Contributions to Permissionless Decentralized Networks for Digital Currencies Based on Delegated Proof of Stake

Rui Pedro Bernardo de Morais

Tese para obtenção do Grau de Doutor em
Engenharia Informática
(3º ciclo de estudos)

Orientador: Prof. Doutor Paul Andrew Crocker
Co-orientador: Prof. Doutor Simão Melo de Sousa

Novembro de 2024

As provas públicas decorreram na Universidade da Beira Interior no dia 30 de outubro de 2024, pelas 15:00 horas.

Composição do júri

Presidente

Doutor Hugo Pedro Martins Carriço Proença
Professor Catedrático da Faculdade de Engenharia da Universidade da Beira Interior

Vogais

Doutor Henrique João Lopes Domingos
Professor Associado da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Doutor José Carlos Bacelar Ferreira Junqueira de Almeida
Professor Associado da Faculdade de Engenharia da Universidade do Minho

Doutor Paul Andrew Crocker
Professor Auxiliar da Faculdade de Engenharia da Universidade da Beira Interior

Doutor André de Matos Pedro
Professor Auxiliar da Faculdade de Engenharia da Universidade da Beira Interior

Doutor Valderi Reis Quietinho Leithardt
Professor Auxiliar do ISCTE-Instituto Universitário de Lisboa

Declaração de Integridade

Eu, Rui Pedro Bernardo de Morais, que abaixo assino, estudante com número de inscrição D2465 do 3º Ciclo de Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridade da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, e que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assim assumo na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 23/11/2024

Acknowledgements

Thank you to my family, friends and advisors, Professor Paul Crocker and Professor Simão Melo de Sousa.

This Thesis has been prepared at Instituto de Telecomunicações, Delegação da Covilhã, and at the Department of Computer Science of the University of Beira Interior, and submitted to the University of Beira Interior for discussion in public session to obtain the Ph.D. Degree in Computer Science and Engineering.

I would also like to thank the support given by RELEASE, RELiablE And SEcure Computation Research Group of the University of Beira Interior, the University of Beira Interior, the Networks Architectures and Protocols group of the Instituto de Telecomunicações and the NOVA LINCS research laboratory.

This work has been funded by Portuguese FCT/MCTES through national funds and, when applicable, co-funded by EU funds under the project UIDB/50008/2020, by operation Centro-01-0145-FEDER-000019 C4 Centro de Competências em Cloud Computing, co-funded by the European Regional Development Fund (ERDF/FEDER) through the Programa Operacional Regional do Centro (Centro 2020) and the NOVA LINCS project (UIDB/04516/2020) with the financial support of FCT—Fundação para a Ciência e a Tecnologia.

Resumo

Com o crescimento e florescimento das sociedades humanas surgiu o desejo de trocar o que era considerado valioso, quer fosse um bem ou um serviço. Inicialmente, essa troca era feita sem intermediários, de forma síncrona (troca direta) ou assíncrona (troca de favores). A primeira tinha a desvantagem de exigir coincidência de desejos e a segunda a necessidade de confiança entre as partes. Ambas eram muito ineficientes e não escalavam de modo adequado. Por isso, o que chamamos de dinheiro foi inventado, que é nada mais do que um bem que é usado como meio de troca entre outros bens e serviços. Desde então, o dinheiro mudou de forma e adquiriu novas funções, nomeadamente unidade de conta e reserva de valor. A forma mais recente de dinheiro é a moeda digital. Este dinheiro não pode ser transferido fisicamente como outras formas, pelo que precisa de uma rede digital para ser transferido, a qual pode ter diferentes características.

Esta tese estuda um tipo específico de redes para moedas digitais: sem permissão (aberta), o que significa que qualquer participante pode ter acesso de leitura e escrita na rede; descentralizada, o que significa, por sua vez, que nenhuma entidade única controla a rede; e que usa Prova de participação delegada *Delegated Proof of Stake* (DPoS) como um mecanismo Sybil de defesa contra ataques Sybil para evitar que a rede seja controlada por atores maliciosos que criam múltiplas falsas identidades.

O objetivo da investigação é cumprir a visão de que uma rede para moedas digitais, para além de ser aberta e descentralizada, deve ser escalável, agnóstica em relação à política monetária, anónima e ter desempenho elevado. Três camadas diferentes da rede são estudadas: a camada de comunicação, responsável por enviar e receber mensagens, a camada de transação, responsável por validar essas mensagens, e a camada de consenso, responsável por chegar a um acordo sobre o estado da rede.

Os dois primeiros objetivos podem ser alcançados na camada de comunicação. Por um lado, é proposta uma forma vertical de escalar o sistema composta por uma gestão de pares e design de prioritização de tráfego baseado em DPoS, oferecendo uma alternativa aos modelos baseados em taxas de transação altamente disseminados. Por outro lado, é apresentada uma forma horizontal de escalar através da fragmentação *sharding* da base de dados.

Na camada de transação, é descrito um quadro geral para tornar o DPoS compatível com o anonimato. Mais especificamente, são propostas duas abordagens diferentes para anonimizar o montante de uma transferência: uma baseada em computação multipartidária e outra na troca de chaves Diffie-Hellman. Finalmente, é desenvolvido um novo algoritmo de seleção de *inputs* engodos, chamado SimpleDSA, para melhorar o anonimato do remetente.

A camada de consenso apresenta dois algoritmos novos de consenso, Nero e Echidna, e

dois métodos para replicação de máquinas de estado: Sphinx (com líder) e Cerberus (sem líder). Estes desenvolvimentos visam realçar o desempenho da rede, especificamente diminuindo a latência das mudanças de estado e aumentando a taxa de transferência, ou seja, aumentando o número de mudanças de estado por unidade de tempo.

Um protocolo que combina a camada de transação e a camada de consenso, chamado Adamastor, é formalizado com provas de segurança e implementado com um protótipo na linguagem Rust. Parâmetros de referência demonstram a praticabilidade do esquema e a aplicação potencial a sistemas de pagamento descentralizados. Embora mais investigação seja necessária, particularmente na implementação de uma rede totalmente operacional, esta contribuição estabelece uma base para futuros avanços.

Em conclusão, esta tese contribui para a área de conhecimento que resulta da fusão da economia e da engenharia informática, oferecendo soluções técnicas para implementar uma visão de um sistema financeiro mais inclusivo, justo, eficiente e seguro. As implicações deste trabalho são de grande alcance, sugerindo um futuro onde as moedas digitais desempenham um papel significativo na formação da finança e tecnologia globais.

Palavras-chave

prova de participação delegada, moedas digitais, criptomoedas, rede descentralizada sem permissão, prioritização, camada de transação, camada de rede, camada de consenso, algoritmo, desempenho, escalabilidade, segurança, anonimato

Abstract

With the growing and flourishing of human societies came the desire to exchange what was deemed as valuable, be it a good or a service. Initially this exchange was made directly through barter, either synchronously or asynchronously with debt. The first had the downside of requiring coincidence of wants and the second the need for trust. Both were very inefficient and did not scale well. So, what we call money was invented, which is nothing more than a good that is used as medium of exchange between other goods and services. Since then, money has changed form and has acquired new functions, namely unit of account and store of value. The most recent form of money is digital currency. This money cannot be transferred physically like other forms, so it needs a digital network to be transferred, which can have different characteristics.

This thesis concerns a specific type of networks for digital currencies: permissionless, meaning that any participant can have read and write access to the network; decentralized, meaning that no single entity controls the network; and that use Delegated Proof of Stake (DPoS) as a Sybil defence mechanism, to prevent the network from being controlled by malicious actors that create numerous false identities.

Its research tries to fulfil the vision that a network for digital currencies, besides being permissionless and decentralized, should be scalable, monetary policy agnostic, anonymous and have high performance. Three different layers of the network are studied: the communication layer, responsible for sending and receiving messages, the transaction layer, responsible for validating those messages, and the consensus layer, responsible for reaching agreement on the state of the network.

The first two goals can be achieved in the communication layer. On one hand, a vertical way to scale the system is proposed composed of a peer management and traffic prioritization design based on DPoS, offering an alternative to highly disseminated fee-based models. On the other hand, a horizontal way to scale is presented through database sharding.

In the transaction layer, a general framework to make DPoS compatible with anonymity is described. More specifically, two different approaches to achieve amount anonymity are proposed: one based on multi-party computation and the other on the Diffie-Hellman key exchange. Finally, a new decoy selection algorithm, called SimpleDSA, is developed to improve sender anonymity.

The consensus layer features two innovative consensus algorithms, Nero and Echidna, and two methods for state machine replication: Sphinx (leader-based) and Cerberus (leaderless). These developments aim to enhance the performance of the network, specifically by decreasing the latency of its state changes and increasing the throughput, i.e., increasing the number of state changes per unit of time.

A protocol that instantiates the transaction and consensus layer, called Adamastor, is formalized with security proofs and implemented with a prototype in the Rust language. Benchmarks demonstrate the practicality of the scheme and potential application to decentralized payment systems. While further research is needed, particularly in implementing a fully operational network, it sets a foundation for future advancements.

In conclusion, this thesis contributes to the area of knowledge that results from the fusion of economics and computer science, by offering technical solutions for implementing a vision of a more inclusive, fairer, efficient, and secure financial system. The implications of this work are far-reaching, suggesting a future where digital currencies play a significant role in shaping global finance and technology.

Keywords

delegated proof of stake, digital currencies, cryptocurrencies, permissionless decentralized network, prioritization, transaction layer, network layer, consensus layer, algorithm, performance, scalability, security, anonymity

Contents

List of Figures

List of Tables

List of Algorithms

UBI	Universidade da Beira Interior
DPoS	Delegated Proof of Stake
PoS	Proof of Stake
PoW	Proof of Work
DSA	Decoy Selection Algorithm
NIZK	Non-Interactive Zero Knowledge
MPC	Multi-Party Computation
DFKE	Diffie-Hellman Key Exchange
DAG	Directed Acyclic Graph
P2P	Peer-to-Peer
CAP	Consistency, Availability, Partition Tolerance
QoS	Quality of Service
SLA	Service Level Agreement
DSA	Decoy Selection Algorithm
CDN	Content Delivery Network
UTXO	Unspent Transaction Output
DDoS	Distributed Denial of Service

Chapter 1

Introduction

1.1 Background

In 2008, Bitcoin was created as the first permissionless decentralized network for the digital currency with the same name. This combination would become known simply as cryptocurrency, in what is essentially the combination of two of the most powerful technologies ever created: money and internet. In this section we take a look on how these two technologies evolved through time and fused together to originate a whole new field of knowledge.

The most fundamental property of money is to act as a medium of exchange of goods and services. There is some dispute about what was the first medium of exchange used, with economists stating that it was barter and anthropologists defending it was based on the principles of the gift economy, where there is not an explicit agreement of exchange, and debt [Gra11]. One can argue that debt without an intermediary means of exchange, like a coin or note, is also barter, but asynchronous instead of synchronous.

The point is that, when human societies were small, economic exchanges were based on trust. But this does not scale well and barter began to be used to trade with strangers or even enemies. However, this also does not scale well, because there must be coincidence of wants and it is hard to function as an unit of account, since there are so many possible combinations of exchange. So, with the growth of societies, there was a discernible shift towards commodity money around 5000 BC [OS03]. The limitations of barter and the gift economy paved the way for the adoption of intrinsically valuable commodities such as salt, spices, cattle, and grains as standardized mediums of exchange. These commodities, imbued with intrinsic value, facilitated trade beyond immediate reciprocal wants.

But this form of money also had its limitations, namely its poor portability and durability. The emergence of metal coins around 600 BC in ancient Lydia marked a significant evolution. These early coins, crafted from an alloy of gold and silver known as Electrum, heralded the advent of coinage. Successive cultures embraced this concept, minting coins adorned with images of rulers and deities, thereby institutionalizing and proliferating coin-based trade [Sch06]. Money started to be controlled and regulated by a central authority, the government.

The innovation of paper money in the 7th century AD in China represented a departure from tangible intrinsic value. The Tang and Yuan Dynasties saw the proliferation of paper

currency, a practice later chronicled by the Venetian explorer Marco Polo and eventually adopted by European nations in the 17th century [AGP+22].

In the transition from intrinsic to representative value, the gold standard emerged in the 19th century, pegging currencies to specific quantities of gold. This system promised stability but was progressively abandoned in favor of greater fiscal flexibility in the mid-20th century [Eic19].

The adoption of fiat money marked a paradigm shift wherein currency, devoid of intrinsic value and not pegged to the price of any particular commodity, was established through governmental decree. Its value was predicated on trust and the societal consensus regarding its utility as a medium of exchange. Around the same time, with the advent of computers, fiat money also started to take a digital form in the form of payment cards [Ste11].

In the late 20th century, the internet would revolutionize money, specially with the advent of the World Wide Web, since, in its infancy, the internet was primarily a tool for researchers, academics, and defense agencies.

The development of the World Wide Web, or Web 1.0, in the late 1980s and early 1990s, marked a revolutionary shift in information dissemination and communication, because it created a user-friendly interface for navigating the vastness of the internet. This led to the explosion of websites, digital communication, and online services, including e-commerce and online payments.

In its initial phase, spanning the late 1980s to the early 2000s, the web was characterized by static content and several communication protocol competed for distributing and searching documents. For example, Websites using the http protocol primarily served as read-only platforms, offering one-way communication with limited user participation. The era was dominated by content publishers, with anyone possessing HTML knowledge able to create a webpage. Interactivity was constrained to basic forms and guestbooks, with no scope for dynamic content generation by users.

The early 2000s ushered in the era of Web2.0, marked by a surge in user-generated content. The web transitioned into a dynamic space, fostering real-time interaction through platforms like blogs, wikis, and social media sites. Users were empowered to create, modify, and share content, with features such as comments, likes, and shares becoming standard.

With the growing population of internet users, there was a significant demand for structured platforms where they could interact, share, and access information. As the limitations and vulnerabilities of centralized systems (like server downtimes, data breaches) became apparent, there was a push towards distributed systems. These systems, like Content Delivery Networks (CDNs), spread data across multiple servers in different locations to ensure uptime and fast access.

Today, the internet is not only composed of centralized services and systems but also decentralized ones. This shift, especially evident in the 2000s and 2010s, has democratized access to a variety of services. Applications such as BitTorrent, which emerged in the early 21st century, exemplified the potential of decentralization by allowing peer-to-peer file sharing, ensuring resilience and censorship-resistance by distributing power evenly among participants.

This transition laid the groundwork for Web3.0 and it was, in this context, that Bitcoin emerged in 2008 as a pioneering peer-to-peer electronic cash system, circumventing traditional financial intermediaries. This innovation represented a fundamental shift in financial power dynamics, not just a novel means of transaction, and can be seen as the first successful implementation of one of the core ideas of the Austrian economics school of thought, which defends that the competition of private money is beneficial for society [Hay78].

This perspective on the evolution of the web is represented in figure 1.1.

