is processed in 4096-byte blocks, causing the SLOTH schemes's encoding and decoding complexity to scale with the number of blocks n. The complexity is $\mathcal{O}(n \cdot T \cdot \log p \cdot M(p))$ for encoding and $\mathcal{O}(n \cdot \log p \cdot M(p))$ for decoding. Thus, n introduces a lower bound time of encoding/decoding.

deals. However, over time, the workload for each file decreases to simple retrieval operations, amortizing initialization costs over time, enabling reductions in both storage and operational costs compared to a centralized solution. During this process, the Web Archive must maintain a database of registered participants and their storage contracts, allowing it to track in which nodes specific data is being stored, to easily access them, and to monitor storage statistics and replication levels of the stored data.

The salts used in the encoding phase serve solely to generate randomness, ensuring each encoding is unique and bound to the farmer. These salts can be public without compromising encoding security, as their values can be extracted from the contract. This eliminates the need for the Web Archive to store multiple secret keys, being only required to secure a single key and having no key management overhead, although the decision of using multiple keys and managing them is open to the Web Archive.

Storage and integrity Verification Phase

The Archival phase focuses exclusively on enabling the Web Archive service to store data from farmers using unique, farmer-specific, non-reproducible file encodings. Our approach also requires a method for continuously verifying both the storage and integrity of files archived by farmers on the blockchain. One straightforward solution would involve the Web Archive randomly requesting the full file from farmers to confirm correct storage at a given time. However, this is neither efficient nor scalable, as it creates significant participation and communication overhead, exacerbated as the number of farmers increases. Instead, an automated mechanism is needed that avoids downloading entire files while ensuring minimal Web Archive workload and efficient communication. To achieve this, we leverage Smart Contracts to periodically issue challenges for random file segments and validate the resulting file proofs.

Merkle Tree commitments. To construct an efficient scheme, we employ the same Merkle Tree Storage commitments described in Section 4.3.1, which enable probabilistic proofs of both file storage and integrity over time without requiring full file downloads. For this, the file encoding data is treated as a vector to be committed by splitting it into leaves $[m_0, m_1, \ldots, m_{n-1}]$ of size S (e.g., 64-byte segments). A Merkle Tree is then built from these leaves, and the resulting Merkle Root serves as the commitment to the Web Archive File being stored. When a file encoding is challenged, farmers must provide proofs consisting of the challenged leaf m and its corresponding Merkle Path, enabling recomputation of the root.

Figure 4.5 shows the Merkle Tree data structure and Merkle Root commitment of the file encoding in question.

Periodic Challenge Mechanism. As described earlier, in this PDP protocol, farmers are challenged to prove that they are correctly storing a given leaf m from a Merkle tree constructed from an encoded file. However, as noted in Section 2.6, this only confirms that the farmer holds **that specific block at the time of the challenge**. An adversary could store only a single block or discard several blocks while still passing the test for that encoded file block proof. To address this,

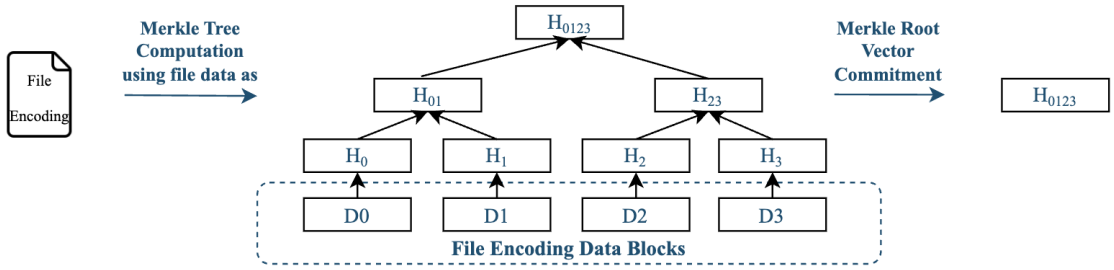

Figure 4.5: Computation of a Merkle root vector commitment from file encoding data (AES/VDE).

we employ a PDP mechanism in which different blocks of the encoded file are periodically and randomly selected for verification through Smart Contracts. Each additional random challenge reduces the probability of successful cheating exponentially, and, combined with the computational difficulty of finding hash collisions in the Merkle tree leaves, enables probabilistic detection of file corruption. This approach ensures both the integrity and long-term availability of stored files.

The storage contract previously submitted to the blockchain includes key metadata necessary for these recurring proofs: the Merkle Root of the file encoding R, file size L, proof frequency F, number of challenged leaves T, challenge window size W, contract value V, and the maximum number of tolerated failed proofs M.

The blockchain serves as a trusted time source, and challenge windows are determined by block heights. Specifically, every F blocks, a new challenge window of size W is initiated (darker blue blockchain blocks shown in Figure 4.6). For instance, with $F = 10$ and $W = 5$, a challenge window starting at block 20 (window start, inclusive) would span blocks 20 through 25 (window end, exclusive). If the farmer submits a valid proof during this window, the next challenge is scheduled to begin at block 35 ($windowend + F$), maintaining a consistent proof periodicity.

Each challenge window requires the farmer to prove possession of specific, T randomly selected leaves (shown in purple in Figure 4.6) from the Merkle Tree that represents the encoded file. This randomness is derived from the blockchain itself: the challenge value for a given window is computed as the hash of the block that immediately precedes the start of the window, i.e., $Challenge_0 = Hash(Block_{h-1})$ (shown in step 2 in Figure 4.6), where h is the window's starting block height. Given this challenge, the farmer computes the index of the target leaf as $leaf = challenge \mod N$, where N is the total number of Merkle leaves, calculated by padding the file size L to the nearest power of two and dividing it by the fixed leaf size S. Since the hash of the preceding block is unpredictable, the leaf selection process is effectively random, preventing the farmer from anticipating or manipulating which part of the encoded file will be challenged, as long as the network is secure via PoUSp consensus. This probabilistically guarantees that to produce correct proofs, farmers must store all leaves (the entire committed vector), from the encoded file, correctly. Similarly to Filecoin, we allow random targeting of multiple file blocks through a recursive PDP protocol. The first challenge is derived from the block immediately preceding the start of the window, while each of the remaining $T - 1$ iterations uses as its challenge the original first challenge combined with the hash of the Merkle proof, obtained in the preceding iteration,

given by the expression $Challenge_T = Hash(Challenge_0 \mid\mid MerkleProof_{T-1})$. Allowing recursive proofs increases the number of file sections covered, thus making the probability of detecting missing or corrupted blocks grow exponentially in T. By using the Hash of the original Challenge combined with the preceding Merkle Proof, we guarantee that the subset T of selected leaves is effectively random across different audits, even if the first target leaf is the same.

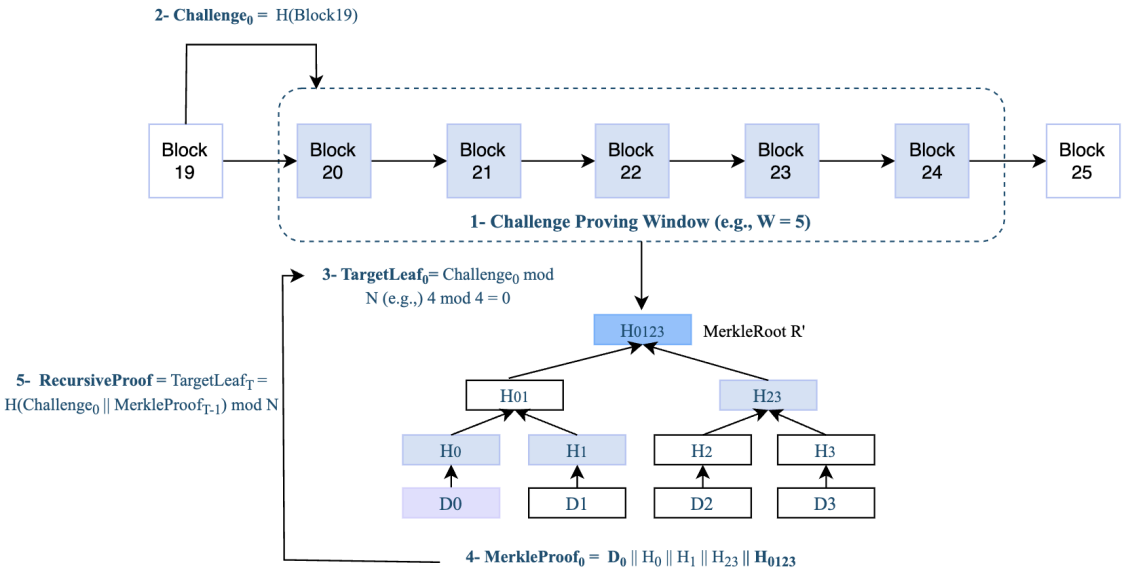

Figure 4.6: Challenge Proving Windows, previous block challenge, and corresponding T recursive Merkle Proofs that target different sections of the encoded file

Proof Submission and Tracking. To submit a valid proof, the farmer generates Merkle proofs for the T selected leaves, consisting of the leaves themselves and their corresponding authentication path that allows the reconstruction of the Merkle Root R. Upon submission, the smart contract verifies that each provided proof corresponds to the correct challenge and that they reconstruct the original Merkle Root R' (darker Blue in Figure 4.6). If the reconstructed roots presented match the one in the archival contract ($R' = R$), it means that the farmer is probabilistically correctly storing the file and the contract rewards the initially agreed reward V. If the farmer fails to respond correctly to M challenge windows, either by missing deadlines or submitting incorrect proofs, the contract is terminated, and the farmer's cryptocurrency is slashed.

Operationally, each farmer maintains a cache of the network active contracts and corresponding proving windows, and monitors the blockchain for the initiation of challenge windows relevant to the files they store. When a relevant window opens, the farmer checks the initial challenge $Hash(Block_{h-1})$ and constructs the Merkle proofs for the corresponding the T selected leaves, and submits it for verification as a blockchain transaction broadcast. If successful, the challenge window is marked as completed, and the next one is automatically scheduled to begin F blocks after the end of the current window. In this way, by leveraging Merkle proofs for compact and efficient verification and the blockchain as a reliable source of both time and randomness, our protocol enables the efficient execution of periodic storage challenges with strong probabilistic

guarantees that the file is being stored in its entirety and remains uncorrupted, and eventually, malicious behavior will be detected with high probability.

AES and VDE encoding security

With respect to encoding types, fixed periodic proofs can only be reliably applied to files encoded using AES, as the encoding is entirely unreproducible by the farmer without access to Web Archive's secret key. In contrast, when using a VDE approach, the situation is more nuanced. If the farmer can predict the start of a proving window, they could exploit this knowledge by fetching the raw file from another source and initiating the VDE process T seconds in advance, where T is the security parameter defining the computational delay of the VDE. This would allow the farmer to recreate the encoded file just in time to respond to the challenge, effectively circumventing the requirement to store the file continuously. To mitigate this vulnerability, our protocol introduces randomness in the scheduling of VDE-based challenges. For such files, challenge windows are not triggered at regular intervals but are instead initiated at randomly selected block heights spaced approximately every $(P \cdot F)$ blocks, where P is a specified unit of VDE proving windows (e.g., 1 Day). This unpredictability ensures that farmers cannot anticipate when proof will be required. Furthermore, to strengthen security, the encoding delay parameter T must be greater than the challenge window size W, thereby guaranteeing that when the network starts a proving window, the farmer will be unable to recompute the VDE encoded file and produce a valid proof before the window closes. This randomized scheduling and time-bound encoding, effectively deters outsourcing attacks and upholds the integrity of the storage verification process.

File Access

When accessing files, clients communicate with a Web Archive proxy, which serves as the initial point of contact for their requests. Clients can access archived files in two ways, depending on the encoding type, AES or VDE.

For AES encoded files, the client requests the file from the Web Archive, which retrieves the encoding from the farmer, decodes it using its Secret Key and the salt Hash(farmerPk + fileId + counter), and forwards the decoded file to the client. This approach requires direct interaction of the Web Archive proxy during decoding, since the secrecy of the secret key must be preserved. Although this encoding type imposes additional work on the Web Archive, modern hardware and software optimizations for AES ensure that the decoding overhead remains minimal while still enabling cost reduction.

For VDE-encoded files, the client requests the Web Archive proxy for file metadata and the identity of a farmer who stores the file. The client then retrieves the VDE-encoded file directly from the farmer and decodes it, using Hash(farmerPk + fileId + counter) as the salt, on the client device. Once decoded, the client verifies the file's integrity by comparing its hash with the original one; if valid, it renders the webpage in its browser. Recall that VDE-decode is fast compared to VDE-encode, so the client will not be affected by page render times. As can be deduced, this

approach offloads most of the workload from the Web Archive, which only handles the retrieval of file metadata and farmer identity. The actual encoded file is obtained directly by the user from the farmer, who also performs the decoding and validation. This approach achieves a greater cost reduction compared to the AES-based approach.

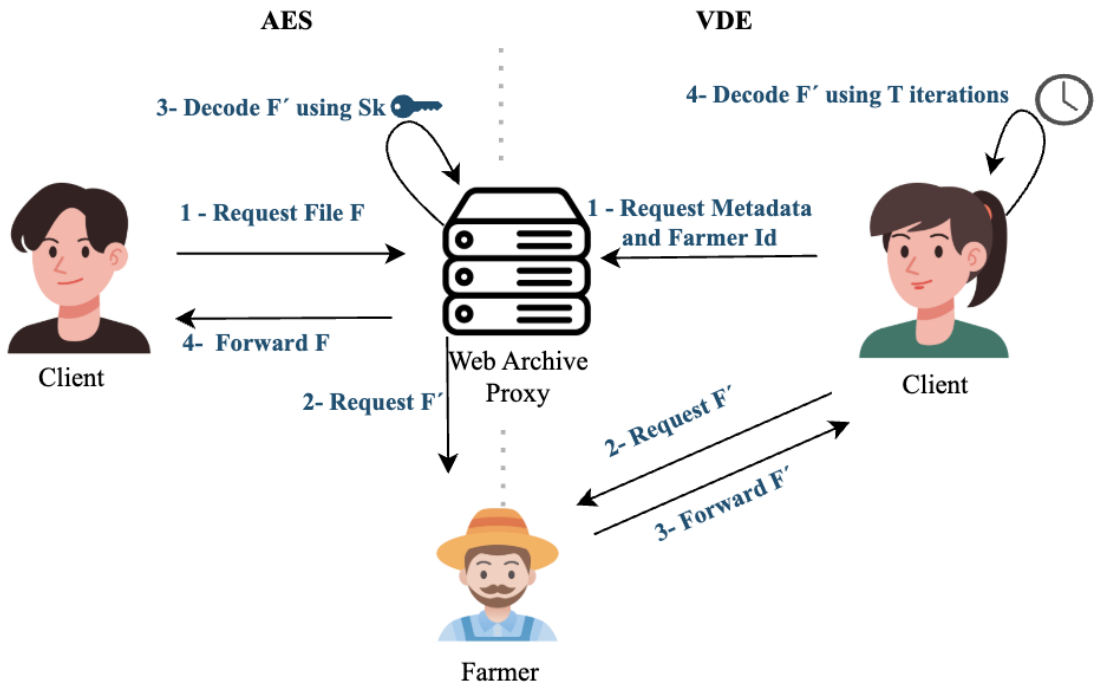

Figure 4.7: Comparison of file access under the two encoding approaches: AES (left) and VDE (right).

The file access process also serves as a random PDP challenge. For AES-encoded files, if the farmer provides an incorrect or missing encoding, the Web Archive proxy detects it. For VDE-encoded files, the client directly detects such failures. To prevent outsourcing attacks, where a farmer might attempt to fetch the file from another source and re-encode it, clients enforce a response time threshold where the decoding time must be less than the VDE encoding delay T. If the farmer's response exceeds this threshold, it suggests that the farmer is attempting to re-encode the file, indicating malicious behavior. The client can then report this to the network, ensuring the integrity of the storage system through this time-bound verification.

Replication and Load Balancing

When outsourcing Web Archive files to farmers on a blockchain, we transition from a reliable, professionally managed infrastructure to a decentralized network of potentially unreliable workers. These workers may exhibit various forms of misbehavior, such as file corruption, data loss, downtime, or refusal to serve files. To mitigate the risk of farmer failure, redundancy is commonly employed. A standard approach involves erasure coding, where data is divided into k data fragments with an additional m parity fragments, resulting in a total of $k + m$ stored fragments. Any k of these fragments can reconstruct the original data. However, this method is suboptimal

for our use case, as accessing a file requires retrieving k fragments, either by the Web Archive or the client. This increases communication overhead, negatively impacting latency, throughput, and ultimately leading to slower page rendering times.

Instead, we adopt a partial replication mechanism to achieve high availability, faster access times and geographical replication. In this approach, a file is entirely replicated across k farmers, enabling data retrieval with a single round of communication, as no data reconstruction is necessary. This results in improved latency and throughput, as well as simpler replication and access protocols with minimal computational overhead. Partial replication tolerates up to $k - 1$ faults, ensuring data recovery as long as at least one replica remains accessible. However, this comes at the expense of increased storage overhead, as the total storage requirement is $k \cdot \text{FileSize}$.

By employing a partial replication protocol for redundancy, we can also implement load-balancing strategies, allowing client requests for an archived page to be evenly distributed across the k farmers replicating a file. This prevents overloading of individual farmers, distributes the workload among them, improves efficiency and performance, and enables traffic to be redirected to other farmers in case of a failure.

Probability of detecting failures in PDP

Our PDP protocol guarantees file integrity and storage over time probabilistically, meaning that there is a high probability that farmers who discard pieces of a file encoding will eventually be detected.

The probability of detection is given by the expression:

$$P_{\text{detect},K} = 1 - \left(\frac{N-F}{N}\right)^{TK},$$

where N is the total number of blocks in a file, F is the number of missing or corrupted blocks, T is the number of challenged leaves per audit, and K is the number of audits. We observe that the probability of detection increases exponentially with both the number of audits and the number of challenged leaves.

Figure 4.8 illustrates the effect of different values of F (percentage of file loss or corruption) and T on the detection probability across K audits. Eventually, this probability reaches one (guaranteed detection), however, careful parameterization is required to choose an appropriate value of T. For instance, using $T = 10$, as in Filecoin, yields certain detection much quicker when compared to other values of T. Moreover, the benefit of discarding a small number of leaves is minimal, even if some leaves are not directly challenged, they are necessary to reconstruct the Merkle Path for proofs. This implies that intermediate nodes need to be stored when leaves are discarded, thus the benefit of discarding leaves to save cost of storage is really small. This small benefit of discarding leaves will be amplified by slashing penalties, which will be discussed later.

Additionally, there exists a type of challenge that always detects failures and is randomly applied, file access requests. When a file access request occurs, the farmer must provide the complete, correct file, otherwise, the failure is immediately detected and penalized.

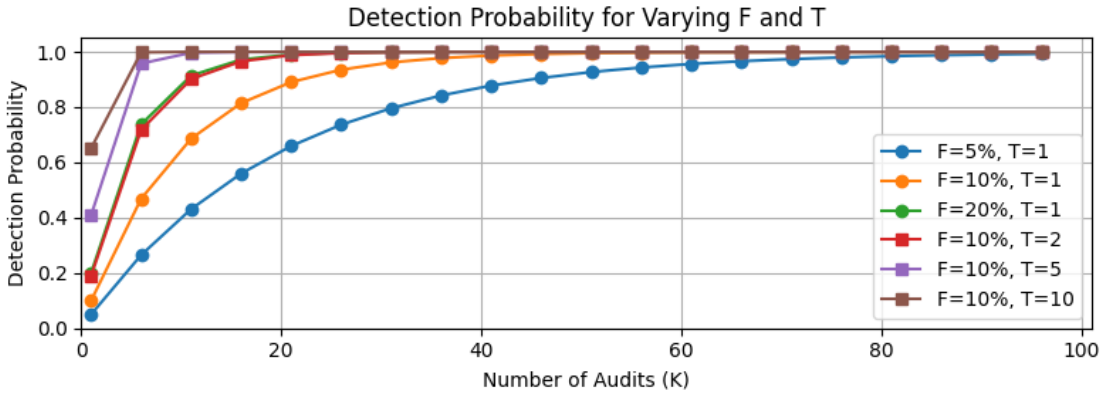

Figure 4.8: Probability of detection for different values of F and T

4.5 Incentive Model

An important aspect of this protocol is the incentive model, as it is responsible for promoting honest behavior and incentivize participation.