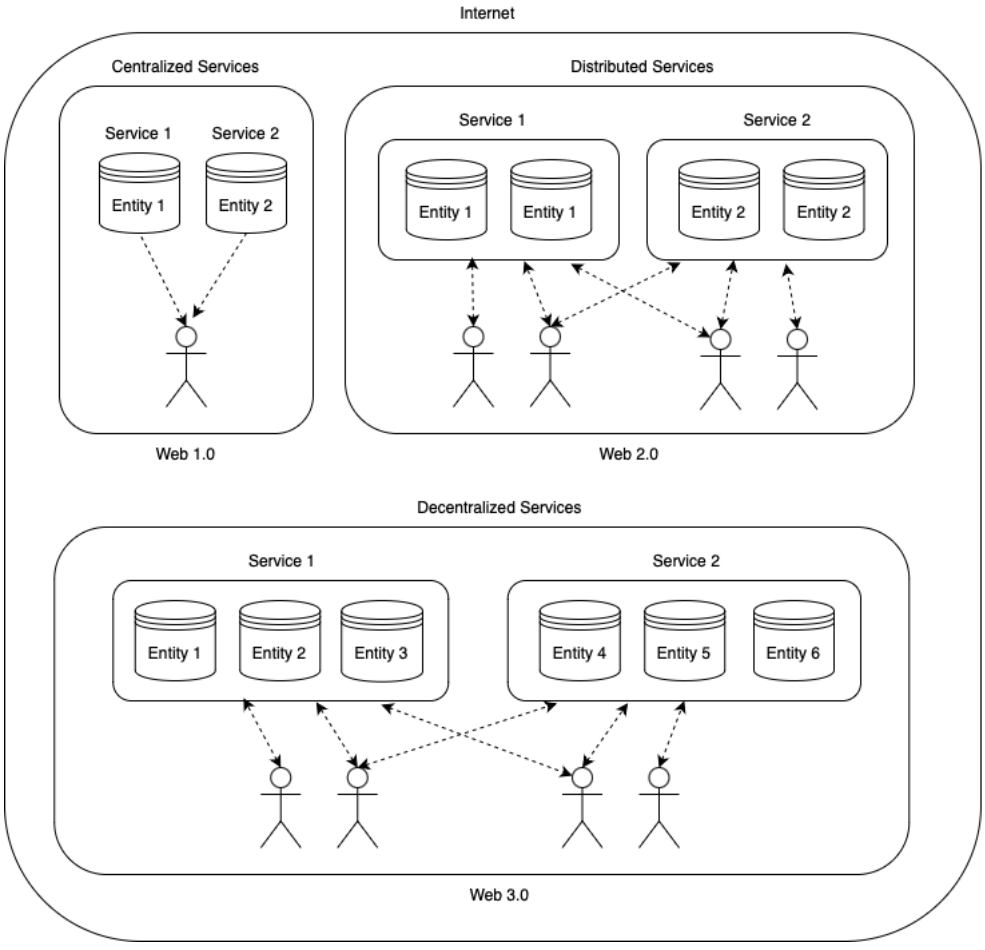

Figure 1.1: The Evolution of the Web

In table 1.1 we summarize in a simplified way the evolution of money, which reflects a progressive adaptation of its properties to meet the demands of increasingly complex eco-

nomic systems. Each successive form of money has introduced or enhanced specific attributes that have improved the efficiency and functionality of financial transactions.

Table 1.1: The Evolution of Money

	Gift	Barter	Commodity	Metal	Paper	Digital	Crypto
Durability			✓	✓	✓	✓	✓
Portability	✓					✓	✓
Fungibility			✓	✓	✓	✓	✓
Finality		✓	✓	✓	✓	✓	✓
Divisibility			✓	✓	✓	✓	✓
Scalability						✓	✓
Auditability							✓
Censorship Resistance	✓	✓	✓	✓	✓		✓
Anonymity	✓	✓	✓	✓	✓		✓
Decentralized	✓	✓	✓				✓

- Gift Economy: In a gift economy, value is exchanged without a direct quid pro quo. Transactions are based on goodwill, social relations, or moral obligations. Durability or divisibility are not primary concerns here, and anonymity can often be maintained.

- Barter System: This involves direct exchange of goods and services without using a medium of exchange like money. It's not particularly portable or scalable, and its effectiveness depends on the coincidence of wants. Anonymity and censorship resistance are inherent due to the direct nature of transactions.

- Commodity Money: This type of money has intrinsic value; examples include gold, silver, or other valued items. It's durable, fungible (each unit is the same as others), divisible, and has finality in transactions. However, it's not very portable or scalable compared to modern money forms.

- Metal Money: Coins made from metals like gold, silver, or copper. They inherit the commodity properties like durability, fungibility, divisibility, and finality. Metal money is more portable than bulk commodities but less so compared to paper or digital forms.

- Paper Money: Represents value in a more portable and scalable form. It's lightweight, easy to carry, and can represent larger values. Paper money maintains durability, fungibility, divisibility, and finality, but it lacks in aspects like scalabilty and, to some extent, auditability.

- Digital Money: Includes electronic representations of money, like bank deposits. It's highly portable, scalable, and durable in the sense that it doesn't physically degrade. Digital money is fungible, divisible, and offers transaction finality. However, it may lack in anonymity and is subject to potential censorship. There is some lack of auditability concerning monetary policy, specially in the creation of new money.

- Cryptocurrency: A digital or virtual currency secured by cryptography, often decentralized. It shares many properties with digital money like portability, scalability, durability, fungibility, and divisibility. Additionally, it often provides anonymity, censorship resistance, and auditability of monetary policy, since the protocols are open-source. Together with Gift and Barter are the only decentralized forms of money.

Each form of money evolved to address specific needs and limitations of its predecessors, reflecting changes in technology, society, and economic practices.

This progression in the properties of money demonstrates a clear trajectory towards addressing the evolving challenges of economic transactions. Each stage in the evolution of money has been instrumental in adapting to the changing needs of economies, driven by technological advancements and shifts in societal structures. This evolution continues to shape the economic landscape, presenting new possibilities and challenges in the financial domain.

1.2 Motivation

In the evolving landscape of global finance, private digital currencies stand as a beacon of hope and opportunity, particularly for those in regions plagued by limited access to traditional banking systems or by unstable national currencies. These digital currencies offer a vital alternative, not just as a medium of exchange but also as a protection against the inflation and loss of purchasing power that often afflict stable currencies, exacerbated by the banking system's propensity for coin creation.

Today there are thousands of cryptocurrencies, each one with its own characteristics and design. The motivation of this thesis is based on the observation that an ideal cryptocurrency does not exist yet due to ideological and technical challenges. Our goal is to contribute to solve the latter.

One of those characteristics is anonymity, which is especially important in countries where authoritarian regimes prevail. Here, decentralized digital currencies offer a discreet channel for financial transactions, safeguarding individual anonymity and providing a measure of protection against oppressive financial surveillance. Most cryptocurrencies only provide pseudo-anonymity, meaning that it is possible at some level to link the identity of a user in the network (the address) to its real world identity.

Besides that, a cryptocurrency should be scalable enough to cater to a potentially global market, and high-performing (high throughput and low latency), to ensure efficient and timely transactions. Performance should be at least comparable with existing centralized payment systems like Visa or Mastercard.

Since a digital currency cannot be traded physically, it needs a network that mediates the transfer of the currency ownership between parties. Table 1.2 describes the different types that these networks can have, being a permissionless network the only one that avoids censorship. However, the permissionless nature of these networks introduces complexities. They are inherently more susceptible to malicious actors and various forms of cyberattacks compared to their permissioned counterparts. Consequently, a robust Sybil defense mechanism is indispensable. While Proof of Work (PoW) and Proof of Stake (PoS) are common defense mechanisms, PoW is often criticized for its inefficiency and high energy consumption. Delegated Proof of Stake (DPoS) has emerged as a more decentralized alternative to PoS, allowing participants to delegate their consensus power without the necessity of running a node, thereby enhancing network participation and decentralization.

	Permissioned		Permissionless
	Private	Public	
Read Access	Restricted	Unrestricted	Unrestricted
Write Access	Restricted	Restricted	Unrestricted

Table 1.2: Typology of Networks

This thesis also explores the idea that the network can and should be agnostic concerning monetary policy. Particularly focusing on the centralization effects spurred by the creation of new coins to pay for transaction fees and consensus rewards, we explore alternative transaction prioritization mechanisms and anti-flooding measures that transcend the traditional reliance on fees. Additionally, the necessity of validation rewards as an incentive is questioned, drawing parallels with other decentralized networks like BitTorrent and Tor, which operate effectively without such incentives. Besides being a centralization force, fees at the network level create nefarious incentives, since nodes with consensus power have interest in its increase, they also exclude users with less economic power from the network and create a bad experience for all users, because fees are hard to estimate and result in dust, which are coins whose amount is too low to pay for the fees, so they cannot be transferred and become useless.

Through this research, we aim to contribute to the development of permissionless decentralized digital currency networks that are not only technically robust and efficient but also align with the broader goals of financial inclusivity, anonymity, and democratization, laying the groundwork for a more equitable, sustainable, performant and scalable digital financial ecosystem.

1.3 Problem Statement

A network of computers requires a set of rules or protocol in order to transfer data between those computers. Various protocols exist depending on the physical connections, type of

transmission etc. The internet protocol is the biggest network in the world for routing and addressing data packets however many transmission protocols exist and the transferred data can be of arbitrary types; At the application level for example, in BitTorrent, it is audio and video files, whilst in crytpocurrencies it is information representing the transfer of ownership of a currency, with at least a sender, a receiver and an amount.

In this document, we undertake a focused research aimed at enhancing key aspects of permissionless decentralized networks for digital currencies based on DPoS, by addressing and proposing improvements in three critical areas: performance, anonymity, scalability and security. Our goal is to design the necessary building blocks to develop a permissionless decentralized network for digital currencies based on DPoS that has low latency and high throughput, is scalable, anonymous and no monetary policy.

In this research, we assume as the system model a general permissionless decentralized authenticated peer-to-peer network with n nodes and a native digital currency that uses DPoS as a Sybil defense mechanism. The goal of this network is to process payments on its digital currency and, for that, the nodes need to reach consensus on what payments are valid and in what order.

The network uses asymmetric cryptography for the authentication of peers, with each message being signed by the public key of the peer. Each node has associated a given delegated stake and its power on the consensus is proportional to that stake of n nodes that run a software. At most $f = (n-1)/3$ of those nodes can be malicious, meaning they can crash and behave arbitrarily by not running the correct software.

This software can be divided into five different layers (figure 1.2): the application layer, which is an interface for interacting with users that want to make and receive payments on the network. This layer is usually called a wallet and is responsible for creating, broadcasting and receiving transactions. The communication layer is responsible for communicating with other nodes on the network, receiving and sending messages, which can be transactions, consensus messages or requests of these. Transactions go to the transaction layer to be processed and validated. If valid, the consensus layer is responsible for reaching agreement about what transactions are confirmed on the network and in which order. Finally, when confirmed, transactions are stored in the storage layer of the node, also called ledger. This ledger can have different structures, being the most common the blockchain. However, lately it has been replaced by Directed Acyclic Graphs (DAGs) for performance reasons.

1.4 Contributions

In this work, we have made several significant contributions (table 1.3) to advance the state of the art in consensus, anonymity and scalability of permissionless decentralized payments networks, spanning across the following layers of this network architecture:

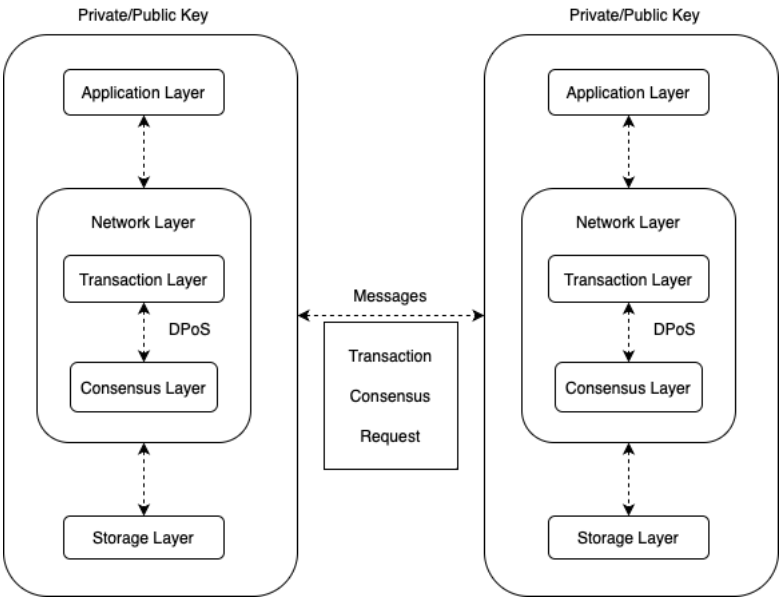

Figure 1.2: Problem Statement

- Communication layer: We introduce a dynamic rate control mechanism tied to each peer's delegated stake in the network. This rate adjusts based on network usage levels, optimizing resource utilization and prioritization without the need for fees. Additionally, we address the scalability challenge in permissionless networks by proposing a well-defined peer structure based on delegated stakes. This structure prevents lower-staked nodes from being censored while allowing for the demotion of higher-stake nodes if they engage in malicious activities, such as broadcasting conflicting transactions or consensus messages. Furthermore, we implement traffic prioritization to allocate resources proportionally to the importance of different message types, such as transactions, consensus messages, and their respective requests.

- Transaction layer: At the transaction layer, we propose several innovations. In a general way, we propose a framework to make anonymity compatible with DPoS. More specifically, we propose SimpleDSA, an efficient Decoy Selection Algorithm that counters homogeneity attacks and chain analyses in cryptocurrencies using linkable ring signatures, providing a mechanism to identify and prune spent outputs. We also introduce a multi-party computation method using the Twisted ElGamal scheme to encrypt all transaction amounts on the network with a shared public key, managed by higher stake accounts. This encryption allows for joint decryption of total delegated stakes without revealing individual account stakes. Additionally, we utilize a Diffie-Hellman Key Exchange protocol for transaction amount anonymization, enabling both the transaction receiver and the coin delegate to know the transferred amount.

- Consensus layer: We contribute several new algorithms. Nero, a leaderless deter-

ministic consensus algorithm, provides eventual order suitable for DAG-based cryptocurrencies. Echidna addresses the Multi-valued Byzantine Consensus with a fast mechanism for handling conflicting values. Sphinx, building upon Echidna, offers a leader-based approach to State Machine Replication, reducing latency due to its efficient handling of conflicting values. Cerebrus, another leaderless algorithm, resolves the scalability issues of leaderless approaches by proposing a set of values instead of a single proposal. We also present open-source Rust implementations of Sphinx and Cerebrus, benchmarking them against the state-of-the-art Bullshark algorithm to demonstrate comparable latency, resilience, and scalability.

- Finally, we propose Adamastor, a protocol that aggregates some of the ideas proposed in this document to serve as the basis of a new high performance anonymous permissionless decentralized payments network based on DPoS. We formalize the security proofs of Adamastor and implement it in Rust with the Nero consensus algorithm, showing significant performance improvements over existing systems. Our open-source implementation contributes to the research community and sets the stage for future developments.

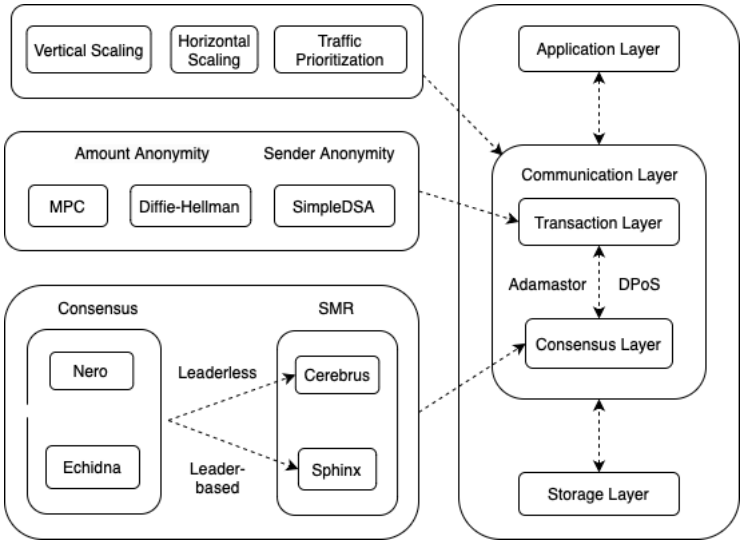

Figure 1.3: Thesis Overview.