There are two categories of incentives and penalties to consider. The first is associated with the PoUSp protocol and blockchain maintenance, while the second relates to PDP, where farmers must be motivated to correctly store Web Archive files.

In the PoUSp protocol, farmers are rewarded through block rewards for producing winning blocks that extend the blockchain by dedicating space to the network consensus, as well as transaction fees for storing and processing transactions. Higher transaction fees grant transactions greater priority, incentivizing farmers to process them first. Consequently, transactions with no associated fees may never be processed, and storage contract uploads or modifications come with a small fee to not be left out by farmers. Following the Chia model, the initial block reward is set at 2 ArchiveCoins, with a halving every 3 years for a total of four halvings (given a block time of roughly 19 seconds). The halving block index is calculated as:

$$32 \times 6 \times 24 \times 365 \times 3 \times x$$

where x is the halving index (maximum of 4). After the fourth halving, the reward remains constant. The reward schedule is shown in Table 4.1.

Block Index	Coin Reward
0	2
5045760	1
10091520	0.5
15137280	0.25
20183040	0.125

Table 4.1: ArchiveCoin block reward schedule by block index.

The PoUSp model imposes no direct penalties, but it relies on strong disincentives. In Bitcoin, such disincentives come from the high computational and electricity costs required to control the

network, making attacks economically unfeasible. Similarly, in PoUSp, the deterrent lies in the substantial initial cost and effort needed to create plots, the hardware and storage investments to maintain them, and the PoT mechanism, which enforces sequential block generation and prevents shortcuts, making attacks both economically and temporally impractical.

In PDP, incentives arise from providing proofs during audits. Farmers receive an agreed reward V from the Web Archive for successful PDP proofs. The value of V is market-driven, as the Web Archive selects farmers that offer the best storage deals. Reputation tracking further allows balancing value and reliability to optimize contracts. Farmers using VDE incur on higher initialization costs and offload the most work from the Web Archive, thus they are naturally incentivized to charge more for storage. However, rewards alone are insufficient to deter malicious behavior, a farmer could discard/corrupt file segments yet still earn rewards until detected. To address this, we introduce a slashing mechanism, where if a farmer fails an audit, a penalty of $200 \times V$ is applied, equating to the loss of 200 audit rewards. Figure 4.8 shows that detection probability at the 80th audit approaches 1 across all tested parameters, making the penalty far greater than any benefit gained from discarding small amounts of data, along with the need to store intermediate nodes of the Merkle Tree. Furthermore, discarding more data increases both the benefit of reducing storage costs to the farmer but increases detection probability, amplifying the risk.

File accesses act as full-file challenges, where any missing data results in immediate detection. Since farmers cannot predict when such accesses will occur, improper behavior can lead to slashing at any moment. This unpredictability, combined with the penalty, strongly incentivizes farmers to correctly store and serve Web Archive files.

4.6 PoUSp and PDP parameters

Our protocols have several parameters that need to be carefully tuned, as they impact performance and security.

These are the PoUSp public parameters that are shared across network participants:

- K, the maximum number of blocks to be worked at the a certain height. (affects the resource honest majority threshold)
- SLOTH p, prime number correspondent for a certain amount of security bits (affects the useful space ratio).
- SLOTH t, the number of iterations that can be mapped to real-clock time (affects the amount of computation required during the plotting phase, making the process more time-consuming and difficulting Hellman time–memory trade-offs).
- Merkle Tree Chunk Size M (affects the number of SLOTH proofs generated per initialization time, the fewer the proofs, the more difficult it is to perform Hellman time–memory trade-offs, the useful space ratio and proof size).

- Merkle Tree leaf Size S. (affects useful space ratio)
- Merkle tree leaf count N. (affects useful space ratio)
- Merkle Tree fanout f. (affects useful space ratio)
- Merkle Tree hash function.

These parameters should be carefully tuned b because they affect different aspects of the protocol. For example, regarding initialization time, longer initialization per proof increases the blockchain's resistance to Hellman attacks; however, if it is too high, it may discourage participants from joining the network. Similarly, for the useful space ratio, larger chunk sizes improve the ratio and reduce the number of SLOTH proofs, but at the cost of larger proof sizes. Therefore, a careful balance must be struck between initialization time, useful space ratio, proof size, and security. Evaluation of changes in these parameters will be shown in Section 6.1.

Although there are no public parameters fixed for everyone in the PDP protocol, there are still storage contract parameters that need to be taken into account:

- Proof frequency F.
- Number of challenged leaves T.
- Challenge window size W.
- Contract value V.
- Maximum number of tolerated failed proofs M.
- VDE/AES size choice $threshold$.
- VDE time t

Although not public parameters, the values discussed above significantly affect the probabilistic PDP detection. For instance, if F is too low, failures may take a long time to be detected if the file is not requested in the meantime; on the other hand, if F is too high, farmers perform excessive work, potentially flooding the network with challenge window proofs transactions. Similarly, if W is too small, PDP proofs may not be included in a block in time; if it is too large, farmers could exploit the window to perform outsourcing attacks or submit fake proofs. In alignment with F, increasing T raises the probability of earlier failure detection, whereas decreasing it reduces this probability. The parameter V directly influences farmer participation in file outsourcing; too low, and farmers may be discouraged from joining the network; too high, and the value of the currency may be adversely affected.

A particularly challenging aspect is tuning the encoding time t of VDE. It must exceed the challenge window size W, which itself must be sufficiently large for proof generation and block inclusion. For smaller files, increasing t to be larger than W can impose excessive initialization

overhead on top of the PoUSp plotting phase, potentially limiting the amount of dedicated storage for the network and discouraging adoption. Consequently, we recommend using VDE primarily for larger files. As noted in Section 4.4.2, file size sets a lower bound on VDE computation even when only a single iteration per VDE CBC block is used. Using VDE on larger files ensures that encoding time exceeds W, while also maximizing storage utilization.

For example, for a threshold of 150 MB, encoding takes roughly 1 minute ($W = 5$, with a block time of ≈ 20 seconds), and larger files further improve VDE security. On top of PoUSp, PDP VDE initialization approximately doubles initialization times, so a higher value of V should be considered accordingly.

In Section 4.4.2, we analyze how T influences the probability of detecting storage failures as the number of audits increases, considering various percentages of file loss or corruption.

4.7 Practical Considerations

In this section, we address practical considerations necessary for the stable and reliable operation of our construction. These include strategies to avoid flooding the network by batching files within the same storage contract, handling data modification, and approaches to efficient utilize disk storage.

4.7.1 File Batching

As mentioned, throughout our work, web archives capture snapshots of web content and store them for future access, preserving the state of the content at the time of capture. However, due to the nature of web content, files are often small in size but numerous, resulting in a high volume of small files. For our construction, this poses a challenge. From 2012 to 2025, average page sizes range from 800 KB to 2.5 MB. As a result , Storing petabytes of data using individual file contracts would require creating millions of storage contracts, overwhelming the network with excessive contract creation, proof windows, and all associated transactions. This would hinder transaction processing, as PDP operates at the transaction layer (similar issue is taken into account by Filecoin).

Our proposed consideration is file batching within a single storage contract in the PDP protocol. This approach allows a large number of encoded files to be grouped under the same contract, with the contract's Merkle Root constructed from all encoded data concatenated into leaves. Individual file encodings are preserved, enabling access to single files upon user request. This would not be possible if multiple files were encoded together, as retrieving any individual file would require decoding the entire batch.

In this way, PDP proofs can cover large portions of data composed of multiple pages without necessitating full retrieval, maintaining storage and respective proof generation efficiency. The network does not get flooded with numerous small storage contracts, instead managing fewer contracts that each represent a large amount of data while preserving efficient retrieval and proof verification.

4.7.2 Data Modification

Web Archive data is particularly well-suited for our system because it consists of snapshots of web content captured over time. Each snapshot represents the state of the content at the moment of capture, making it highly unlikely that the data is subsequently modified. This characteristic aligns with a Write Once, Read Many (WORM) storage model. Consequently, once data is initialized in PoUSp and PDP, it typically remains unchanged throughout the lifetime of the blockchain.

Despite having a WORM storage model, our protocol still allows data modifications by the Web Archive. When such modifications occur, the PDP and PoUSp commitments are invalidated. As a result, the initialization processes of PoUSp (plotting) and PDP (archival phase) must be re-executed. Following the CREATE, READ, UPDATE, DELETE (CRUD) paradigm, UPDATE and DELETE PDP transactions are required to reflect these changes. After re-initialization, the modified data can be used in the blockchain consensus, with its storage and integrity verifiable over time. It is important to note that previous PoUSp and PDP proofs already recorded on the blockchain remain valid. Nevertheless, file updates or removals are expected to be infrequent.

4.7.3 PoUSp and PDP Data Heterogeneity and Storage Utilization

Our PoUSp consensus protocol allows farmers to encode any useful data of their choice within the cryptographic proofs, while leveraging a PDP protocol to provide verifiable storage for outsourced files. As discussed in Section 4.4.1, it is possible to combine PoUSp and PDP into a single protocol. However, doing so would require sacrificing some properties of Chia and Chia Bread Pudding, most notably non-interactiveness, as storage commitments would need to be submitted to the blockchain prior to participation. Therefore, we chose to keep the protocols separate, allowing them to complement each other without compromising their individual features.