The document is structured to reflect the order of these contributions and concludes with a section on conclusions and future work. The following contributions were published:

- Chapter 3: *A tool for implementing privacy in Nano*, 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS) [MCMdS20].
- Chapter 4: *Nero: A Deterministic Leaderless Consensus Algorithm for DAG-Based Cryptocurrencies*, Algorithms 2023 [MCL23].
- Chapter 4: *Echidna: A New Consensus Algorithm for Efficient State Machine Replication*, 2023 IEEE The Fifth International Conference on Blockchain Computing

and Application(BCCA) [MCDS23b].

- Chapter 5: *Adamastor: a New Low Latency and Scalable Decentralized Anonymous Payment System*, 2023 arXiv [MCdS23a].

Chapter 2

Communication Layer

2.1 Introduction

In a peer-to-peer (P2P) network, the communication layer plays a critical role in the overall architecture and performance of the system. This layer is fundamental because it provides the necessary mechanisms for communication and data exchange between peers, which are the core functionalities of any P2P network.

The communication layer is responsible for the routing of messages and the management of connections, which is essential for a decentralized system without a central server. It handles the complexity of a dynamic network topology as peers join and leave the network, maintaining robustness and fault tolerance. Efficient routing algorithms at this layer can significantly reduce latency, increase throughput, and ensure that the network scales effectively to handle a large number of nodes.

The communication layer also incorporates various techniques to ensure security and privacy in P2P networks. It can implement encryption, authentication, and anonymity protocols to protect the data and the identities of the network participants from various threats like eavesdropping, spoofing, and man-in-the-middle attacks.

Furthermore, the communication layer has a significant role in traffic management and congestion control. It can prioritize data packets and manage bandwidth allocation to maintain the performance of the network under different loads. This is particularly important in P2P networks, where the distribution of resources can be highly variable.

In the context of P2P cryptocurrencies and blockchain technologies, the communication layer's importance is magnified. It must not only handle the typical challenges of P2P systems but also ensure that the ledger is consistently and securely maintained across all nodes without a trusted third party. This includes the propagation of transactions and blocks, handling forks, and ensuring consensus without centralized control.

In summary, the communication layer is the backbone of a P2P network's infrastructure, providing the essential services that allow for the decentralized, efficient, and secure functioning of the network. Its design and implementation are critical determinants of the network's ability to meet its performance, reliability, and scalability goals.

2.1.1 Problem Statement

We formulate our research problem directly from the following open question (open question number 5) from [DPS+21]:

"How to avoid the security (and we should add, quality of service risks) associated with transaction floods? An attacker may try to flood the network with meaningless transactions and thus cause the nodes to waste resources on them. How can one quickly identify bad transactions and discard them? How to trade off the verification cost, punishment mechanisms and velocity?"

These questions are one of the most important ones to ask in a permissionless decentralized network, since an open system is inherently fairer, but is also more prone to abuse by malicious actors. However, and surprisingly, not a lot of attention has been given to them in the literature when compared to other topics in this field. One of the reasons for this can be found the widespread use of fees in the industry, the underlying idea being that a malicious actor can abuse the system, as long as it pays the right price, and this cost converts directly into revenue to those that control the system. This is bad for the users, which are affected by it, and can even create a conflict of interests, where it is in the interest of those who control the system to keep fees high, in order to profit more from them.

In the forthcoming chapter, we address the challenge of scaling permissionless decentralized networks in general and the complexities at the communication level for digital currencies with Delegated Proof of Stake based systems in particular.

A primary concern in this realm is the phenomenon of transaction floods and the accompanying risks since it is not possible to objectively distinguish between an honest user that wants to legitimately participate in many transactions and a malicious user that wants to attack the system by spamming it, this subject is also directly related to the security of the network. These floods manifest as an onslaught of meaningless or low-value transactions initiated by attackers, aimed at inundating the decentralized network. Such actions pose a grave threat, potentially exhausting the resources of network nodes, thereby slowing down legitimate transactions. In extreme scenarios, this can escalate to full-scale network paralysis. The challenge lies in devising effective strategies to mitigate these attacks, a task compounded by the need to preserve the inherent decentralized and open nature of the network.

Another pivotal aspect is the identification and discarding of bad transactions. The ability to swiftly and accurately discern and eliminate illegitimate or meaningless transactions is crucial. However, this strategy has its limitations, as attackers can perpetrate the floods using transactions that are correctly formed.

Furthermore, the solution to these problems must strike a delicate balance between various factors. These include the cost associated with the verification and propagation of

transactions, the design and effectiveness of punishment mechanisms targeted at malicious actors, and the overall velocity of transactions within the network. It is imperative that the measures implemented do not adversely affect honest users of the system. Addressing these interconnected issues requires a nuanced approach that considers the unique dynamics of DPoS systems and the broader implications on network efficiency and security.

2.1.2 Technical Overview

In this chapter we propose two main ways to scale a permissionless decentralized network that, together, can form a resilient and fair communication layer without fees based on Delegated Proof of Stake. Each contribution can be summarized in the following way:

- Vertical Scaling: The two main components of this strategy are peer management and traffic prioritization. It is not feasible to have all nodes on a permissionless network connected with each other, since it does not scale well. Because of that, a well defined peer structure is needed in order to optimize the control flow of the network. The base for the distribution of peers is based on their delegated stakes in a way that lower staked nodes can not be censored from using the network. Also higher stake nodes can still be demoted on the hierarchy if they misbehave, such as for example by broadcasting conflicting transactions or consensus messages.

  Complementing that, each peer has a rate control associated with its delegated stake on the network. This rate is dynamic and adjusted based on the levels of usage on the network, so that resources are not wasted. There are different types of messages in a network, being the most important transactions, consensus messages and requests of both transactions and consensus messages. Resources allocated to propagate each one of these messages should be proportional to its importance to the network and the weight of the node that originated them. According to that, they can be either rebroadcast to all peers (point to point communication), only to a few (as in a gossip protocol) or none at all.

- Horizontal Scaling: This can be see as intra-scaling (within a node), instead of the inter-scaling of vertical scaling (between nodes). The idea is that each node can be divided into different peers in different locations. This not only increases the resilience of the network, but it also increases performance, since less computational resources are allocated to each peer (input), which can produce confirmed transactions (output) in an independent way,

  We propose a deterministic approach to sharding instead of an ever changing randomized one, that relies only on database sharding and not on consensus sharding. This is more secure because the security model of consensus remains the same and

the shards do not have to change dynamically for security reasons. It is also easier to implement.

2.1.3 Related Work

The related work section of this study draws upon a diverse range of research, focusing on the exploration and analysis of peer-to-peer networks in various cryptocurrencies. These works collectively contribute to a deeper understanding of the dynamics, challenges, and security aspects of permissionless decentralized networks.

A significant contribution to the field is the comprehensive study of Monero's peer-to-peer network [CYD+20], which offers an in-depth analysis of Monero's network topology, node distribution, and connectivity. The authors developed tools such as NodeScanner and NeighborFinder for network exploration, revealing insights and potential vulnerabilities that could impact blockchain safety, such as the network centralization, and user privacy, such as the identification of the geolocation of the nodes.

A detailed technical analysis of the Bitcoin peer-to-peer network can be found in reference [ECP21]. This research delves into the geographical distribution of peers, autonomous systems, latency measurements, and the propagation of transactions and blocks within the Bitcoin network. It provides valuable insights into the performance, scalability, and security aspects of Bitcoin's network infrastructure. For example, a small fraction of influential nodes, accounting for about 2% of the total but controlling a disproportionate share of the mining power, was identified. These nodes are hypothesized to act as gateways to mining pools, significantly influencing transaction and block inclusion.

The BlockP2P protocol [HZD+19], which clusters peer nodes based on geographical proximity and uses a hierarchical topological structure for efficient data dissemination, demonstrates significant improvements in network performance.

Reference [WZY+21] offers a comparative analysis of Monero's network with other leading cryptocurrencies like Bitcoin and Ethereum. It addresses network attacks, security, and privacy implications, contributing to a broader understanding of the security and efficiency of peer-to-peer networks in blockchain systems.

Another useful tool is the Blockchain Measurement System (BMS), which was designed to analyze Bitcoin and Ethereum networks [GBE+18]. BMS focuses on assessing peer-to-peer latency, bandwidth allocation, and network structure. Their findings suggested a tendency for Bitcoin nodes to be clustered in data centers, indicating a degree of centralization in the mining process. They posited that decentralizing the mining process could enhance the efficacy of consensus protocols.

A study on the impact of churn in the Bitcoin network concluded that a substantial majority of nodes (97%) experienced churn, meaning that they frequently leave and join the

network. This results in a significant increase in propagation time, adversely affecting network performance [ISTY19].

One work presented a more security-focused perspective by demonstrating an attack capable of isolating a specific node within the network [HKZG15]. By monopolizing a node's connections, they successfully populated the node's tables with malicious entities, significantly increasing the likelihood of a successful attack. Following this revelation, countermeasures were developed and implemented to mitigate such eclipse attacks.

Despite these advances, gaps in the literature remain. The potential for Sybil attacks have not been thoroughly explored, specifically in (delegated) PoS network, which is the focus of this chapter. These unaddressed areas highlight the need for ongoing research to fully understand and secure permissionless networks.

2.2 Vertical Scaling

This section provides some high-level tools in the form of algorithms with the goal of optimizing the structure of a P2P authenticated network based on the (delegated) stake of its peers. This includes allocating peers into tiers based on their stake, establishing connections between them and define the type of outbound traffic in order to maximize the probability that messages reach every peer while minimizing bandwidth costs, and adjusting the inbound traffic of each connection based on its stake to prevent DoS attacks.

2.2.1 Peer Management

The peer management system is structured as a pyramid with tiers dynamically defined by stake distribution, is an innovative approach to scaling a decentralized network while maintaining performance. This system aims to efficiently manage network traffic and resource allocation, adapting to the varying stake levels of participants.

In a hierarchical pyramid structure with multiple tiers, the system organizes peers based on their stake in the network. The network is segmented into tiers determined by the delegated stake, and each tier is further divided into n equal parts. Each part corresponds to a segment of the account public key space. Unlike static models, this system dynamically adjusts its tiers in response to the fluctuating distribution of stakes, ensuring balanced traffic flow and resource allocation. The periodic assessment of stake distribution allows for the reallocation of peers to suitable tiers.

This tiered structure enhances scalability and performance by managing the number of active participants in critical network processes such as transaction validation and consensus. It prevents performance bottlenecks, allowing the network to accommodate thousands of peers without compromising performance. Each tier assigns roles and respon-

sibilities; higher-tier peers, given their larger stake, undertake more significant tasks like leading consensus rounds or validating larger transactions. Lower-tier peers contribute by propagating transactions and participating in secondary validation processes, integral to network operations.

Resource management within this framework is optimized by allocating more substantial tasks and higher bandwidth requirements to higher-tier peers, while lower-tier peers handle tasks and traffic within their capacity. This ensures network efficiency and responsiveness. The system's design aligns incentives across different stake levels, motivating peers to act in the network's best interest. Those in higher tiers, facing greater risks in the event of network failure or malicious activity due to their larger stake, are incentivized to maintain network integrity and performance.

The flexibility of the dynamic tier system allows the network to adapt to stake distribution changes, critical for long-term stability and scalability. This adaptability is essential in responding to evolving network conditions and stakeholder behaviors. Security is a paramount consideration in the pyramid structure, ensuring that no single tier or group of peers can dominate or compromise the network. Mechanisms are in place to prevent manipulation of tier assignments and to detect and respond to malicious activities at any tier level.

To efficiently allocate new peers into the respective tier, each peer must know the IP address of at least one peer in each account space of every tier. This enables new peers to connect to any node in the network and be directed to a peer in their bucket, which introduces them to other relevant connections. By utilizing a decentralized hash table adapted to use public keys as keys instead of hashes, with the value being the corresponding IP address, the system ensures effective connectivity and network integration.

In the proposed framework, as an example peers within the network can be categorized into three tiers based on their delegated stakes (figure 2.1), creating a hierarchical structure that influences their roles and responsibilities in the network. This tiered system is designed to optimize network traffic and scalability, while also ensuring robustness against potential censorship and faults.

The first tier involves peers divided into 'n' segments, with each segment corresponding to an equal portion of the public key space. This division aims to create an orderly and manageable structure within the tier, facilitating more efficient connections and interactions among peers.

The second tier is crucial for maintaining the network's integrity. It is structured to hold at least 33% + 1 of the voting weight, a measure put in place to prevent the possibility of censorship of nodes belonging to the third tier. This ensures that the network remains decentralized and resistant to manipulation by any single group or entity.

However, the third tier presents a unique challenge. Due to the potential for an unlimited

number of nodes, creating a deterministic structure for connections becomes impractical. Therefore, this tier may require a degree of randomness in its connectivity, possibly limited to a certain subset to maintain manageability and efficiency.

One of the primary objectives of this tiered system is to ensure that all nodes have varying communication paths for confirming transactions. This diversity is essential to safeguard against faults or Byzantine nodes that might attempt to disrupt the transaction process. While the most efficient path for communication would theoretically be a fully connected network where all nodes are directly linked to each other, such a structure is not scalable. As networks grow, the need for a more structured approach becomes apparent, and the proposed tiered system with its varied and efficient communication paths offers a viable solution to address these scaling challenges.

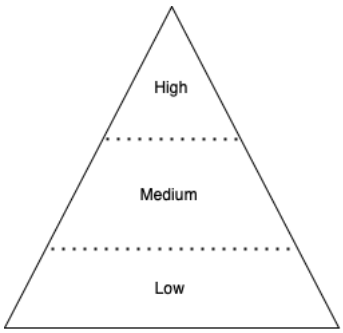

Figure 2.1: General Peer Structure

A specific peer structure example can be seen in figure 2.2, where the nodes identified with ”A” correspond to the high tier, B to the medium tier and C to the low tier. The peers of the high tier are all interconnected, the peers of the medium tier are connected to the peers of the contiguous sections in the tier above, the same tier (left and right), and the tier below. The lowest tier is similar, except it does not have a tier below.

To better understand it, some peer connections are described below:

- A1 peers: A2, A3, A4, B1, B2, B3, B4.
- A2 peers: A1, A3, A4, B5, B6, B7, B8.
- A3 peers: A2, A1, A4, B9, B10, B11, B12.
- A4 peers: A2, A3, A1, B13, B14, B15, B16.
- B1 peers: A1, B5, B6, B7, B8, B13, B14, B15, B16, C1, C2, C3, C4, C5, C6, C7, C8.
- B3 peers: A1, B5, B6, B7, B8, B13, B14, B15, B16, C1, C2, C3, C4, C5, C6, C7, C8.
- B2 peers: A1, B5, B6, B7, B8, B13, B14, B15, B16, C9, C10, C11, C12, C13, C14, C15, C16.

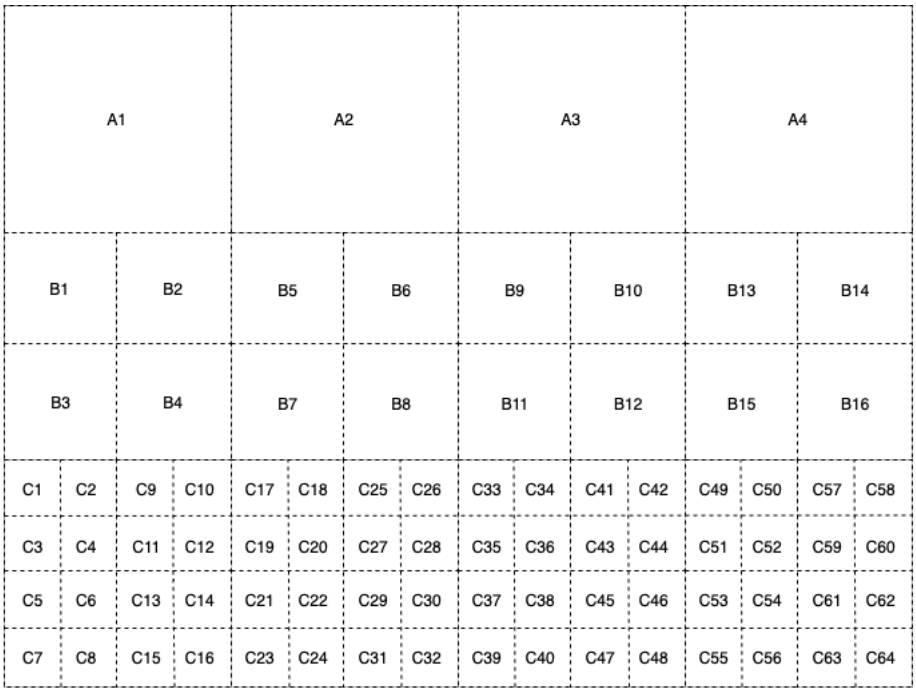

Figure 2.2: Specific Peer Structure

- B4 peers: A1, B5, B6, B7, B8, B13, B14, B15, B16, C9, C10, C11, C12, C13, C14, C15, C16.
- C1 peers: B1, B3, C9, C10, C11, C12, C57, C58, C59, C60.
- C2 peers: B1, B3, C9, C10, C11, C12, C57, C58, C59, C60.
- C3 peers: B1, B3, C9, C10, C11, C12, C57, C58, C59, C60.
- C4 peers: B1, B3, C9, C10, C11, C12, C57, C58, C59, C60.
- C5 peers: B1, B3, C13, C14, C15, C16, C61, C62, C63, C64.
- C6 peers: B1, B3, C13, C14, C15, C16, C61, C62, C63, C64.
- C7 peers: B1, B3, C13, C14, C15, C16, C61, C62, C63, C64.
- C8 peers: B1, B3, C13, C14, C15, C16, C61, C62, C63, C64.

Algorithm 1 is designed for organizing peers in a network into different tiers according to the amount of stake the peer hold or control and amount of stake allocated to each tier. This approach is particularly relevant in blockchain and decentralized systems where stakeholding can be an important factor in determining a peer's role or influence.

Algorithm 1: Distribute Peers into Tiers Based on Predefined Stake Allocations

Result: Distribute peers into tiers based on predefined stake allocations for each tier

Input: List of peers with their stakes, List of predefined stake allocations for each tier

Output: List of tiers with peers distributed based on stakes

```
Sort peers in descending order of their stakes;
Initialize an empty list of tiers, each corresponding to one of the predefined stake allocations;
Initialize a list of current stakes for each tier, initially zeros;
for each peer in the sorted list of peers do
    /* Find the tier for the current peer based on the remaining stake capacity */
    tierIndex ← −1;
    for each tier from the first to the last do
        if current stake of the tier + peer's stake ≤ predefined stake allocation for the tier then
            tierIndex ← current tier index;
            Break;
            /* Break the loop once the suitable tier is found */
        end
    end
    /* Since total stakes match, tierIndex should never be -1 */
    Add peer to the tier corresponding to tierIndex;
    Update the current stake for that tier by adding the peer's stake;
end
return List of tiers with peers;
```

The algorithm starts by accepting two inputs: a list of peers each associated with a stake value, and a list of predefined stake allocations for each tier. The stakes represent the importance or weight of each peer within the network, which could be in the form of tokens, reputation scores, or any other quantifiable metric.

Initially, the algorithm sorts the peers in descending order according to their stakes, positioning the peers with higher stakes at the beginning of the list. An empty list of tiers is then initialized, corresponding to the predefined stake allocations, along with a list to track the current total stake in each tier, initially set to zero.

The algorithm proceeds by iterating through each peer in the sorted list. For each peer, it looks for a suitable tier where the peer's stake does not exceed the remaining stake capacity of that tier (i.e., predefined stake allocation minus the current stake). Once a suitable tier is found, the peer is added to that tier, and the current stake of the tier is updated by adding the peer's stake.

This process continues until all peers are assigned to a tier. The assumption underlying the algorithm is that the total stakes of all peers exactly match the total predefined stake allocations for all tiers, ensuring that every peer is assigned to a tier.

The algorithm ultimately outputs a list of tiers, with each tier containing a group of peers distributed based on the predefined stake allocations. This structured distribution aims to organize peers into tiers reflecting their stakes, ensuring a hierarchical organization within the network.

This algorithm is particularly useful in networks where the stake or contribution of peers plays a critical role in governance, resource allocation, or decision-making processes. Examples include proof-of-stake blockchain networks, where the stake can influence a peer's chances of being chosen to validate new blocks, or in networks where access to resources or voting power is proportional to the stake held by peers.

Algorithm 2 is designed for setting up connections in a P2P network with tiers of peers that have already been distributed. Each peer in the network is intended to connect with a certain percentage of other peers both within their own tier and in the adjacent tiers above and below, based on a specified redundancy percentage.

Algorithm 2: Setup P2P Network Connections with Evenly Distributed Redundancy

Result: Setup of P2P Network Connections with evenly distributed redundancy

Input: tiersWithPeers - List of tiers with peers already distributed,
redundancyPercentage - Percentage of other peers to which a given peer is connected within and across tiers

Output: Structured P2P network with defined connections

```
/* Define connections for each tier based on redundancy percentage
   */
```

1 **for** $tierIndex \leftarrow 0$ ***to*** *length(tiersWithPeers) - 1* **do**

2 currentTier ← tiersWithPeers[tierIndex];

```
/* Connect peers within the same tier                              */
```

3 EvenlyDistributeConnections(currentTier, redundancyPercentage);

```
/* Connect peers with the higher tier if it exists                 */
```

4 **if** *tierIndex > 0* **then**

5 upperTier ← tiersWithPeers[tierIndex - 1];

6 EvenlyDistributeCrossTierConnections(currentTier, upperTier, redundancyPercentage);

7 **end**

```
/* Connect peers with the lower tier if it exists                  */
```

8 **if** *tierIndex < length(tiersWithPeers) - 1* **then**

9 lowerTier ← tiersWithPeers[tierIndex + 1];

10 EvenlyDistributeCrossTierConnections(currentTier, lowerTier, redundancyPercentage);

11 **end**

12 **end**

The algorithm takes two inputs: a list of tiers with peers that have already been distributed (tiersWithPeers) and a percentage (redundancyPercentage) that defines the extent of connection redundancy both within the same tier and across different tiers. This redundancy ensures network resilience by providing multiple communication pathways between peers.

The main process involves iterating through each tier of the P2P network. For every tier, the algorithm ensures that each peer is connected to a calculated portion of other peers within the same tier, achieving internal redundancy. This is done through the EvenlyDistributeConnections function, which organizes connections so that they are distributed evenly among peers to prevent any single point of failure within the tier.

Additionally, the algorithm establishes connections between tiers: each peer in a given tier is also connected to a proportion of peers in the immediately higher and lower tiers, if such tiers exist. This is managed by the EvenlyDistributeCrossTierConnections function. If a higher tier is present (for all but the topmost tier), the algorithm connects each peer

to a portion of the peers in that higher tier. Similarly, if a lower tier exists (for all but the bottommost tier), it ensures that each peer is connected to a portion of the peers in that lower tier, according to the defined redundancyPercentage.

The purpose of these tier-to-tier connections is to ensure the P2P network is robust and can handle failures or disconnections without disrupting the overall network connectivity. By establishing these structured connections based on the redundancy percentage, the algorithm aims to balance load and enhance the fault tolerance of the network, ensuring that communication remains possible even if some connections or peers fail.

In many P2P networks, peers may join or leave dynamically. Hence, the algorithm should be run regularly and the network structure adjusted accordingly, by adding and removing connections while maintaining the overall integrity and redundancy of the network. This dynamic adjustment is vital for the network's adaptability and long-term sustainability.

The output of this process is a structured P2P network where each peer is connected to a predefined number of peers in adjacent tiers. This structure ensures redundancy and efficient network communication, which is essential for the network's effectiveness.

Such an algorithm finds its application in scenarios where network resilience is critical. It is particularly useful in distributed systems like blockchain networks, file-sharing systems, and decentralized data storage solutions, where reliable and efficient peer-to-peer communication is essential for the system's stability and performance.

2.2.2 Traffic Prioritization

Quality of Service (QoS) is a term used in various fields, including computer networking to describe the performance level of a service or system, especially in terms of the user's experience. It is one key component of performance that is often overlooked in the blockchain industry, where raw metrics like throughput and latency are given more attention. However, a system that even though it may have a high performance this is not enough if it is unreliable and only services a small set of users. Since computational resources are finite, the network needs to have a prioritization system that prevents the system from becoming overloaded when the demand exceeds those resources.

In the realm of digital transactions, particularly within permissionless decentralized networks, the challenge of executing a multitude of transactions concurrently without sacrificing speed or security is a pressing concern. Decentralized networks distribute trust without a central overseer, making the timely validation of transactions critical. As such networks become more popular, they often struggle with issues like network congestion, unpredictable transaction times, and fluctuating fees.

A sophisticated system for transaction prioritization is not just a nice-to-have; it's essential. As network traffic grows, being able to prioritize transactions effectively ensures that

resources are used efficiently, which can lead to quicker confirmations and a better overall user experience. Economically, an effective prioritization system can also help stabilize transaction fees, thereby enhancing the attractiveness of the network.

The challenge becomes acute when transaction volumes exceed a network's capacity, potentially leading to backlogs and delays. Traditional solutions have typically been rigid, lacking the flexibility to adjust to changing network demands, which can lead to inefficiencies and unfairness. Therefore, the crucial question is how a network based on delegated stakes can maintain smooth and equitable operations, even during periods of high transaction volume.

This section proposes a new transaction prioritization system specifically designed for networks that use delegated stakes. Our efforts seek to reconcile the robust theoretical foundation with the realities of implementation, ensuring that networks with delegated stakes remain efficient without being overwhelmed by transaction backlogs.

One universal challenge across transaction systems is balancing efficiency with fairness. Prioritization based on fees can unintentionally deprioritize low-fee transactions, thereby possibly excluding some network participants. Additionally, static prioritization models struggle with sudden increases in transaction demand, often resulting in congestion. The dynamic and stake-based nature of delegated networks particularly accentuates these challenges, calling for innovative solutions.

Our novel architectural framework is meticulously crafted to optimize transaction prioritization within delegated stake networks without using fees. It is designed to ensure a seamless flow of transactions, maximizing the throughput of the network without causing congestion. The essence of this approach is to keep the network humming along at its full potential without leading to a buildup of pending transactions.

The bedrock of our proposition comprises the following central elements:

- Stake-Based Peer Prioritization: Tying each peer's transaction prioritization directly to their stake in the network.
- Adaptive Broadcasting Rate: Modulating the flow of transactions in response to the current state of the network.
- Service Level Agreements (SLAs): Every node on the network is guaranteed to have a level of service proportional to its delegated stake.
- Capped Prioritization: Even peers with significant stakes face a cap on their transaction broadcast rates to ensure fairness.
- Regular Stake Assessments: Stake evaluations are conducted periodically to keep the prioritization current and fair.

We design a system where each network participant is assigned a unique digital identity, in this case an authenticated public key. The priority level for transaction broadcasting is ascertained by the size of the stake associated with each peer's public key. Thus, the more significant the stake, the higher the transaction processing priority, reflecting a deeper vested interest in the network's health and longevity.

The concept of stake-based peer prioritization is anchored in the idea that a stake in the network is directly proportional to the commitment to its welfare. As such, individuals or entities with a more substantial stake are likely to contribute positively, thereby reinforcing the stability and robustness of the network.

Our system envisions a transparent and immutable mapping of peers to their public keys, thereby creating a clear, tamper-resistant ledger of stakes which simplifies the ranking and prioritization process.

Upon this mapping, peers are categorized into tiers based on their delegated stakes, allowing the network to swiftly assign priority during peak transaction periods. This system not only motivates constructive behavior due to the higher risk of loss for stakeholders but also ensures that resources are strategically allocated, sidelining potentially damaging nodes with a lower stake and commitment level.

The operational protocol activates a control mechanism when transaction traffic is high, whereby nodes manage the flow by reducing the broadcast rate, starting from peers with lower priority. This regulation continues until the system stabilizes, thereby averting a logjam of transactions.

Unlike static systems, our adaptive rate adjusts in accordance with real-time network conditions. It is sensitive to the volume of pending transactions, mitigating possible backlogs by reducing the rate of transaction entry, commencing with those held by peers of lesser priority.

This dynamic method guarantees that the network processes the maximum number of transactions without overtaxing its capabilities. The system remains fair, as lower-priority transactions are delayed but not denied during high traffic periods. The network stays robust, with safeguards preventing unchecked blocks from reaching a critical volume.

In summary, our system represents a symphony of elements working in concert: the rate of incoming messages and the broadcasting rate dynamically adjusts with the network's conditions, and stake-based prioritization fine-tunes messages flow. Together, they form a cohesive mechanism ensuring messages are processed effectively, maintaining the health and equilibrium of the system.

To design a dynamic peer management system focused on preventing system overload, identified by the accumulation of unprocessed messages, the system must adjust both bandwidth allocation and peer connections based on real-time network performance metrics.

Initially, each node begins operations with a set number of peer connections, accompanied by specific bandwidth allocations. This includes a base bandwidth for standard operations and a minimum threshold critical for sustaining network stability.

The core of the system is to monitor the accumulation of unprocessed messages continuously. If there's an increase in unprocessed messages, indicating potential system overload, the system implements a protocol to reduce the allocated bandwidth. This reduction is performed incrementally, continuing until the accumulation of unprocessed messages stabilizes or the minimum bandwidth threshold is reached.

When the minimum bandwidth level is reached without stabilizing the unprocessed messages, the node begins to disconnect from peers, starting with those with the lowest stakes. This disconnection process progresses up the stake hierarchy until the accumulation of unprocessed messages is under control.

During periods when the number of unprocessed messages falls and system stability is achieved, the node gradually reestablishes connections with peers, giving priority to those with higher stakes, and increases its bandwidth allocation back to the base level. The system also has the capacity to scale up by adding more peers and enhancing bandwidth allocation, especially when the node operates under capacity, indicated by a consistently low number of unprocessed messages.

The system's design allows for dynamic adjustments based on network conditions, stake distribution, and individual node performance. Periodic evaluations of peers, considering their stakes, past performance, and contribution to managing unprocessed messages, are integral to this dynamic adaptation.