Maintaining separate protocols also enables heterogeneity in the data used by PoUSp and PDP. Farmers can continue to leverage any data they find useful (whether Web Archive data, personal files, or random data), ensuring full freedom of choice. Granting farmers control over the data used for consensus further incentivizes participation, regardless of how their resources are allocated, thereby promoting greater decentralization. For example, even if a user's disk space is fully occupied with their own data and they are unable to store Web Archive data, they can still participate in the consensus process.

Moreover, farmers may not immediately be able to fill their storage with Web Archive or personal data. To maximize their chances of winning PoUSp proofs by dedicating all their available disk space to the network, they can utilize the remaining free space with random data and run the PoUSp plotting phase to integrate it into consensus. For instance, a farmer with 1TB of disk space could allocate 250GB to Web Archive data, 250GB to personal data, and use the remaining 500GB for random data, thereby fully utilizing the 1TB for consensus rather than only 500GB. Over time, as the farmer accumulates more useful data, the random data can be progressively replaced, requiring only a new plotting phase for the newly added data.

This design provides flexibility for farmers to optimize disk usage, benefiting both individual

participants and the blockchain by having more space allocated and participants who will increase security and decentralization.

Keeping the protocols heterogeneous also allows them to evolve independently and modularly, and, for example, using them separately, as PoUSp can be employed without PDP and vice-versa.

Chapter 5

Implementation

In this chapter, we discuss implementation aspects that, although not part of the PoUSp and PDP protocols, were essential to develop a working prototype. These include the signature scheme and key generation and management, the dissemination protocol that ensures messages reach most nodes in the network, and, finally, general details concerning the programming language, frameworks, and developed components.

5.1 Signature scheme and Key generation

In our construction, we leverage Asymmetric Encryption to identify participants in the system and provide them with ownership and control over their funds, transactions, and other assets. This is achieved through digital signatures, where each transaction has an associated signature, generated using a private key, while the corresponding public key generates an address that serves as the participant's identity within the network and validates the signatures. Digital signatures provide authenticity and non-repudiation. If a signature is valid for a given public key, the participant possessing the corresponding private key has effectively signed it and cannot deny it nor their identity. Additionally, digital signatures ensure integrity, meaning that if a transaction is tampered with, the signature becomes invalid. This is critical for maintaining the immutability and integrity of the blockchain. Since trust in the blockchain is decentralized, trust in the cryptographic mechanisms is essential for establishing ownership, authenticity, and integrity. Each asset in our system (cryptocurrency, file storage contracts, transactions, etc.) is associated with a wallet responsible for managing it. The wallet is derived from the public key through a double-hashing process.

Our chosen asymmetric encryption scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA), primarily because, for the same level of security, ECDSA public keys are significantly smaller than RSA, signatures are more compact, and the protocol is computationally more efficient. For example, at a 128-bit security level, ECDSA keys have a 32-byte private key and a 64-byte public key, while RSA keys require approximately 384 bytes. Signature sizes for ECDSA are 64 bytes compared to RSA's 384 bytes, using our selected curve *secp256k1*. These characteristics make ECDSA a common choice in blockchain systems.

A key consideration is that private and public keys are large numbers, which can make storage

and management challenging. For instance, an uncompressed public key:

$$0x00b1a572379ef091d48c555a454bdf708ec4ace6a66256ee5466b682e0b4ef54ca566eea64...$$

and a private key:

$$0x7e72bc1a16a3968f606238e70d763b8a9a89692bfff4fdbb0ac054e898c16e878$$

are extremely difficult to remember. Loss of these keys could result in permanent loss of access to assets. To improve usability, we use a mnemonic system that encodes random seeds (which generate public/private key pairs) into a human-readable 12-word mnemonic phrase. Specifically, we employ BIP-39, which uses a word list of 2048 words, where each word encodes an 11-bit segment of the seed. This seed, combined with a passphrase for added security, is input into our key derivation function PBKDF2 with HMAC and SHA256 to produce the ECDSA key pair.

For example, the previously mentioned key pair can be represented by the mnemonic phrase: *"manage gas obscure update tag exotic gun climb match sentence hair faculty"* and the passphrase *"user123"*. This approach makes key generation and management both secure and user-friendly. The wallet address, derived from the public key, for this example is:

$$0xa38e0bcf6ce20f0ac7fb1bf4652694298241cdabce6584ce382c6b6c64e667f9$$

and all assets and transactions are directly linked to this address, providing a clear and secure association between participants and their holdings.

5.2 Message dissemination

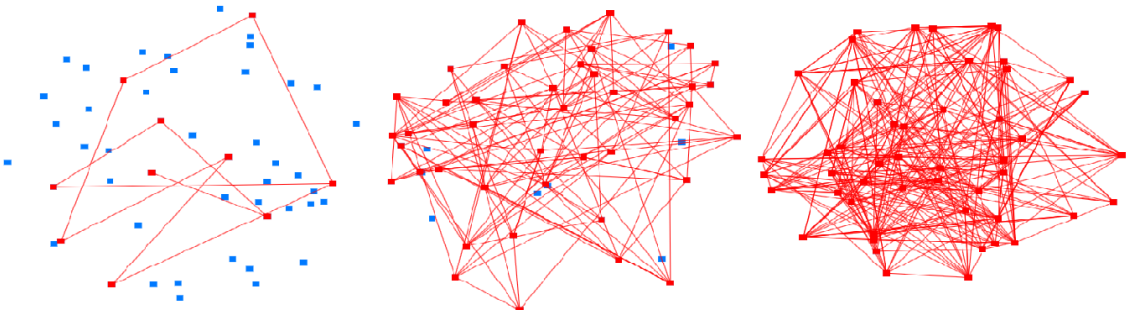

Figure 5.1: Simulation of gossip-based message dissemination in a network of 50 nodes. Gossip simulation on 50 nodes with broadcast degrees of 1, 3, and $\ln(50) \approx 4$.

Being a permissionless blockchain, our system's network overlay follows an unstructured P2P design, where nodes establish links with random neighbors, and join the network through initial points of contact called seed nodes. Each node broadcasts a peer discovery message, allowing existing peers to learn about the newcomer while allowing the new node to discover potential neighbors. A key aspect of this design is message dissemination where messages must be broadcast across the network so that, ideally, every node eventually receives them. The network is

fully decentralized, nodes know neighbors at random, and no global view of the topology is required. To achieve this, we implement an eager-push gossip/epidemic dissemination algorithm, where nodes forward messages to their neighbors and a random subset of peers. According to epidemic theory, a subset size of $\ln(n)$, where n is the total number of nodes, is sufficient to ensure probabilistic delivery. Following Bitcoin's approach, we allow up to 125 connections per node, which is enough to scale to networks of millions of peers. With gossip, when a node broadcasts to its subset, receivers continue the process, and epidemic theory ensures eventual coverage. This allows efficient broadcasting with lower overhead compared to flooding, is simple to implement since only neighbor knowledge is required, and is both fast and probabilistically reliable. However, because delivery is not guaranteed, consensus and state synchronization remain necessary. In Figure 5.1, we utilized a Gossip Simulator [28] to evaluate how well epidemic theory applies in practice. For a network of 50 nodes, we tested three scenarios: (i) nodes broadcast the message to only one neighbor, (ii) nodes broadcast to three neighbors, and (iii) nodes broadcast to $\ln(50)$ neighbors, which is rounded to four. As shown, when nodes broadcast to only one neighbor, many nodes never received the message. With three neighbors, almost all nodes were reached, but a few still missed the message. Finally, with the number suggested by the epidemic theory, every node received it successfully. To further increase the probability of full dissemination, a constant C can be added, yielding the expression $\ln(n) + C$.

5.3 Languages, Frameworks and Components

For the implementation of the blockchain, we selected Java 21 with SpringBoot as the main framework, enabling rapid development of scalable backend applications. Its modular design, supported by dependency injection, allows flexible system composition. To connect nodes, we exposed a REST API, with messages exchanged in JSON format through the SpringBoot networking components. The choice of Java 21 was motivated mainly by the introduction of virtual threads (Project Loom), which are well suited for highly I/O-intensive tasks such as those in a blockchain, where massive communication and concurrent request handling are required. However, this applies only to networking and message processing. For CPU-bound tasks, such as SLOTH and Wesolowski's VDF, the implementations were developed in C/C++ and invoked from Java through the Java Native Access/Interface mechanism.

To ensure data persistence on disk, we employed MapDB [49], an embedded database and collections engine backed by disk and/or memory. It provides a lightweight alternative to traditional SQL databases and is well suited to implement efficient key–value stores required to store our data model.

Our coin model of choice was Bitcoin's Unspent Transaction Output (UTXO), which simplifies double-spending prevention and makes it both transparent and verifiable. The UTXO model also enables parallel spending, where different coins can be used simultaneously without conflicts, and avoids the overhead of tracking spends and versioning transactions, as required in Ethereum. Additionally, nodes manage transactions through a local memory pool (mempool), from which

miners select entries for inclusion in the next block. This mechanism also allows farmers to maintain the transaction state while preventing data loss, since orphaned transactions (belonging to blocks not integrated into the chain) can be reinserted into the mempool.

We also implemented the Web Archive Proxy using the same technology stack as in the blockchain nodes. As detailed in Section 6.1, Arquivo.pt was used as our archival data source. In our archival process, we first accessed the Arquivo.pt CDX API, which provides indexes containing metadata about page URLs and archived snapshots. Subsequently, we leveraged the noFrame API, available at *https://arquivo.pt/noFrame/replay*, to retrieve the raw content corresponding to the CDX indexes. In the final stage, the archived pages were distributed across the blockchain nodes, followed by the experiments described later.