The primary goal of this system is to optimize bandwidth consumption by maintaining a defined level of incoming messages based on the tiers of connected peers and the total available bandwidth. Secondary objectives include preserving a fair and decentralized network structure and ensuring no single node becomes a bottleneck in message processing. This approach is formalized in algorithm 3, ensuring network robustness and equitable functionality.

Algorithm 3: Dynamic Tier-Based Bandwidth Consumption Management

Result: Adjust peers' bandwidth in tiers according to total available bandwidth
Input: defaultTotalBandwidth, totalBandwidth
Output: Adjusted bandwidth per peer in each tier based on total bandwidth comparison to default settings

```
1  while node is active do
2      if totalBandwidth > defaultTotalBandwidth then
           /* Decrease bandwidth starting from the lowest tier to the
              highest                                                  */
3          DecreasePeersBandwidth(peer);
4      else
5          if totalBandwidth < defaultTotalBandwidth then
               /* Increase bandwidth starting from the highest tier to the
                  lowest                                               */
6              IncreasePeersBandwidth(peer);
7          end
8      end
9      sleep(adjustmentInterval);
10 end
```

Initially, the algorithm sets a default bandwidth for each peer according to its predefined tier. This step ensures that every peer starts with a standard bandwidth allocation corresponding to its importance or role within the system, as determined by its tier.

The total bandwidth usage is then monitored by counting the bytes of the incoming messages. Based on this, the algorithm decides whether to adjust peer bandwidths. If the total bandwidth is greater than the predefined default total bandwidth, the algorithm redistributes the available bandwidth starting from peers in the lowest tier to those in the highest. This approach ensures that all peers, especially those in lower tiers, are guaranteed a minimum level of service before allocating surplus bandwidth to higher tiers.

Conversely, if the total bandwidth is less than the default, the algorithm increases the bandwidth allocation for peers, beginning with those in the highest tier and moving down to the lowest. This prioritizes higher-tiered peers, ensuring they receive adequate resources, before addressing the needs of lower-tiered peers.

The process repeats at regular intervals, as defined by the 'adjustmentInterval', allowing the system to adapt to changes in bandwidth usage and maintain balanced and efficient operations across all peer tiers.

The challenge lies in determining the optimal default levels of bandwidth for each node to prevent underutilization and overload of the network. This balance is essential for a healthy network flow and requires empirical testing through real-world simulations or analysis of historical data. Machine learning models can be employed to predict optimal

settings based on changing network conditions and transaction volumes.

Concerning outbound traffic, there are two main types:

- Point-to-Point: broadcast the message to all connected peers.
- Gossip: broadcast the message to only a random subset of the connected peers.

This traffic management system is designed to optimize the speed and reach of transaction dissemination and consensus message propagation within a tiered network. It seeks to balance the need for rapid confirmation by higher-stake peers with the robustness and fault tolerance provided by the gossip protocol within and across peer tiers. The dual strategy ensures that while transactions are quickly brought to the attention of influential nodes, they are also widely circulated among all network participants, fostering a democratic and resilient network environment.

Algorithm 4 is designed to estimate the probability that a given peer within a peer-to-peer network receives all messages from connected peers across different tiers taking into account the different types of traffic per tier, such as gossip (where messages are sent to only a fraction of the total connected peers) and point-to-point (where messages are sent to every connected peer), we introduce a new dimension to our calculations based on the communication style in each tier.

The algorithm starts by initializing a probability map and iterates over each peer within the provided network structure. For every peer, it maintains a list of tier probabilities, representing the chances of receiving all messages considering the tier's communication method.

As the algorithm evaluates each tier applicable to the peer (the peer's own tier, the one immediately above, and the one immediately below), it distinguishes between the gossip and point-to-point communication styles. For gossip-based communication, it calculates the probability that the peer receives a message based on the fraction of total connected peers it is expected to communicate with, adjusting this calculation according to the specific gossip protocol parameters. For point-to-point communication, it assumes messages are sent to every connected peer, leading to a different set of probabilities based on direct connections.

For each tier, depending on its traffic type, the algorithm determines the likelihood that the peer successfully receives messages from that tier, influenced by the failure probability 'p' and the nature of traffic (gossip or point-to-point). This computed probability for each tier is then aggregated into the tier probabilities list.

After processing all relevant tiers, the algorithm calculates the overall probability that the peer receives all necessary communications across tiers. This overall probability, reflecting the combined impact of different communication styles and the network structure, is assigned to the corresponding peer in the probability map.

Algorithm 4: Probabilities of a Peer Receiving All Messages

```
1  Function CalculateProbabilityOfFullMessageReceipt(peersWithConnections, p,
   trafficTypePerTier):
       /* Initialize an empty map to store probabilities for each peer
          */
2      Initialize an empty map called probabilityMap
3      for each peer in peersWithConnections do
           /* Calculate probabilities for each tier considering the
              connections and traffic type                              */
4          Initialize an empty list called tierProbabilities
5          for each tier in [peer's tier, higher tier, lower tier] do
6              if tier exists then
7                  connectedPeers ← number of peer's connections in this tier
8                  totalPeers ← total number of peers in this tier
9                  trafficType ← trafficTypePerTier[tier]
10                 tierProbability ←
                    calculateProbability(connectedPeers, totalPeers, trafficType);
                    Append tierProbability to tierProbabilities
11             end
12         end
           /* Calculate the overall probability for the peer receiving
              all messages                                              */
13         overallProbability ← ∏_{prob∈tierProbabilities} prob
           /* Store the calculated overall probability for each peer    */
14         probabilityMap[peer] ← overallProbability
15     end
16     return probabilityMap
17 End Function
```

Once every peer is assessed, the algorithm concludes by returning the probability map, reflecting the diversity in communication styles across the network's tiers and considering the message transmission's failure rate. This updated map offers a more detailed view of each peer's likelihood of successfully receiving all messages, tailored to the specific dynamics of gossip and point-to-point traffic within the network.

In a broader context, this algorithm serves as a tool for evaluating the robustness and efficiency of communication within a network structured in layers. By incorporating factors like network size, message distribution strategy, and the reliability of message transmission, it offers insights into how well the network performs in ensuring information dissemination to all its intended recipients. This can be particularly useful in scenarios where complete coverage or widespread information sharing is critical, such as in sensor networks, distributed computing systems, or information propagation in decentralized networks.

Algorithm 5 calculates the bandwidth cost for a given peer based on the traffic strategy

employed in each tier of a peer-to-peer network. The inputs include a list detailing the tiers along with peers and their corresponding traffic types, a map defining the traffic strategy used by a specific peer for each tier (either 'gossip' or 'point-to-point'), and the base bandwidth cost per message.

Algorithm 5: Calculate Bandwidth Cost Based on Peer Traffic Strategy

Result: Calculate bandwidth cost for a given peer traffic strategy
Input: tiersWithPeers - List of tiers with peers and their traffic type,
peerTrafficStrategy - Traffic type used in each tier by a specific peer,
baseCost - Base bandwidth cost per message
Output: Total bandwidth cost for the specified peer

```
1  for each tier in tiersWithPeers do
2      tierTrafficType ← peerTrafficStrategy[tier];
3      numPeersInTier ← countPeersInTier(tier);
4      if tierTrafficType == 'gossip' then
           /* Assume gossip traffic involves communicating with a
              fraction of peers                                    */
5          gossipFraction ← getGossipFraction(tier);
6          numPeersContacted ← gossipFraction * numPeersInTier;
7          tierBandwidthCost ← numPeersContacted * baseCost;
8      else
           /* Point-to-point traffic involves communicating with all
              peers                                                */
9          tierBandwidthCost ← numPeersInTier * baseCost;
10     end
       /* Add the bandwidth cost of this tier to the total bandwidth
          cost                                                     */
11     totalBandwidthCost ← totalBandwidthCost + tierBandwidthCost;
12 end
13 return totalBandwidthCost;
```

Initially, the algorithm sets the total bandwidth cost to zero. It then proceeds to examine each tier in which the peer operates. For every tier, the algorithm identifies the traffic type chosen by the peer according to the peerTrafficStrategy input.

If the traffic type for the current tier is determined to be 'gossip', the algorithm calculates the number of peers to be contacted based on a specific gossip fraction relevant to that tier. This fraction represents the percentage of total peers in the tier with whom the peer will communicate under the gossip protocol. The number of peers contacted is then multiplied by the base cost per message to derive the bandwidth cost for operating under the gossip strategy in that tier.

Conversely, if the traffic type is 'point-to-point', the algorithm assumes that the peer com-

municates with every other peer in the tier. Therefore, it calculates the bandwidth cost by multiplying the total number of peers in the tier by the base cost per message, reflecting the more extensive communication scope of point-to-point interactions.

After determining the bandwidth cost for the peer in the current tier, the algorithm adds this cost to the total bandwidth cost. This process repeats for each tier in which the peer is active, aggregating the costs.

Upon completing the iteration across all tiers, the algorithm returns the total bandwidth cost, which encapsulates the cumulative bandwidth required for the peer to execute its operations across the network, adhering to its specified traffic strategies in each tier. This final output provides a comprehensive view of the peer's bandwidth needs based on its interaction patterns within the network's tiered structure.

2.3 Horizontal Scaling

In the context of Delegated Proof of Stake (DPoS) based networks, horizontal scaling presents a viable solution to enhance network efficiency and manageability. This approach involves dividing the network into multiple shards, each corresponding to equal divisions of the key space. In such a system, each delegate is responsible for managing a number of shards, with each shard tasked with reaching consensus on transactions whose sender accounts fall within its specific key space. This division of labor not only streamlines transaction processing but also contributes to the overall scalability of the network.

In figure 2.3 we can observe a typical sharding architecture, which can be seen as a division of the network into n shards, each shard corresponding to a subnetwork. Each node is allocated to a shard and stores the fraction of the ledger that corresponds to that shard. This design looks simple in theory, but in practice it is complex to implement, because it requires epochs, randomness and protocol reconfiguration, since shards need to change from time to time for security reasons.

We propose a different and new approach in figure 2.4, a deterministic one instead of randomized. The idea is to shard only the database, but not the consensus. In other words, every node is present on every shard with a peer, participating in the consensus of that shard, but each peer only stores the fraction of the ledger that corresponds to that shard.

We start by dividing the network into n equal parts corresponding to n parts of the key space. For example, suppose there are four groups of accounts: the first starts with A, the second with B, the third with C and the fourth with D. Each group corresponds to a shard ans since a transaction involves two parties, the sender and the receiver can be on different shards, so the shard of the sender is responsible for reaching consensus on that transaction.

A critical aspect of this design is the handling of scenarios where a shard goes offline.

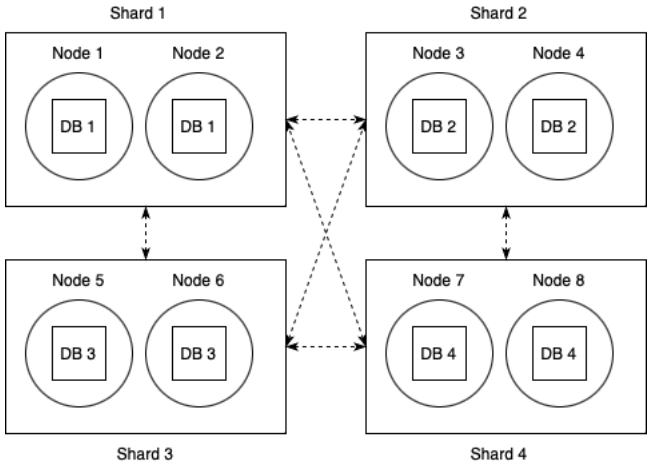

Figure 2.3: Typical Sharding Architecture

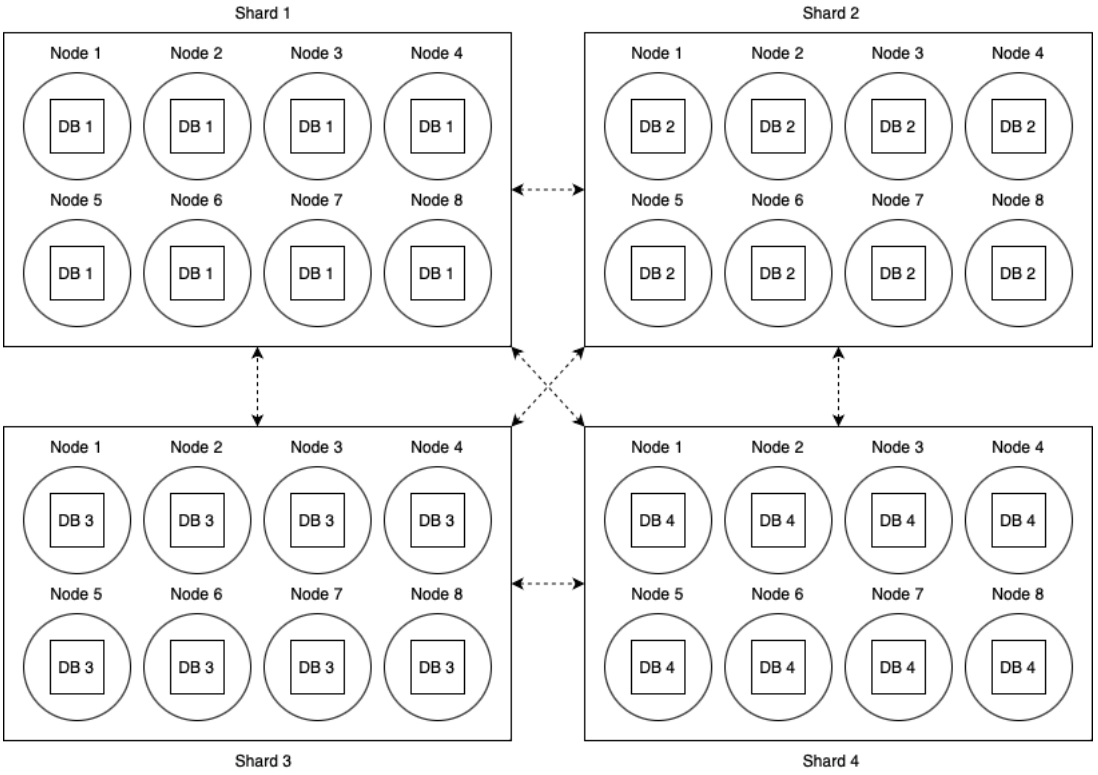

Figure 2.4: New Sharding Architecture

In such cases, another shard inherits the key space of the offline shard, thereby ensuring continuity in transaction processing. This mechanism requires a robust protocol to prevent conflicting consensus messages emanating from the same delegate, particularly when there is a transition of key space responsibility.

The CAP theorem [GL02], which posits that it is impossible for a distributed system to simultaneously guarantee consistency, availability, and partition tolerance, plays a significant role in the design of such networks. Given this constraint, our proposed system opts to prioritize consistency over availability. This is particularly evident in the handling of consensus messages, which are stored in a distributed database that favors consistency.

When a shard goes offline, a fallback mechanism is needed to maintain service, which is described in algorithm 6. The core idea is that another shard inherits its key space, and the new shard queries the database to ascertain whether the previous shard had already produced a consensus message for any pending transactions. In the absence of a response from the database – perhaps due to its unavailability – the new shard refrains from producing any consensus message. This approach mitigates the risk of conflicting messages and preserves the integrity of the consensus process. In the worst-case scenario, where the database remains inaccessible or the original shard does not come back online, the consensus instance may time out. This design decision, favoring consistency, may impact the network's availability but is critical for maintaining the accuracy and reliability of the transaction records in the DPoS system.

By adopting this architecture, DPoS-based networks can achieve horizontal scaling while navigating the inherent challenges posed by the CAP theorem. This balance between consistency, availability, and partition tolerance is crucial for the effective functioning of a decentralized, sharded network.

Algorithm 6: DPoS Shard Fallback Algorithm

```
Data: shards - the list of shards, offlineShard - the shard that went offline
Result: Handle the fallback of the offline shard
Procedure ShardFallback(shards, offlineShard):
    if offlineShard is in shards then
        newShard ← SelectFallbackShard(shards, offlineShard)
        InheritKeySpace(newShard, offlineShard)
        PropagateConsensusMsg(newShard)
    end
return
Function SelectFallbackShard(shards, offlineShard):
    for each shard in shards do
        if shard ≠ offlineShard then
            return shard
        end
    end
Procedure InheritKeySpace(newShard, offlineShard):
    newShard.keySpace ← offlineShard.keySpace
return
Procedure PropagateConsensusMsg(shard):
    CheckDatabaseForConsensus(shard)
    // Implement message handling
return
```

2.4 Conclusion

In concluding the chapter on the Communication Layer, it is essential to emphasize the critical importance of scalability in the context of a global permissionless decentralized network. Scalability is not merely a technical requirement; it is the lifeblood that enables these networks to serve a global user base effectively and efficiently. In this regard, the chapter's focus on vertical and horizontal scaling approaches—namely, peer management and traffic prioritization for vertical scaling, and database sharding for horizontal scaling—becomes particularly salient.

Vertical scaling, with its emphasis on peer management and traffic prioritization, addresses the need to optimize the performance of individual nodes. This optimization is crucial for sustaining high transaction throughput and ensuring reliable network performance, particularly in scenarios of varying or unpredictable network loads. By improving the capacity and efficiency of each node, vertical scaling ensures that the network can handle increased demands without compromising on speed or reliability, a key consideration for a global system that must remain operational and responsive at all times.

Horizontal scaling, on the other hand, is approached through database sharding — a technique that involves dividing the network's database into smaller, more manageable parts, or 'shards.' This approach is particularly well-suited to permissionless decentralized networks, as it allows the network to expand its infrastructure by adding more nodes, each handling a portion of the overall data. This method of scaling is crucial for maintaining network performance and efficiency as the user base grows. Database sharding enables the network to scale outwards, accommodating more transactions and participants, while preventing any single node from becoming a bottleneck.

The combination of vertical and horizontal scaling strategies is therefore of paramount importance for a global permissionless decentralized network. These approaches ensure that the network remains robust, efficient, and capable of handling the demands of a growing global user base. By focusing on both individual node optimization and the expansion of network capacity, the research presented in this chapter lays a solid foundation for addressing the scalability challenges inherent in decentralized digital currencies, thereby facilitating their evolution into truly global financial systems.

Chapter 3

Transaction Layer

3.1 Introduction

In this chapter, we explore a conceptual framework focused on the transaction layer within digital payments networks, emphasizing its role in managing network state. A transaction in any network is essentially a proposed alteration to the network's state, necessitating a robust mechanism for ensuring integrity and validity.

In payments networks, the transaction layer is crucial for verifying state modifications. This process involves adherence to specific rules, like preventing unauthorized new coin creation, thus serving as a guardian of the network's state through transaction validation and execution. The execution process computes and actualizes the network's new state after modification.

A significant challenge in this domain is balancing transaction validation with anonymity preservation. Traditional structuring of transaction data and validation rules may expose sensitive information, compromising participant anonymity. Consequently, redefining transaction data and rule construction to protect sensitive information is critical.

The chapter aims to develop a scheme that maintains transactional party anonymity while integrating with the Delegated Proof of Stake (DPoS) mechanism, a more energy-efficient and democratic alternative to traditional proof of work systems. This scheme is intended to leverage DPoS strengths while addressing privacy concerns in transactional data processing.

The goal is to balance transaction validation, network state management, and user anonymity within a DPoS-based payments network, enhancing security and privacy in digital transactions and aligning with blockchain technology's evolving paradigms.

The transaction layer in payments networks is essential for secure and accurate transaction processing. It acts as an intermediary between the application layer, where transactions are created, and the storage layer, where they are stored, and usually has the architecture of one of the two models: the account model and the Unspent Transaction Output (UTXO) model.

The account model maintains a global state of accounts, with each account having a balance. Transactions transfer values between accounts, adjusting their balances. This model supports complex smart contracts and straightforward state transitions but raises privacy

concerns due to the ease of tracking financial flows and potential challenges in parallel transaction execution.

The UTXO model treats transactions as inputs and outputs. Inputs reference outputs from previous transactions, and outputs define cryptocurrency quantity and spending conditions. Balances are calculated by summing associated UTXOs. While more complex, the UTXO Model enhances anonymity and supports parallel transaction execution due to the independence of each transaction output.

In this chapter we use the term privacy in a general way to describe hiding some information and anonymity to specifically refer to the way that data is hidden.

3.1.1 Problem Statement

Our goal in this chapter is to investigate how to design a transaction layer that provides anonymity and is compatible with DPoS, allowing nodes to verify transactions without accessing specific information, such as the sender's account, transaction amount, or receiver's account.

We start by proposing a general framework for proving and updating the delegated stakes on the network without affecting anonymity. In the following section "Signer Anonymity" we propose a new Decoy Selection Algorithm (DSA) for the UTXO model that can be of independent interest, even for networks that do not use DPoS, since most of research in this field focus on the study of the linkable ring signature as cryptographic primitive, and treat the selection of the inputs of the ring signature as a blackbox.

Finally, we propose two different techniques to achieve transaction amount anonymity and to also know the total delegated stakes of the nodes on the network whilst at the same time being compatible with both the account model and the UTXO model. One if these techniques is based on Multi-Party Computation (MPC) and the other is based on the Diffie-Hellman Key Exchange (DFKE).

3.1.2 Related Work

In the realm of blockchain technology, particularly focusing on Proof-of-Stake (PoS) and its variant, Delegated Proof-of-Stake (DPoS), there has been a significant emphasis on addressing the challenges of anonymity. The scholarly contributions in this field have been instrumental in advancing our understanding of how these decentralized systems can be improved to offer greater security and privacy.

One of the cornerstone studies in this domain is by Chenghong Wang and Kartik Nayak, who have analyzed the intricacies of embedding anonymity into PoS blockchain mechanisms. They have critiqued the foundational limitations of Bitcoin's Proof-of-Work (PoW) protocol, as originally introduced by Nakamoto, particularly highlighting its high energy

consumption and the public nature of its transactions. This critique paved the way for exploring more sustainable and private alternatives, such as PoS, which promises enhanced energy efficiency and stake-based leadership integrity. However, Wang and Nayak have astutely noted the privacy challenges inherent in PoS systems, specifically the risk of stake-related information disclosure [WPNM23].

In addressing these challenges, the authors have proposed potential strategies with unique trade-offs. They discuss the concept of equal weight assignment to all participants, which enhances privacy but raises safety concerns. They also explore differentially-private stake distortion as a method to balance safety and privacy, despite its security compromises. Their empirical analysis, including case studies like Ethereum 2.0, provides valuable insights into the real-world applicability of these privacy solutions.

Their exploration delves into the complexities of maintaining transaction confidentiality in PoS systems. They have scrutinized existing privacy-oriented protocols, such as Ouroboros Crypsinous [KKKZ19a], revealing vulnerabilities under targeted privacy attacks. Wang and Nayak's analysis is enriched by referencing the works of Kohlweiss, Madathil et al. and their own studies, highlighting the susceptibility of PoS systems to attacks like the Reverse Tagging Attack (RTA) and the Stake Inference Attack (SIA). These attacks can exploit deterministic and probabilistic liveness conditions to infer stake information, thus compromising anonymity [KMNS21].

Parallel to this, significant advancements have been made in DPoS systems. A noteworthy contribution is the "DT-DPoS: A Delegated Proof of Stake Consensus Algorithm with Dynamic Trust," which integrates a dynamic trust model into the traditional DPoS system. This model calculates the trustworthiness of nodes based on their transaction history, aiming to enhance the reliability of witness nodes and mitigate the risk of collusion attacks. The paper, through comprehensive performance analysis, establishes the superiority of DT-DPoS in terms of security and scalability compared to PoW, PoS, and traditional DPoS [SYYY21].

Another work in this field is the implementation of the DPoPS consensus mechanism within the X-Cash blockchain, which adapts DPoS for privacy coins and introduces a unique methodology for consensus in environments with private transactions [CBHG19]. The paper discusses the integration of Verifiable Random Functions (VRF) for block producer selection and a DBFT mechanism, along with the role of reserve proofs in staking for privacy coins.

The comparative analysis of these studies reveals distinct approaches to enhancing DPoS. While DT-DPoS focuses on incorporating dynamic trust into witness node selection, DPoPS adapts DPoS for privacy coins, maintaining transaction anonymity and stake privacy. The technical innovations in DT-DPoS, such as the trust model based on transaction history and ring signature schemes, aim to enhance privacy and security. Conversely, DPoPS introduces VRFs and a DBFT mechanism tailored for privacy coins within

the DPoS framework.

In summary, these scholarly endeavors exemplify the ongoing evolution and diversification of blockchain consensus mechanisms, showcasing innovative methods to enhance security, privacy, and efficiency in decentralized networks, particularly in DPoS systems. This body of work not only contributes to the theoretical understanding of blockchain privacy but also sets the stage for future explorations and innovations in this rapidly evolving field.

3.2 General Framework

In this section we outline the general framework for providing anonymity to the transaction layer compatible with Delegated Proof of Stake. Generally, full anonymity in a digital currency implies anonymity in all components of a transaction life cycle that can leak information about the transaction itself, namely the sender, the amount and the receiver. So, we need a way for the network to know the delegated stakes of the nodes and how they change through time without breaking anonymity.

We start by intuiting that we only need to know the total delegated stake of each node, not the individual delegated stake of each account. Since a transaction represents a transfer of ownership between two parties (the sender and the receiver) of one or more coins, and each coin is associated with a specific delegate, if a coin is represented by a data structure with additive homomorphic properties [KL14] that hides its amount, we can group all the coins that have the same delegate in a group and reveal only the total amount of that group.

In section amount anonymity we propose two different ways to achieve this, each one with its advantages and disadvantages: one is by having a group of special nodes (the ones with the most delegated stake) on the network to produce a shared pair of keys that are used to encrypt all the coins on the network and then jointly decrypt only the total delegated stakes of each node; the other is by having the delegate itself prove its own total delegated stake. The first is more computationally intensive and requires interaction between nodes, but no single node gets to know the individual stake of a given account. The second is more simple and efficient, but each delegated gets to know the stake of each individual account that delegates to it.

To achieve sender anonymity in cryptocurrency transactions, a prevalent method involves utilizing decoy inputs in conjunction with linkable ring signatures. This approach ensures that the true input used in a transaction remains indistinguishable within a set of potential inputs, thereby obscuring the link between the sender and the transaction outputs.

Each coin, represented as an unspent transaction output (UTXO), is associated with a unique identifier, such as a tag or serial number. This unique identifier is crucial for pre-

venting double-spending, as it ensures that once a UTXO is consumed in a transaction, it cannot be reused in future transactions. The system relies on the integrity of these unique identifiers to maintain the consistency and security of the ledger.

Linkable ring signatures serve as the cryptographic primitive facilitating anonymity while maintaining the integrity of the transactional system. Originally introduced in cryptographic literature, ring signatures enable a signer to produce a signature on behalf of a group (the ring) without revealing which member of the group actually generated the signature. The linkable property allows for the detection of signatures produced by the same signer without compromising anonymity. This is essential for preventing double-spending, as it enables the network to link transactions that attempt to spend the same input more than once.

The process operates as follows. For each real input that the sender intends to spend, the sender selects decoy inputs from other unspent transaction outputs available on the blockchain. These decoy inputs are legitimate UTXOs belonging to other users but are included solely to enhance anonymity. The real input and the selected decoy inputs are aggregated to form a ring of size . This ring constitutes the anonymity set for the transaction. Within this set, all inputs are treated equivalently, making it computationally infeasible for an observer to distinguish the real input from the decoys.

The sender generates a linkable ring signature on the transaction message using their private key corresponding to the real input and the public keys associated with all inputs in the ring. The signature satisfies several critical properties: anonymity, unforgeability, and linkability. Anonymity ensures that it is impossible to determine which member of the ring produced the signature. Unforgeability guarantees that only a holder of a private key corresponding to one of the public keys in the ring can produce a valid signature. The linkability property allows any two signatures produced using the same private key to be linked together, enabling the detection of double-spending attempts without revealing the signer's identity.

The ring signature allows network participants to verify that the transaction is authorized by one of the private keys associated with the ring's public keys, without knowing which one. The verification process confirms the validity of the signature and ensures that the key image (a cryptographic construct derived from the private key) has not been used in any previous transaction, thus preventing double-spending.

The anonymity provided by this mechanism is directly related to the ring size . A larger ring size increases the anonymity set, thereby enhancing the sender's anonymity. As grows, the probability of correctly identifying the real input diminishes, making it increasingly difficult for an adversary to link transactions to the sender. However, increasing the ring size also impacts computational efficiency. Larger rings require more computational resources for the generation and verification of ring signatures. This can result in longer transaction processing times and increased computational overhead for network nodes.

Taking this into account, we need to define an anonymity set that is compatible with DPoS. If the anonymity set is composed of coins with different delegates, there is no way to prove that the a delegated stake is correct without revealing information about transactions. So, we need an anonymity set composed of coins that delegate to that same delegate. In practice, this means that a transaction does not change the total delegated stake of the delegate, it only changes the delegated stakes of individual accounts, but only the first is relevant to consensus. This technique also allows to maintain receiver anonymity. Next, we propose an algorithm that can be used to compute the delegated stake held by a delegate, and a practical example for better understanding.

Algorithm 7: Compute Total Delegated Stake of Delegate D

Input: T (List of all transactions), D (Delegate)
Output: S_D (Delegated stake of Delegate)

1. Initialize $S_D \leftarrow D_a$.
2. Initialize $first \leftarrow$ true.
3. For each transaction $tx \in T$ do:
   (a) If tx is a delegate transaction then:
      i. Set $delegateTo \leftarrow tx.delegateTo$.
      ii. Set $delegateFrom \leftarrow tx.delegateFrom$.
      iii. Set $amount \leftarrow$ Decrypt$(tx.commitment)$.
      iv. If $first$ is true and $delegateTo = D$ then:
         A. Set $S_D \leftarrow amount$.
         B. Set $first \leftarrow$ false.
      v. If $first$ is false then:
         A. If $delegateTo = D$ then set $S_D \leftarrow S_D + amount$.
         B. If $delegateFrom = D$ then set $S_D \leftarrow S_D - amount$.
4. Return S_D.

The algorithm 7 starts with the amount of D (D_a), which delegates to itself by default, and then finds the first output in the ledger that was delegated to D, where D is the account of the delegate. It then finds all subsequent delegate transactions that involve delegate D. If the transaction delegates to A, it adds that amount to the total delegated stake; if the transaction delegates from D, it subtracts that amount to the total delegated stake.