Finally, we developed a blockchain web explorer to support monitoring, data retrieval, and rendering of archived web pages. The explorer enables inspection of every system component, including PDP and PoUSp proofs, transactions, contracts, and the overall system state. It also provides visibility into file replication across farmers and details of farmer wallets. Unlike the other two components, this system was developed using Angular 19, a framework well suited for building interactive web applications. Our project is available here: **`https://archivechain.pt`**. Figures 5.2 illustrate the main exlporer page where the user can see general blockchain metrics, the blockchain history of blocks and corresponding files who generated the winning nonce as weel as the most recent transactions. Figures 5.3 illustrate a page rendered using ArchiveChain, remembering that it is being stored by a blockchain farmer andit was retrieved directly from him.

Figures 5.2 illustrate the main explorer page, where the user can view general blockchain metrics, the block history and corresponding files that generated the winning nonces, as well as the most recent transactions. Figures 5.3 show a page rendered using ArchiveChain, which is being replicated accross blockchain farmers and retrieved directly from them.

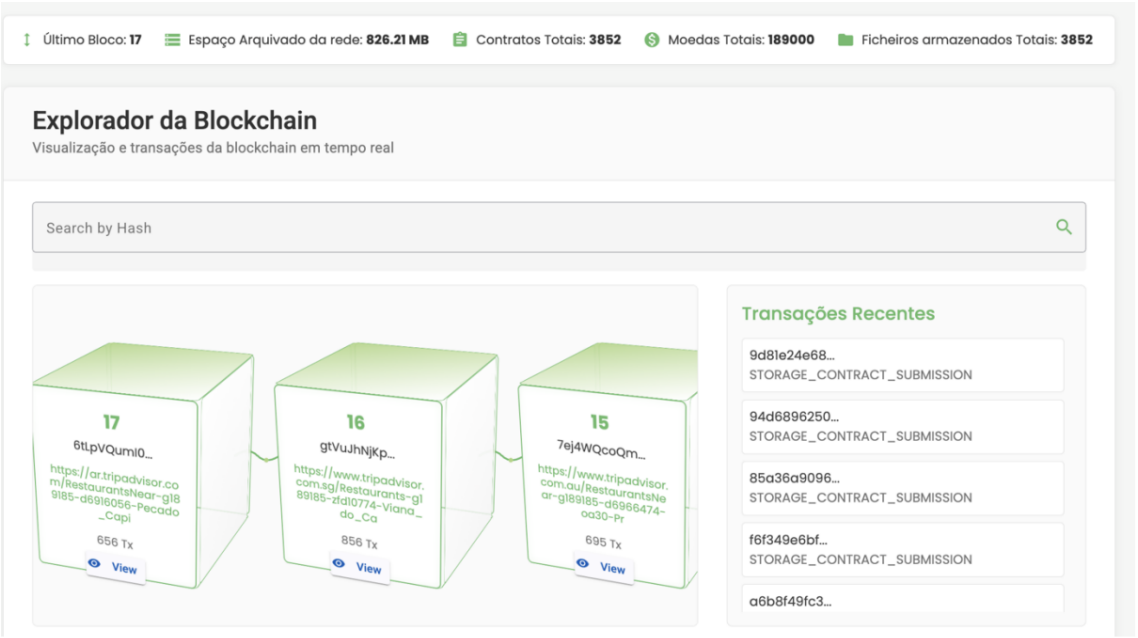

Figure 5.2: Recent Blockchain State Visualization

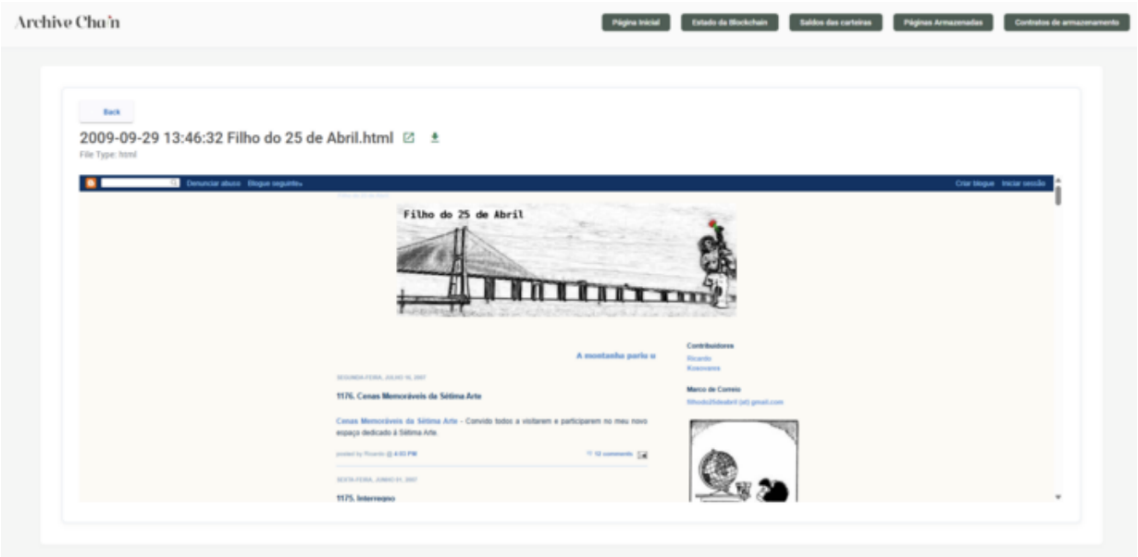

Figure 5.3: Outsourced Paged rendered directly from the farmer

Chapter 6

Evaluation

In this chapter, we present the experimental evaluation of the ArchiveChain blockchain. Our analysis focuses on several key performance metrics. First, we examine the Useful Space Ratio and the proof size of PoUSp under different data chunk sizes. Next, we evaluate how PDP proof size scales with increasing file sizes. We then verify that the probability of winning a PoUSp block is proportional to the disk space allocated by a farmer. Additionally, we compare the initialization times of ArchiveChain with those of the Chia blockchain. Finally, we assess system throughput and latency to determine whether ArchiveChain can maintain a comparable Quality of Service (QoS) to existing centralized web archive solutions.

6.1 Experimental Setup

To help scale Web Archives, we focus on Arquivo.pt, managed by FCCN [2], as our data source and test subject. Arquivo.pt currently stores 1 Petabyte (PB) of compressed data, including websites, images, videos, etc., across 77 servers with 2,180 vCPUs, 18 TB of RAM, and 1,234 hard drives (5.18 PB) [3].

To accurately simulate the node setup for general users, we will conduct experiments on an Apple MacBook Air equipped with an M1 ARM processor (4 performance cores and 4 efficiency cores), macOS Sequoia 15.6.1, 16 GB of RAM, a 256 GB SSD, connected via WiFi 5 and running 4 full nodes, 1 Web Archive Proxy, and 1 Frontend Server. To mitigate the impact of network latency due to distance, our monitoring machine (located in Lisbon) was configured with a VPN endpoint in Porto, ensuring it has approximately equal network proximity to both our Host Nodes (Lisbon) and FCCN (Lisbon).

The PoUSp Public parameters were selected as follows:

- K = 3 blocks maximum, to be worked at each depth
- SLOTH $p = 2^{256} - 189$ prime with 256 security bits
- SLOTH t = 14 iterations $\approx$ 0.404 ms, which matches Chia plotting computational difficulty.
- Merkle Tree Chunk Size M = 2048 bytes

- Merkle Tree leaf Size S = 32 bytes
- Merkle tree leaf count $N = \frac{M}{S}$ = 64 leaves
- Merkle Tree fanout f = 2, Binary tree
- Merkle Tree hash function = SHA3-256

These parameters were chosen to match the current Chia Blockchain parameters related to the amount of PoUSp proofs generated for a specific amount of disk space that should consume a specific amount of time. All the parameters are fixed except M that in the PoUSp useful space ratio and proof size will be changed according to the test.

6.2 Reward Fairness

An important property of the PoUSp protocol is that the probability of winning a block should be proportional to the amount of storage space allocated. This ensures fairness in incentives and upholds network security, as the allocated storage directly represents each node's voting power. To evaluate reward fairness, we will measure the proportion of blocks won by each node relative to its share of the total network storage, with allocations centered around powers of two, which causes expected proportions to roughly double at each step, minimizing the impact of small variations across various initial space allocations.

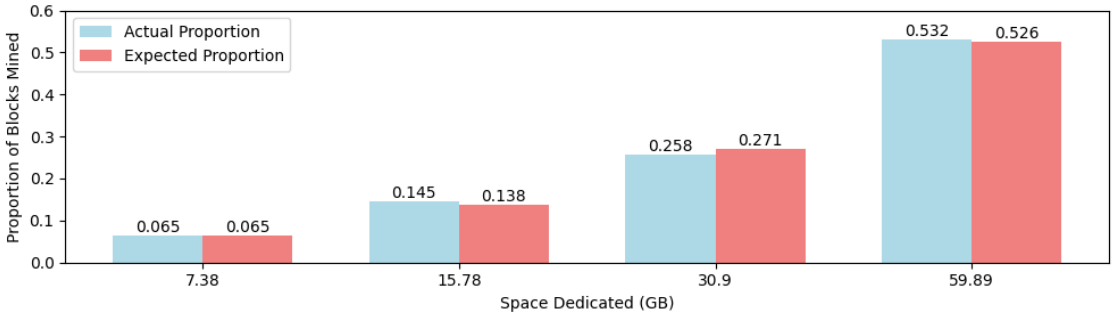

Figure 6.1: Proportion of Blocks Mined vs. Expected (Total Blocks: 3000)

Figure 6.1 shows a clear relationship between the number of blocks mined and the storage space allocated relative to the total system's capacity. The slight deviations observed can be attributed to random statistical fluctuations due to the limited sample size. As the number of blocks grows, the actual proportions will converge towards the expected values, reinforcing the fairness of the system. This fairness is supported by the quality function outlined in Section 4.3.3 and the fact that, the more storage space a farmer dedicates to the network, the greater the number of SLOTH nonces they must generate, increasing their chances of participating in the consensus. In other words, farmers are incentivized to allocate more data to the network, which directly leads to producing more nonces eligible for consensus participation. In addition, this experiment proves that we maintained the entropy of the proofs, guaranteeing they are uniformly distributed.

6.3 PoUSp Plotting Time

We now compare the initialization time of our space allocation to Chia's, using various Chia plot sizes and their corresponding initialization times as benchmarks. Our initialization time, which can be adjusted via the SLOTH T parameter, is adjusted to align with Chia's performance for a fair comparison.

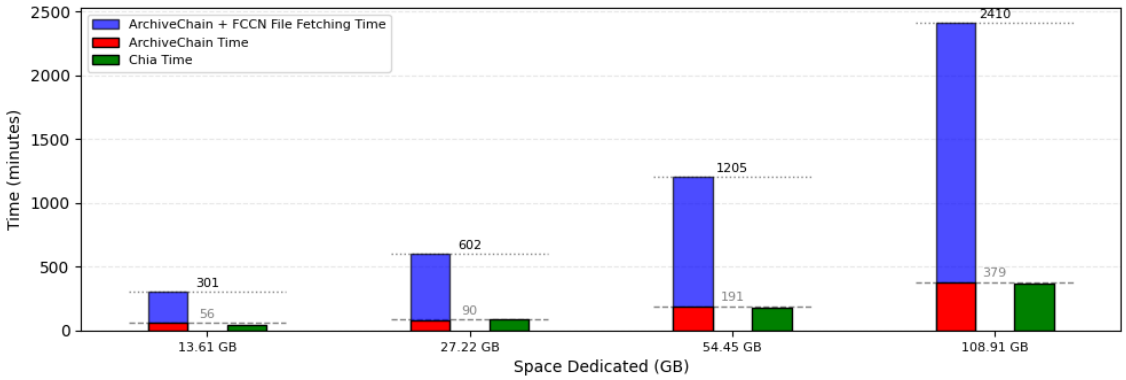

Figure 6.2: Initialization times for different dedicated spaces of Chia (Green), ArchiveChain local time (Red), and ArchiveChain including the file downloading time from Arquivo.pt (Blue)

Figure 6.2 illustrates that ArchiveChain's local initialization times are comparable to those of the Chia protocol, with minimal variations. However, the total initialization time is significantly influenced by the file download from the Web Archive, which introduces extra time due to network transfer latency and bandwidth limitations that are absent in purely local initialization. This initial cost, amortized over time, occurs only once when data is first stored, thus, despite added download times, it does not affect the blockchain's security or performance. Additionally, with potential optimizations and coordination between ArchiveChain and the Web Archives infrastructure, these file download times could be substantially reduced. Nonetheless, this process shows that the protocol initialization is time-consuming, deterring Hellman Attacks, and ensuring correct data and proof storage in disk continuously.

6.4 PoUSp Useful space ratio and Proof Size

In this experiment, we evaluate the useful space ratio achieved by our improvement of the SLOTH encoding, focused solely on the root of the Merkle Tree for each data chunk during the PoUSp plotting phase. We will analyze how the useful space ratio increases as the size of the data chunk grows.

In Figure 6.3, we observe that as the data chunk size N increases, the useful space ratio U also increases, approaching $\lim_{N\to\infty} U = 100\%$. This trend arises because the SLOTH proof size, applied only to the root of the tree, remains constant at 32 bytes, so as the amount of data represented by each tree grows, the SLOTH proof becomes proportionally smaller relative to the total space commitment. The useful space ratio can be further optimized by adjusting the tree's fanout parameter, as demonstrated in Chia Bread Pudding. However, this increases the proof

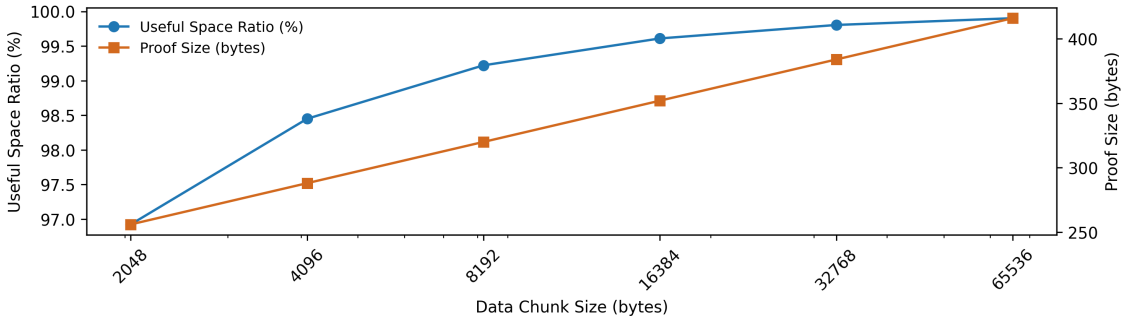

Figure 6.3: Useful Space Ratio and Proof Size vs Data Chunk Size

size, so parameterization requires careful tuning to balance these trade-offs. Nonetheless, our approach achieved a minimum of $\approx 97\%$ useful space ratio, showing significant improvements over Chia's non-useful space, with Chia Bread Pudding achieving $\approx 50\%$ useful space ratio. Most importantly, with the selected parameters, our proof size maintained the original Chia blockchain 256-byte proof size.

6.5 PDP Proof Size

The PDP protocol operates at the transaction layer. Consequently, it is crucial that the size of PDP proofs scales efficiently with the amount of stored data, otherwise, blocks may fill rapidly, degrading transaction throughput and rendering the overall system obsolete. In this experiment, we evaluate the effectiveness of the PDP proof protocol in scaling with increasing amounts of committed data.

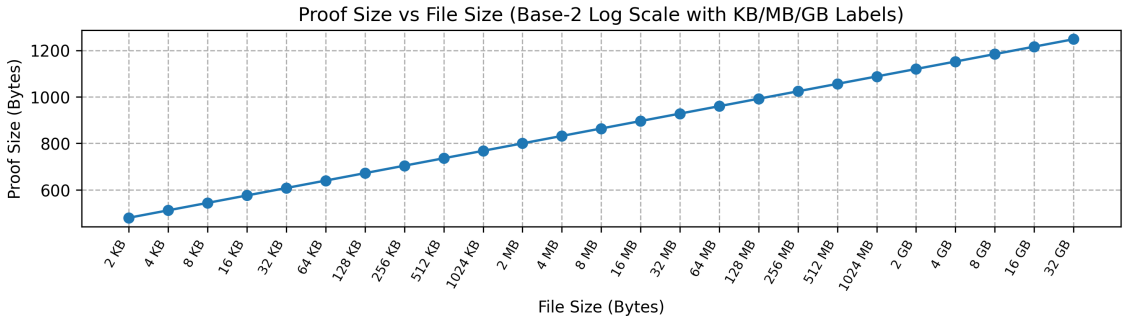

Figure 6.4: Scaling of PDP proof size with increasing file sizes from 2 KB up to 32 GB on a base-2 logarithmic scale.

Figure 6.4 shows that while the file size grows exponentially in powers of two, the proof size increases only logarithmically. Specifically, each doubling of the file size adds one level to the height of the Merkle Tree. However, the corresponding Merkle Proof requires only a single additional 32-byte hash to represent this extra level. As a result, for every power-of-two increase in file size, the proof size grows by just 32 bytes. For instance, as illustrated in Figure 6.4, a 32 GB data block stored on a farmer via PDP, produces a Merkle Proof of 1248 bytes for a single challenged leaf, demonstrating the efficiency of the protocol.

6.6 Quality of Service

To be useful, ArchiveChain must preserve the end-user experience, namely, the latency of accessing archived web pages. In this experiment, we measure the time it takes for a client to receive both the first and last byte of a file, which corresponds to when the page starts and finishes loading, respectively. This procedure was executed for 3674 web pages.

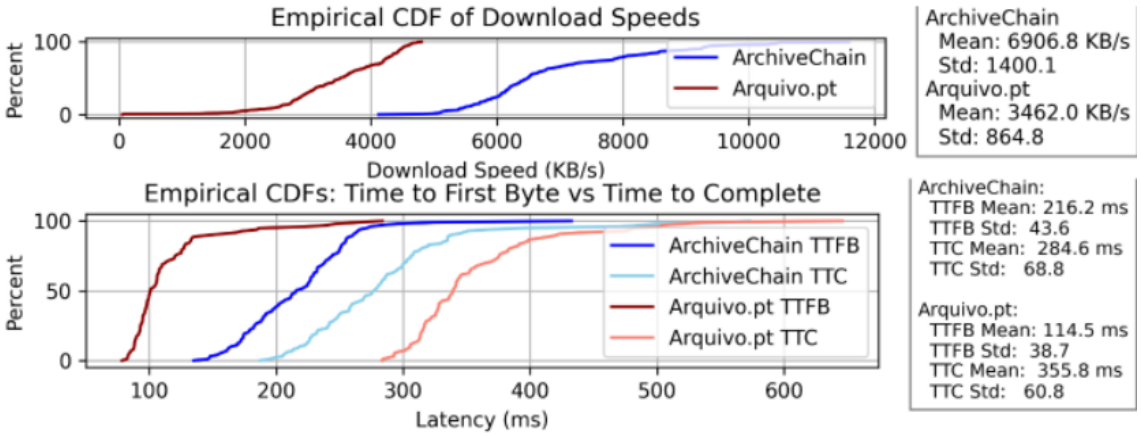

Figure 6.5: Cumulative Distribution Function (CDF) of file download speed and CDF of time to first and last byte

In Figure 6.5, we observe that the Time to First Byte (TTFB) for Arquivo.pt is approximately half that of our system. This difference can be attributed to an additional round of communication in our architecture: we first contact Arquivo.pt to obtain the farmer who holds the file, and only then request the file itself from the farmer. However, despite the higher TTFB, our system achieves a better Time to Complete (TTC). This means that the end client renders the page faster using our system. This can be explained by the CDF of download speeds, which shows that our system delivers roughly twice the download speed of Arquivo.pt, resulting from a simple retrieval that does not require decompression. As a result, even though the first byte arrives later, the overall request completes more quickly, showing that a decentralized approach can also bring performance benefits.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

This thesis addresses two main challenges: scaling web archives such as Arquivo.pt and the Wayback Machine, which face increasing storage and operational costs due to exponential data growth over the years; and improving the resource inefficiencies present in state-of-the-art blockchains. To tackle these, we introduce *ArchiveChain*, a permissionless blockchain that leverages web archival data as a non-forgeable resource for a resource-efficient consensus mechanism, enabling secure and verifiable data outsourcing. This is achieved by combining the PoUSp and PoT schemes: PoUSp allows proving that a certain amount of space was dedicated to the network while embedding web archive data within consensus proofs, and PoT mitigates long-range and costless simulation attacks, ensuring security under the honest-majority assumption. It is important to note that this consensus mechanism guarantees secure blockchain operation, but does not provide verifiable storage itself. To scale the system into a distributed storage network, we complement it with the PDP protocol, which automatically verifies the storage and integrity of archived data while maintaining fast and efficient access to contents. In this way, the system achieves verifiable storage and supports secure outsourcing of web archives.