In the example of figure 3.1, there are two types of transactions: spend transactions, with two possible input coins (ring size of two, one real spend) and two output coins; and delegate transactions, which change the delegate of one output. The total delegated stake of A is computed by finding the first output that delegates to it, which in this case is the genesis transaction (amount of 100), and then finding the delegate transactions that involve A: the first delegates 60 to B and the second delegates 10 to B, so the total delegated stake of A would be 30. By the same logic, the total delegated stake of B would be 70. The sum

of the two is equal to the amount of the genesis block.

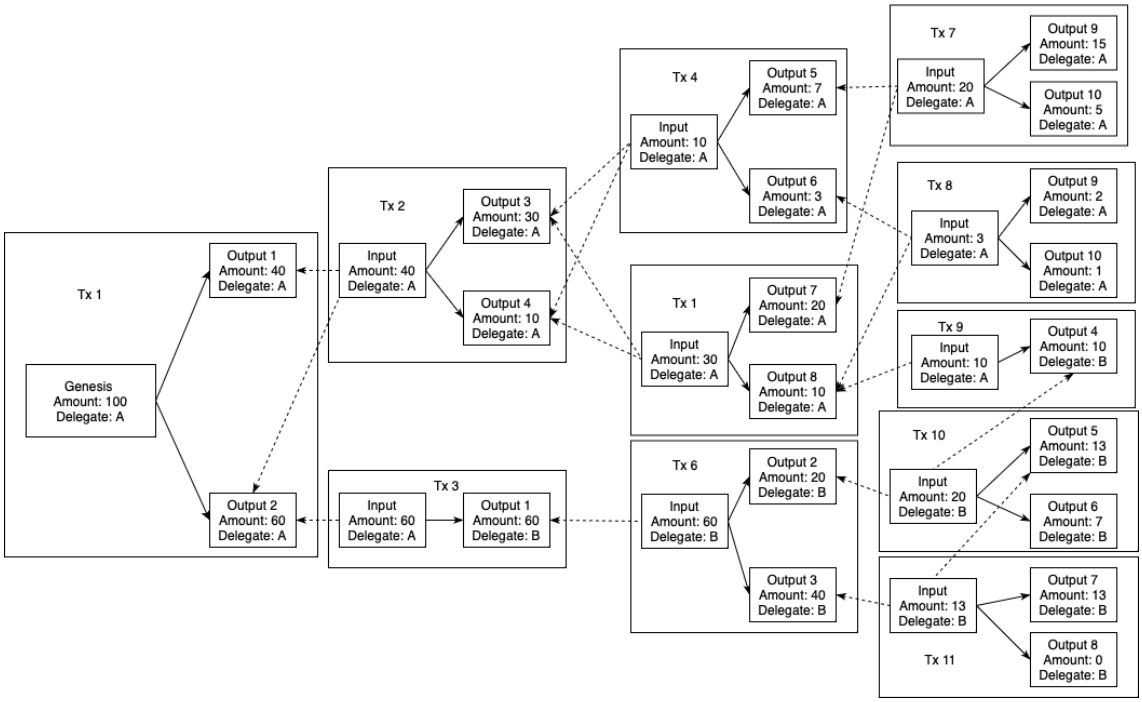

Figure 3.1: Ledger Example

Regardless of the technique used to anonymize the amounts on the network, the total delegated stakes cannot be updated everytime a transaction is made, since that would reveal the amount of that transaction, so this update needs to be done from time to time.

The diagram of figure 3.2 illustrates the process of updating delegated stakes in a proof of stake blockchain system, encompassing activities from the conclusion of one epoch to the commencement of the next. As Epoch 0 unfolds, validators engage in the network's consensus activities, leading to the establishment of a final state at the epoch's closure. This final state is pivotal, as it forms the foundation upon which delegates will base their proof of stake.

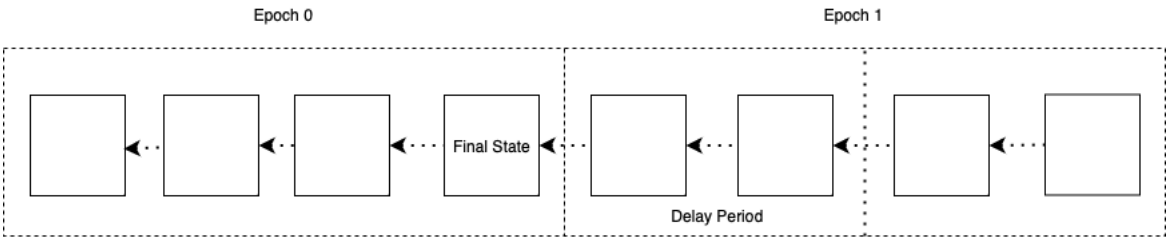

Figure 3.2: Update of the delegated stakes

Following the closure of Epoch 0, there commences a delay period. This intermission is designed to allow delegates ample time to assimilate the final state of the previous epoch and generate their corresponding proof of stake. The proofs are essential for validating the delegates' stakes, which, in turn, are vital for the network's security.

The transition to the new delegated stakes for the ensuing Epoch 1 is contingent on the presentation of proofs of stake from delegates summing 2f+1 of the total stake, where f symbolizes the count of Byzantine nodes that the network can accommodate while still achieving consensus. This requirement is a security measure, ensuring a majority consensus among the nodes and diminishing the likelihood of nefarious influences on the network's integrity.

Post the delay period, nodes retain the ability to update their delegated stakes. This flexibility ensures that nodes that were not primed during the delay period can still contribute to and influence the consensus process.

The delay period is instrumental in mitigating the risk of malicious attacks. It provides a cushion that allows for the thorough preparation and verification of the proofs of stake, thereby safeguarding the transition into the new epoch. By mandating a proof of stake predicated on the final state of the preceding epoch and by stipulating a minimum threshold for the amount of stake that needs validation, the system fortifies its defenses, averting abrupt and potentially malign changes in stake delegation.

3.3 Sender Anonymity

Some privacy focused cryptocurrencies use a method where, instead of just trying to anonymize the actual transactions, they introduce multiple "decoy" transactions (also called mixins) that are meant to confuse anyone trying to trace or identify which transaction is real. For example, in Monero, each transaction is obfuscated by using Ring Signatures, where a real input (the real transaction being spent) is mixed with decoy inputs that are unrelated but drawn from other users' transactions. To an outside observer, it appears as though there are multiple possible inputs (the ring), but only one real input is being used, and it's unclear which one. The goal is that, with a large enough set of decoys, it becomes computationally or statistically very difficult, if not impossible, to trace the real source of a transaction. If the way the decoy transactions are chosen isn't truly random or lacks sufficient entropy (unpredictability), attackers can use statistical analysis or timing analysis to isolate the likely real transaction input from the decoys. One of the most prominent examples of this type of attack comes from Monero's early implementation phases specifically regarding its RingCT (Ring Confidential Transaction) system. It was discovered [MSH+18] that a pattern in how decoys were selected could allow attackers to statistically infer which input in the ring signature was the real one and as a result a large proportion of transactions could be easily deanonymized.

In this section we propose a new Decoy Selection Algorithm (DSA), called SimpleDSA, that can be of independent interest, even for networks that do not use DPoS.

3.3.1 SimpleDSA: A Simple Decoy Selection Algorithm

We adopt a definition of DSA which deterministically chooses ring members according to a pre-determined partition of all output coins from the ledger. It consists of the following algorithms::

- $\mathrm{Setup}(1^\lambda) \to pp$. It takes a security parameter $\lambda \in N$, and outputs the system parameters pp, which include the number of inputs m and the number of outputs n. All algorithms below have implicitly pp as part of their inputs.
- $\mathrm{selectDecoys}(n, m, i) \to pp$. It takes as input the order i of the real output, the number of inputs in a transaction m and the number of outputs in a transaction n, where i is contained in m.

Security model. A Digital Signature Algorithm (DSA) is considered secure if it is computationally infeasible for an adversary to distinguish the real input (the actual coin being spent) from the decoys in a ring signature, except with negligible probability in the security parameter λ. This security definition focuses on preventing an adversary from identifying the true transaction among the set of possible transactions included in the ring.

Various attacks aim to compromise this security by exploiting patterns or information leakage in the ring signature. Two notable variants are:

- **Homogeneity Attack.** It is infeasible for an adversary A to determine the historical transaction t' where the real input of a transaction t originated ($t' < t$), except with negligible probability in the security parameter λ.
- **Chain-reaction Analysis.** It is computationally infeasible for an adversary A to determine if an input (decoy) of a transaction t has been previously consumed in a transaction t' ($t' < t$), except with negligible probability in the security parameter λ.

Choosing the right decoys in a ring signature is thus essential for achieving better anonymity. Ideally, we want a decoy selection algorithm that prevents the homogeneity attack and the chain-reaction analysis as defined in the previous security model, and that does not have a high computation complexity, since we want an user to be able to send and validate a transaction as fast as possible.

Because of that, we propose a new decoy selection algorithm (8) that has only one predefined set of decoys for each real input in a transaction according to the number of inputs and outputs in the transaction and does not depend on past transactions.

It generates a subset of elements (decoys) from a larger set, ensuring that a specific, real element (the actual signer in a ring signature) is included in this subset but is indistinguishable from the decoys.

Algorithm 8: SimpleDSA

Result: Select decoys

```
1  Algorithm select_decoys(number_of_outputs, number_of_inputs,
     real_input_number)
       Input: number_of_outputs, number_of_inputs, real_input_index, Ledger
       Output: A shuffled subsequence
2      current_subsequence ← []
3      v ← 0
4      while real_input_number not in current_subsequence do
5          a ← v
6          for i ← 1 to number_of_inputs do
7              current_subsequence.append(v)
8              v ← v + number_of_outputs
9              if v > real_input_index and len(current_subsequence) ==
                 number_of_inputs then
10                 a ← a + 1
11                 v ← a
12                 break
13             end
14         end
15         if real_input_number in current_subsequence then
16             break
17         end
18         current_subsequence ← []
19     end
20     return current_subsequence
```

The algorithm is based on the idea that the sequence of ordered outputs is divided into buckets of a determined size, which is the number of inputs of a transaction. The outputs within each bucket are separated by that same number. The goal of the algorithm is to find the bucket that corresponds to the real output, and the inputs of the transaction will be the outputs contained in that bucket, including the real one.

For a better understanding we give two simple examples with a single input ring of size two and two outputs, represented in figures 3.3 and 3.4, respectively.

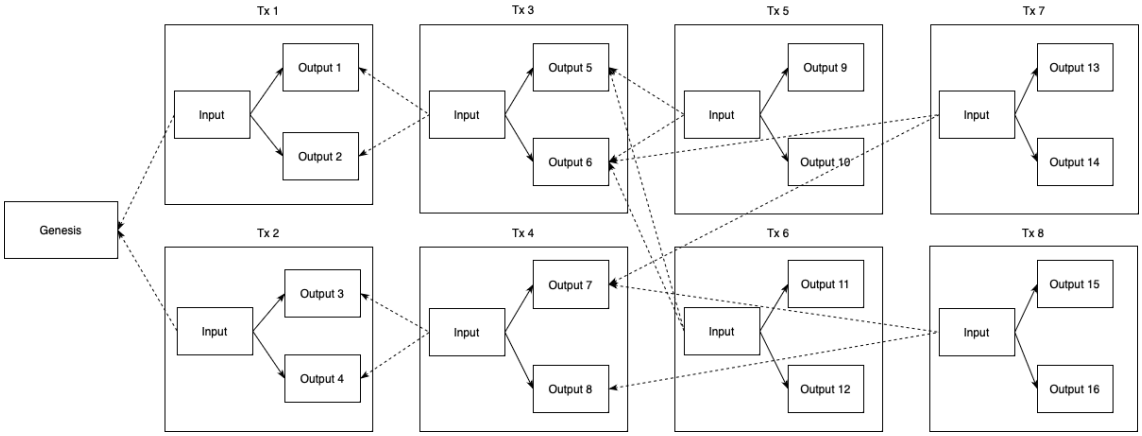

Figure 3.3: Scenario 1

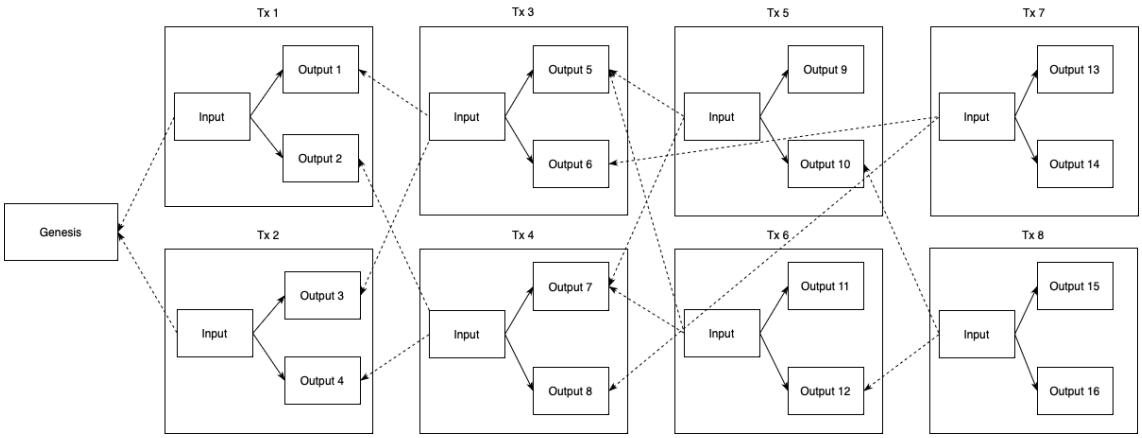

Figure 3.4: Scenario 2

Below we use the notation Transaction X: i,j->k,l to indicate that transaction X has inputs i and j and outputs k and l.

In scenario 1, where inputs are chosen at random, we can have the following transactions:

- Transaction 1: genesis $\rightarrow 1, 2$
- Transaction 2: genesis $\rightarrow 3, 4$
- Transaction 3: $1, 2 \rightarrow 5, 6$

- Transaction 4: $3, 4 \rightarrow 7, 8$
- Transaction 5: $5, 6 \rightarrow 9, 10$
- Transaction 6: $5, 6 \rightarrow 11, 12$
- Transaction 7: $6, 7 \rightarrow 13, 14$
- Transaction 8: $7, 8 \rightarrow 15, 16$

In this scenario we can easily see that transactions 3 and 4 come from transactions 1 and 2, respectively (homogeneity attack). Even worse, it is possible to observe that the input of transaction 7 is the output 7, since either transaction 5 or 6 must have spent the output 6. Because of that, we can infer that the input spent in transaction 8 is the output 8. Hence the name of this attack: chain-reaction analysis.

The ledger is structured as a DAG, where each transaction is independent of the others, however the same weaknesses can be explored if transactions are grouped in blocks, like a blockchain.

On the other hand, we have the following transactions in scenario 2:

- Transaction 1: genesis $\rightarrow 1, 2$
- Transaction 2: genesis $\rightarrow 3, 4$
- Transaction 3: $1, 3 \rightarrow 5, 6$
- Transaction 4: $2, 4 \rightarrow 7, 8$
- Transaction 5: $5, 7 \rightarrow 9, 10$
- Transaction 6: $5, 7 \rightarrow 11, 12$
- Transaction 7: $6, 8 \rightarrow 13, 14$
- Transaction 8: $10, 12 \rightarrow 15, 16$

We can see that none of the previous attacks work and the probability of a given input being spent remains the same (in this case, 50%).