Our prototype achieved a useful space ratio of approximately 97%, significantly reducing the storage waste observed in previous systems and thereby leading to a greater useful work in consensus. The protocol ensures that the storage space committed by each farmer is directly proportional to their voting power, promoting fairness and resistance to Sybil attacks. Experimental results demonstrate that it is possible to preserve the performance and properties of Chia, with the most notable difference being the initialization cost of downloading web archive data. Both the PoUSp and PDP protocols produced compact and efficient proofs, a crucial property given the sensitivity of block size and processing time in blockchain systems. Most importantly, ArchiveChain shows potential to enhance the quality of service for web archives. Its decentralized architecture and support for direct file access enable faster downloads and reduced page render times.

These results confirm that a permissionless blockchain can scale web archives, lower operational costs, and support additional features such as decentralized currencies and potential smart contracts. ArchiveChain advances blockchain research by prioritizing useful storage and aligning

incentives with societal benefits such as digital preservation. While our evaluation focused on Arquivo.pt, the framework is adaptable to other data-intensive applications.

7.2 Future Work

Given the considerable size and complexity of a permissionless blockchain, future work will focus on scaling the development of ArchiveChain to meet current industry standards, extending its features, and enhancing its overall performance. Potential directions for future work include:

- Extending the transaction layer to support any type of smart contracts, similar to Ethereum's Virtual Machine, Solana's Sealevel, and other platforms.
- Conducting detailed analysis and implementing performance optimizations across all layers of the system, from the data layer to the application layer.
- Ensuring compatibility with the IPFS protocol and extending the system beyond Web Archive data, thereby creating a distributed storage cloud where users can store data and pay for these services. This includes developing an incentive model in which farmers act as vendors and clients rent storage, designing secure handshake and file download protocols, and creating an external marketplace for services.
- Performing formal security proofs and advanced attack modeling for PoUSp and PDP, as well as conducting large-scale blockchain testing and simulations.
- Similar to Filecoin, optimizing PDP proofs that execute on the transaction layer to be as compact as possible. Given the support for recursive proofs, proof compression techniques (such as zk-SNARKs) can be explored to prevent the system from being overwhelmed by excessive proof data.
- Explore how to replace PoT and remove the need for timelords in the system.

Glossary

CBC Cipher Block Chaining. 53

CRUD CREATE, READ, UPDATE, DELETE. 65

ECDSA Elliptic Curve Digital Signature Algorithm. 67

FCCN Foundation for National Scientific Computing. 1, 73

IPFS InterPlanetary File System. 27, 80

PDP Proof-of-Data-Posession. xii, xvi, 3, 5, 6, 20, 33, 35, 37–39, 44, 50–56, 59–67, 70, 73, 76, 79, 80

PoRep Proof-of-Replication. 3, 21, 25, 26, 28–30, 32, 33

PoS Proof-of-Stake. 3, 4, 13, 14, 16, 29, 31–33

PoSp Proof-of-Space. 3–5, 10, 14–17, 21–25, 31–33, 35, 37, 40

PoSpT Proofs-of-Spacetime. 3, 17, 21, 27–30, 32, 33

PoSt Proof-of-Storage. 3, 4, 7, 17, 32

PoT Proof-of-Time. 5, 6, 18, 21–26, 32, 33, 45–50, 62, 79, 80

PoUSp Proof-of-Usefull-Space. xii, xv, 5, 6, 18, 21, 27, 33, 36–38, 40–42, 44–54, 56, 61, 62, 64–67, 70, 73–75, 79, 80

PoW Proof-of-Work. 2, 3, 10–14, 16, 21, 29, 30, 33, 35, 37, 51

QoS Quality of Service. 73

SLOTH Slow-Timed Hash Function. xv, 5, 19, 20, 36, 38, 42–48, 50, 53, 54, 62, 63, 69, 73–75, 89

VC Vector Commitment. 41, 45

VDD Verifiable Delay Decoder. 3, 20, 25–27, 32, 36, 42, 53

VDE Verifiable Delay Encoder. 3, 5, 20, 25–27, 32, 36, 38, 39, 42, 43, 53

VDF Verifiable Delay Functions. 5, 18, 19, 23, 24, 26, 32, 36, 37, 42, 50, 53, 69

WORM Write Once, Read Many. 65

zk-SNARK Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge. 28, 29, 80

Bibliography

[1] Arquivo.pt. https://arquivo.pt/. Accessed: 2025-01-03.

[2] Fccn. https://www.fccn.pt/. Accessed: 2025-01-03.

[3] O arquivo.pt em números. https://sobre.arquivo.pt/pt/imprensa/o-arquivo-pt-em-numeros/. Accessed: 2025-01-03.

[4] Wayback machine. https://web.archive.org/. Accessed: 2025-01-03.

[5] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 357–379. Springer, 2017.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[7] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, 2007.

[8] Autonomys. sloth256-189: Encoder/decoder for the subspace network blockchain based on the sloth permutation. https://github.com/autonomys/sloth256-189/tree/master, 2025. Accessed: 2025-07-22.

[9] Adam Back. Hashcash - a denial of service counter-measure. http://www.cypherspace.org/hashcash/, 1997. Accessed: 2024-11-29.

[10] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[11] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[12] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[13] Juan Benet, David Dalrymple, and Nicola Greco. Proof of replication. *Protocol Labs, July*, 27:20, 2017.

[14] Nir Bitansky, Alessandro Chiesa, and Yuval Ishai. Succinct non-interactive arguments via linear interactive proofs. In *Springer*, 2013.

[15] Bitcoin.org. How does bitcoin mining work? `https://bitcoin.org/en/faq#how-does-bitcoin-mining-work`, 2024. Accessed: 2024-12-02.

[16] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[17] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. `https://eprint.iacr.org/2018/712`, 2018. Accessed: 2024-12-04.

[18] Richard Gendal Brown. The corda platform: An introduction. *Retrieved*, 27:2018, 2018.

[19] Daniel Burkhardt, Maximilian Werling, and Heiner Lasi. Distributed ledger. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–9, 2018.

[20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv preprint arXiv:2003.03052*, 2020.

[21] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association. Accessed: 2024-12-04.

[22] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kazuo Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013, Lecture Notes in Computer Science, vol. 7778*, pages 55–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[23] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. *White Paper, Chia. net*, 9, 2019.

[24] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. Technical report, Chia Network, July 2019. Accessed: 2024-12-13.

[25] NXT Developer Community. Proof of stake discussion. `https://bitcointalk.org/index.php?topic=27787.0`, 2011. Accessed: 2024-12-03.

[26] ConsenSys. Quorum whitepaper v0.2. https://github.com/Consensys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf, 2018. Accessed: 2024-11-19.

[27] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains: (a position paper). In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

[28] Ctufaro. Spreading gossip: Messaging on the bitcoin network. https://ctufaro.github.io/GossipPlot/index.html, 2025. Accessed: 2025-08-22.

[29] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE access*, 7:28712–28725, 2019.

[30] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*, pages 139–147. Springer, 1992.

[31] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

[32] Ethereum. Weak subjectivity. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/weak-subjectivity/, 2023. Accessed: 2024-12-04.

[33] Ethereum. Energy consumption. https://ethereum.org/en/energy-consumption/, 2024. Accessed: 2024-12-04.

[34] Ethereum. The merge. https://ethereum.org/en/roadmap/merge/, 2024. Accessed: 2024-12-04.

[35] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[36] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *Journal of the ACM*, 71(4):1–49, 2024.

[37] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[38] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[39] Chia Network Inc. Chia blockchain state dashboard. https://dashboard.chia.net/d/CL1X4UWnk/blockchain-state?orgId=1&from=now-6h&to=now&timezone=browser, 2024. Accessed: 2024-12-16.

[40] Chia Network Inc. Proof of space. https://docs.chia.net/chia-blockchain/consensus/proof-of-space-1.0/, 2025. Accessed: 2025-07-23.

[41] Polytechnique Insights. Bitcoin electricity consumption comparable to that of poland. https://www.polytechnique-insights.com/en/columns/energy/bitcoin-electricity-consumption-comparable-to-that-of-poland/, 2024. Accessed: 2024-12-02.

[42] IOHK. Motivation. https://why.cardano.org/en/introduction/motivation/, 2020. Accessed: 2024-12-04.

[43] Jelurida. Nxt: The blockchain platform for decentralized applications. https://www.jelurida.com/sites/default/files/NxtWhitepaper.pdf, 2014. Accessed: 2024-12-03.

[44] Monica Chen Jin. Chia bread pudding: A blockchain of useful storage. Master of science thesis, Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal, December 2024.

[45] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[46] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. *Cryptology ePrint Archive*, 2015.

[47] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[48] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. https://www.peercoin.net/read/papers/peercoin-paper.pdf, 2012. Accessed: 2024-12-03.

[49] Jan Kotek and the MapDB Development Team. Mapdb: A java embedded database / collection library. https://mapdb.org/, 2025. Accessed: 2025-09-28.

[50] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013. Citeseer, 2013.

[51] Protocol Labs. Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf, 2017. Accessed: 2024-12-16.

[52] Protocol Labs. Network performance - filecoin docs. https://docs.filecoin.io/networks/mainnet/network-performance, 2024. Accessed: 2024-12-29.

[53] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. 2019.

[54] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate transparency. `https://www.rfc-editor.org/info/rfc6962`, June 2013.

[55] Arjen K Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *Cryptology ePrint Archive*, 2015.

[56] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[57] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[58] Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*, pages 381–409. Springer, 2019.

[59] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Satoshi Nakamoto*, 2008.

[60] Chia Network. Chia proof of space construction v1.1. `https://www.chia.net/wp-content/uploads/2022/09/Chia_Proof_of_Space_Construction_v1.1.pdf`, 2022. Accessed: 2024-12-13.

[61] Chia Network. Proof of time documentation. `https://docs.chia.net/proof-of-time/`, 2024. Accessed: 2024-12-07.

[62] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.

[63] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gaži, Joël Alwen, and Krzysztof Pietrzak. Spacemint: A cryptocurrency based on proofs of space. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 480–499. Springer, 2018.

[64] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 643–673. Springer, 2017.

[65] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

[66] Stefan Scharnowski and Yanghua Shi. Bitcoin blackout: Proof-of-work and the centralization of mining. *Available at SSRN 3936787*, 2021.

[67] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *Journal of cryptology*, 26(3):442–483, 2013.

[68] Filecoin Blog Team. Filecoin and storacha: Spicing up decentralized hot storage like never before. February 2024. Accessed: 2024-12-11.

[69] David Vorick and Luke Champine. Sia: Simple decentralized storage. *Retrieved May*, 8:2018, 2014.

[70] Ieuan Walker, Chaminda Hewage, and Ambikesh Jayal. Provable data possession (pdp) and proofs of retrievability (por) of current big user data. *SN Computer Science*, 3(1):83, 2022.

[71] Hai Wang, Yong Wang, Zigang Cao, Zhen Li, and Gang Xiong. An overview of blockchain security analysis. In *Cyber Security: 15th International Annual Conference, CNCERT 2018, Beijing, China, August 14–16, 2018, Revised Selected Papers 15*, pages 55–72. Springer Singapore, 2019.

[72] Roger Wattenhofer. *The science of the blockchain*. CreateSpace Independent Publishing Platform, 2016.

[73] Jan Werth, Mohammad Hajian Berenjestanaki, Hamid R Barzegar, Nabil El Ioini, and Claus Pahl. A review of blockchain platforms based on the scalability, security and decentralization trilemma. *ICEIS (1)*, pages 146–155, 2023.

[74] Benjamin Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 379–407. Springer, 2019.

[75] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.

Appendix A

Proof-of-Useful-Space Plotting Algorithm

Algorithm 1 $PoUSp.plotFile$ - Parallel Plotting of File Chunks

Input: Raw data D, chunk size C, leaf size S, hash function H, fanout $F = 2$, delay t, leaf size n, farmer public key pk, prime P

Split data D into chunks $\{C_1, C_2, \ldots, C_m\}$ of size C
Initialize empty list $roots \leftarrow \varnothing$
for all chunks C_i in parallel **do**
    Split C_i data into n leaves $L_i = \{l_1, l_2, \ldots, l_k\}$ of size S
    Initialize $indexPosition \leftarrow 0$
    while $|L_i| > 1$ **do**
        Set $parents \leftarrow \varnothing$
        for $j = 1$ to $|L_i|$ step F **do**
            Initialize $hash \leftarrow H(indexPosition||L_i[j].d_j||L_{i+1}[j].d_j)$
            Create node $parent \leftarrow$ Node$(hash, L_i[j], L_{i+1}[j])$
            $parents \leftarrow parents \cup [parent]$
            $indexPosition \leftarrow indexPosition + 1$
        end for
        Set $L_i \leftarrow parents$
    end while
    Set $root_i \leftarrow$ SLOTH.Hash$(L_i[0]||pk, t, p)$ ▷ Slow-Timed Hash Function (SLOTH) encode once at the root level with a time delay parameter t
    $roots \leftarrow roots \cup [roots_i]$
end for
Write all leaf data and $roots$ to output file

Appendix B

Proof-of-Useful-Space Farmer and Timelord Algorithms

Algorithm 2 Farmer.farm

Global Parameters: K ▷ max number of chains per depth
Input: $\gamma_i = (B_i = (i, PoUSp_i, PoT_i), \alpha_i)$ ▷ finalized, fresh, valid block at depth i with payload α_i
State: $S = (S.pk, S.sk, S.plots)$, seen count
if VDF output PoT_i of B_i is invalid **then**
    return ▷ ignore invalid block
end if
if seen count$(i) \geq K$ **then**
    return ▷ ignore if already seen K blocks at depth i
end if
seen count$(i) \leftarrow$ seen count$(i) + 1$
$\mu \leftarrow$ Sig.sign$(S.sk, PoT_i)$ ▷ sign last VDF output
$c \leftarrow H(\mu)$ ▷ challenge is hash of signature
$\pi_{i+1} \leftarrow$ PoUSp.prove$(S.plots, c)$ ▷ compute PoUSp for previous PoT challenge
$\pi_{i+1}.\mu \leftarrow \mu$ ▷ attach signature to the proof
$data_{i+1} \leftarrow$ GeneratePayload() ▷ create new block payload
$\phi_{i+1} \leftarrow$ Sig.sign$(S.sk, (\alpha_i, \pi_{i+1}, data_{i+1}))$ ▷ bind the prev. foliage to the new block

Unfinalized Block $\leftarrow (i + 1, \pi_{i+1}, \phi_{i+1}, \alpha_{i+1} = (\phi_{i+1}, data_{i+1}), S.pk)$
Broadcast Unfinalized Block to network
Update local chain view

Algorithm 3 Timelord.FinalizeBlock

Global Parameters: K ▷ max number of blocks per depth
Input: $\gamma_i = (B_i = (i, \text{PoUSp}_i), \alpha_i)$ ▷ non-finalized block at depth i without VDF
State: $S = (\text{running\_VDFs}, \text{finalized\_VDFs})$
if finalized[i] = K **then**
    return ▷ Already finalized K blocks at this depth
end if
$PoTChallenge \leftarrow H(\text{PoUSp}_i)$
$Q \leftarrow \text{PoUSp.quality}(\pi_i)$ ▷ compute quality of PoUSp
$t \leftarrow \frac{T}{Q}$ ▷ required VDF time inversely proportional to quality
if running_VDFs[i] + finalized_VDFs[i] $< K$ **then** ▷ Less than K blocks finalized or being finalized
    Start computing VDF for B_i with time t iterations and PoTChallenge
    running_VDFs(i) $\leftarrow$ running_VDFs(i) + 1
else if running_VDFs[i] + finalized_VDFs[i] $\geq K$ **then**
    Let B_{slow} be the block with slowest ongoing VDF at depth i
    if $t < t_{\text{slow}}$ **then**
        Abort VDF for B_{slow}
        Start computing VDF for B_i with time t iterations and PoTChallenge
    end if
end if

Upon VDF completion:
Finalize B_i with VDF output
Broadcast finalized block to network
Update local chain view

Appendix C

Proof-of-Data-Possession Archival Phase Algorithms

Algorithm 4 $Farmer.archive$ – Farmer file archival process

Input: File F, Blockchain B, Contract Details D, Farmer $farmer$, WebArchive WA

if $F.size > farmer.availableStorage$ **then** ▷ Farmer lacks enough storage space
    return false
end if

if $D.type = AES$ **then** ▷ AES-encoded file already provided with contract
    $contract \leftarrow D.contract$ ▷ Pre-Signed Storage Contract comes in the same request as the file

else if $D.type = VDE$ **then** ▷ Raw file received, contract will be obtained after encoding
    salt $\leftarrow H(D.fileId \| farmer.publicKey \| D.replicationCounter)$
    $F \leftarrow$ SLOTH_VDE.Encode(F, salt, $D.encodeTime$)
    $contract \leftarrow WA.validateVDE(F)$ ▷ Send encoded file for verification and receive contract
end if

if validate($contract$) **then** ▷ Farmer checks cryptographic details, metadata, incentives
    $contract.sign(farmer.privateKey)$
    B.broadcastContractTransaction($contract$)

    PoUSp.plotFile($F, D.plotTime$) ▷ Generate PoUSp proofs with t intensive computations
else
    return false
end if

return true

Algorithm 5 $WebArchive.archive$ – Web Archive file archival on farmers

Input: file F, contract details D, Web Archive WA, Farmer $farmer$
if $F.size > VDE.Threshold$ **then** ▷ Select VDE for large files
    $D.type \leftarrow VDE$
    farmer.send(F, $D \setminus \{\text{contract}\}$) ▷ Send raw file and details (without contract)
    $F \leftarrow farmer$.WaitVDEEncode(F) ▷ Wait for encoded file from farmer for verification
    if VDE_Decode(F) $= F_{\text{original}}$ **then**
        $contract \leftarrow WA.generateSignedContract(F, farmer.publicKey, D)$ ▷ WA side to maintain sk secrecy.
        Send $contract$ to $farmer$
    else
        return failure
    end if
else ▷ Select AES for smaller files
    salt $\leftarrow H(D.fileId \| farmer.publicKey \| F.replicationCounter)$
    $F \leftarrow$ AES.Encode(F, salt, $WA.secretKey$)
    $D.contract \leftarrow WA.generateSignedContract(F, farmer.publicKey, D)$
    Send F and D (with contract) to $farmer$
end if

Wait for $contract$ to appear on the blockchain
if validateOnBlockchain($contract$) **then**
    Register successful storage of F for $farmer$
    $F.replicationCounter \leftarrow F.replicationCounter + 1$
else
    return failure
end if
return success