The time complexity of our algorithm is only $O(n)$, since the running time of the algorithm grows linearly with the size of the inputs. The algorithm has the advantage that if all possible ring signatures of a batch have been constructed, we know that all the outputs in the batch have been spent. Because of that we can delete them from the ledger and prune it. The security proofs are presented in the next section.

The disadvantage of SimpleDSA is that the number of inputs and the number of outputs must be fixed, which should not be a problem in practice, since most transactions have

only two outputs and the size of the ring should be irrelevant as long as it is big enough to provide a good level of anonymity. Besides that, you need to wait that a bucket is full to make a new transaction that references that bucket, which can only be a problem if the network has very low usage levels. If that is the case, transactions can be made with the purpose of preserving the flow of the network.

The application of this algorithm to a Delegated Proof of Stake based network requires that each selected input is delegated to the same account, otherwise it would not be possible to update the total delegated stakes of the involved accounts in a given transaction without revealing the amount of the spent input.

In the worst case scenario where there is no new outputs with the same delegate to select the decoys from, the account can change its delegate.

3.3.1.1 Security Proofs of SimpleDSA

According to the security model, SimpleDSA is secure if it is computationally infeasible for an adversary to distinguish the real input from the decoys in a ring signature.

Theorem 1. *SimpleDSA is secure against an adversary attempting to distinguish the real input from decoys in a ring signature.*

Proof. Assume, for contradiction, that there exists an adversary who can distinguish the real input from the decoys in a ring signature produced by SimpleDSA with non-negligible probability.

In SimpleDSA, each ring signature includes the real input and several decoys selected from different past transactions. The decoy selection process has two crucial properties:

- Decoys from Different Transactions: Each decoy comes from a different past transaction. This design ensures that there are no distinguishable patterns or homogeneity that link the real input to a specific transaction, mitigating the risk of a homogeneity attack. Since all decoys are equally likely to be the real input, the adversary cannot leverage transaction-specific information to identify the real input.
- Shared Decoy Set Across Outputs: All outputs in a transaction share the same predetermined set of decoys. This means that the ring signatures for all outputs contain the same elements. Even if an adversary knows that certain decoys have been previously spent (which could be exploited in a chain-reaction analysis), this information does not help in distinguishing the real input. Since all ring signatures include these decoys, and they are used consistently across all outputs, the adversary cannot attribute the spending to any particular input.

Furthermore, the consistent use of decoys across all outputs obfuscates any information that could potentially link an input to an output. The adversary cannot gain any advantage

by analyzing which decoys are spent or unspent because this status does not vary between ring signatures within the same transaction.

Therefore, any strategy that the adversary might employ—whether exploiting patterns in decoy selection or leveraging knowledge about spent decoys—is rendered ineffective by the design of SimpleDSA. The adversary cannot distinguish the real input from the decoys except with negligible probability, which contradicts our initial assumption.

□

3.4 Amount Anonymity

In this section we investigate how we can achieve amount anonymity in a DPoS based payments network.

3.4.1 Multi-Party Computation

We present a tool for implementing amount anonymity in permissionless decentralized networks that use Delegated Proof of Stake as a defence mechanism against Sybil attacks that is both compatible with the account model and the UTXO model.

Depending on the model, every account or transaction output on the network has a delegate, and the delegate can use that stake to participate in the consensus on behalf of the delegator. The fundamental idea is for the consensus algorithm to function with the network knowing only the total delegated stakes to each delegator but without knowing the individual stakes of each account or transaction output. It is composed of three different schemes.

- The first is a weighted threshold secret sharing scheme based on Shamir's secret sharing scheme, used to generate a secret amongst a set of distributed parties, which will be a private key of an additive homomorphic ElGamal cryptosystem over elliptic curves.
- The second is a polynomials commitment scheme used to make the previous scheme verifiable, i.e., without the need for a trusted dealer.
- The third scheme is used to decrypt an ElGamal ciphertext without reconstructing the private key, which, because of this, can be used multiple times.

As mentioned in the previous section, Delegated Proof of Stake is a *Sybil* defense mechanism that relies on knowing the *total* balances that are delegated to delegates (that can include their own individual balances or not) to function. However, it doesn't need to know the *individual* balances of the accounts that delegate to a certain delegate.

Moreover, total delegated balances of delegates, i.e., voting weights, could be trended over time (using an average, for example) to prevent attacks that temporarily shift the delegated stakes in a drastic way, like a DDoS attack.

Our idea for implementing amount anonymity at the transaction level is based on these premises. First, we need a cryptosystem that is additive homomorphic, so that a delegate can decrypt the total of a sum without decrypting its individual components, which would be the encrypted balances of the accounts. We want this cryptosystem to be as efficient as possible, in computation time (encryption and decryption) and in bandwidth (generating a small size ciphertext). We also need a scheme for generating a private key for the previous cryptosystem in a distributed way (without a trusted dealer), and we want each part to have a weight on the scheme proportional to their voting weight on the network. This private key will be used to generate a public key that will be used by all accounts to encrypt the balances and transaction amounts.

From time to time, the voting weights will have to be updated and, for that to happen, the new total delegated balances will have to be known. Therefore, we also need a way to decrypt the total delegated balances without reconstructing the private key, otherwise all individual balances could be also decrypted. Moreover, we require that delegates with a certain threshold of total weight are able to perform this decryption.

Based on the requirements stated in the previous section, we now present a scheme for implementing amount anonymity in the transaction layer, which has three different components: a weighted threshold secret sharing scheme, which will generate an asymmetric cryptosystem in a distributed fashion; a way to make this scheme verifiable, and, finally, a way to decrypt the ciphertexts of the cryptosystem without reconstructing the private key.

3.4.1.1 Notation

We use the following notation:

- Z is the ring of integers and $[n]$ is the set $1, 2, ..., n$ of n elements.
- F_p and F_q are finite fields of p and q elements respectively, with $p, q \in Z$.
- i and j represent the individual elements of $[n]$, with $i \neq j$, unless stated otherwise.
- P_i and P_j (with weights w_i and w_j respectively) are the participants of a scheme.
- $R[x]$ is the univariate polynomial ring in the variable x over R, R being a ring.
- deg($f(x)$) is the degree of $f(x) \in R[x]$.
- <group> is a bilinear group parsed as a tuple $(G_1, G_2, G_T, p, G, H, e)$.

- G_1 and G_2 are two additive groups of prime order p and G_T is a multiplicative group of the same order p.
- G is an element of G_1 that generates a QR_p subgroup of order q and H is an element generator of G_2.
- $e : G_1 \text{ x } G_2 \rightarrow G_T$ is a pairing that satisfies the properties of bilinearity, non-degeneracy and computability [Men05].

3.4.1.2 A weighted threshold secret sharing scheme

In our weighted threshold secret sharing scheme (n, t) we want to generate a secret s and divide it into n parts, s_i, in a distributed way with no dealer.

With t or more parts, the secret s can be easily reconstructed, and with less than t parts you get no knowledge about it. For this, we are going to use Shamir's secret sharing scheme [Sha79], in the following way:

1. P_i chooses a random polynomial $f_i(x) \in F_q[x]$ of degree $t-1$, with $f_i(0) = s_i$ being the subsecret of each P_i.
2. P_i computes $f_i(x)$ for $x = 1, ..., w_i$, where w_i is the number of shares corresponding to its weight.
3. Each P_i sends those points to P_j. These points are called the subsecret private share/s $s_{ik} = f_i(k)$, where $k = 1 + \sum_{l=1}^{j-1} w_l, ..., \sum_{l=1}^{j-1} w_l$.[1]
4. Each P_i will calculate w_i secret private shares $s_k = \sum_{i=1}^{n} s_{ik} = \sum_{i=1}^{n} f_i(k)$.
5. The secret $x = f(0)$, with $f(x) = \sum_{i=1}^{n} f_i(x)$, can be reconstructed with at least t secret private shares using Lagrange's interpolation as:

$$x = \sum_{k=1}^{t} s_k \prod_{k=1, i \neq k}^{t} \frac{i}{i-k} \quad (3.1)$$

3.4.1.3 A verifiable weighted threshold secret sharing scheme

In this section we show how we can make the previous scheme verifiable. This can be achieved if P_j can verify that the subsecret private shares s_{ik} are indeed points of the polynomials $f_i(x)$.

Feldman's scheme [Fel87] allows for this, however it requires commitments for every coefficient of $f_i(x)$, which doesn't scale very well for large polynomials.

[1] As an example, consider three participants P_1, P_2, P_3 with weights 4, 3 and 2 respectively. Then P_1 will generate a polynomial $f_1(x)$ with private subsecret x_1 and calculate 9 shares $s_{1k} = f_i(k)$, for $k = 1, ..., 9$. P_1 then sends the points s_{15}, s_{16}, s_{17} to P_2 and the points s_{18}, s_{19} to P_3.

Because of this, we use instead the scheme proposed in [CHM+20], an improvement to [KZG10], that allows multiple univariate polynomial commitments for a single degree bound and a single evaluation over a field family $\mathbb{F}$.

For efficiency purposes, we will use the construction in the *algebraic group model*.

The scheme is composed of the following tuple of algorithms PC = (Setup, Commit, Open, Check), defined as following:

Setup. On input a security parameter λ and a maximum degree bound $D \in N$, PC.Setup samples public parameters (ck, rk) as follows. Choose an appropriate bilinear group <group> $\leftarrow$ Group(1^λ), and parse <group> as a tuple $(G_1, G_2, G_T, p, G, H, e)$. Sample random elements $\beta \in F_q$.

Then compute the vector

$$\sum := \Big(G, \beta G, \beta^2 G, ..., \beta^D G\Big) \in G_1^{D+1}$$

The parameter β can be calculated in a distributed way, where P_i chooses a β_i and broadcasts $\beta_i G$,$\beta_i^2 G$,...,$\beta_i^D G$.

Values can be verified by checking that :

$e(\beta_i G, H) = e(G, \beta_i H), e(\beta_i^2, H) = e(\beta_i G, \beta_i H), ..., e(\beta_i^D, H) = e(\beta_i^{D-1}, \beta_i H)$.

The parameters will be $\beta G = \sum_{i=1}^n \beta_i G$, $\beta^2 G = \sum_{i=1}^n \beta_i^2 G$,...,$\beta^D G = \sum_{i=1}^n \beta_i^D G$.

Set ck := (<group>, $\sum$) and rk := (D, <group>, βH), and then output the public parameters (ck, rk).

These public parameters will support polynomials over the field $F_q[x]$ of degree at most D.

Commit. On input ck, univariate polynomial p over $F_q[x]$, PC.Commit outputs the commitment c that is computed as follows:

- If $deg(p(x)) > D$, abort.
- Otherwise, output $c := p(\beta)G$.

Note that because p_i has degree at most D, the above terms are linear combinations of terms in ck.

Open. On input ck, univariate polynomial $p(x)$ over F_q, evaluation point $z \in F_q$, opening challenge $\varepsilon \in F_q$, PC.Open outputs the evaluation proof π in $\mathbb{G}_1$, that is computed as follows. Compute the linear combination of polynomials $p(x)' = \varepsilon p$ and witness polynomial $w(x) := \frac{p(x)' - p(z)}{x - z}$. Set $w := w(\beta)G \in \mathbb{G}_1$. The evaluation proof is $\pi := (w)$.

Check. On input rk, commitment c, evaluation point $z \in F_q$, alleged evaluation v, evalu-

ation proof π, and randomness $\varepsilon \in F_q$, PC.Check proceeds as follows. Compute $C := \varepsilon c$, $y := \varepsilon v$ and check the evaluation proof via equality $e = (C - yG, H) = e(w, \beta H - zH)$.

This protocol can be made zero-knowledge by having the prover send vG instead of v.

With this protocol, the subsecret private shares can be verified in the following way:

1. P_i uses PC.Commit algorithm to commit to $f_i(x)$, with $D = t - 1$, and sends it to all participants.
2. P_i makes a proof evaluation of $s_{ik}G$ using PC.Open algorithm and sends it to P_j, so that the other participants can verify s_{ik} without getting to know it.

3.4.1.4 Decrypting without reconstructing the secret

As already stated, the secret distributed in the previous scheme will be the private key of an asymmetric cryptosystem, and will generate the public key that will be used to encrypt the balances of the network. This section proposes a scheme to decrypt these ciphertexts **without** reconstructing the private key.

According to [CGS97], the most efficient cryptosystem for this purpose is the additive ElGamal over elliptic curves (EC-EG), whose key set-up consists of an elliptic curve $\mathbb{G}_1$ over F_p with a generator G of order q. Its security is based upon the Elliptic Curve Discrete Log Problem (ECDLP).

$x \in Fq$ is the private key of the system and $Y = \sum_i^n f_i(0)G = \sum_i^n x_i G = xG \in \mathbb{G}_1$ is the public key. The public key can be verified using the previous scheme, with a proof evaluation that $f_i(0)G$ comes indeed from the previously committed $f_i(x)$ polynomial.

There is an efficient and invertible function $map()$ that maps values (e.g. plaintexts) into points on the curve, and vice versa.

The encryption of the balance $m \in F_q$ is done the following way:

- r is a random value $\in Fq$.
- ciphertext $C = (C_1, C_2)$, where $C_1 = rG$ and $C_2 = M + rY$.

The decryption process is done in the following way:

- $M = -xC_1 + C_2 = -xrG + M + xrG$.

In this scheme we don't want the private key to be reconstructed during the decryption process, in order to be able to use it to decrypt multiple ciphertexts. For that, we can use the distributed decryption based on [RNH12], in the following way:

1. P_i computes his decryption share/s $D_k = b_k s_k C_1$, where $b_k = \prod_{k=1, i \neq k}^{t} \frac{i}{i-k}$ is public and can be calculated by any participant.
2. $s_k C_1$ can be proved correct with an equality proof of logarithms [CS97] of $s_k G$ and $s_k C_1$, with $s_k G = \sum_{i=1}^{n} s_{ik} G$.
3. The message can be decrypted as $M = C_2 - \sum_{k=1}^{t} D_k$.

To find the balance m, it is first necessary to reverse the mapping function in order to map an elliptic curve point M back to a numeric value m. Balance fields with more than 64 bits are impractical, so either the encrypted balance will have to be composed of various ElGamal ciphertexts (4x32 bits, for example) or another additive homomorphic cryptosystem will have to be used [CGS97].

Another issue is that the decryption protocol can create a new attack vector on the network, namely with thousands of accounts asking for decryption and thereby clogging the network. Therefore, some kind of mechanism will be needed to prevent this, like a proof of a minimum balance.

3.4.2 Diffie-Hellman Key Exchange

We propose a technique to implement amount anonymity in the transaction layer that is compatible with Delegated Proof of Stake, where each node on the network is able to delegate its stake to another node, and with both the account model and the utxo model.

Every transaction amount is represented by a Pedersen commitment and the idea is to do two Diffie-Hellman key exchanges, one between the sender and the receiver, and another between the sender and the delegate of the receiving account or the resulting utxo.

The sender produces two ciphertexts by encrypting a plaintext with two symmetric keys, each one corresponding to the resulting shared secret of each of the Diffie-Hellman exchange. This way, both the receiver and the delegate are able to recover the same plaintext, which will then be used to open the Pedersen commitment to reveal the amount.

Since a delegate knows the amounts of the accounts or utxos that delegated to him, he can prove its total delegated stake to the network by opening the sum of all the Pedersen commitments, which are additive hommomorphic, without revealing the individual amounts of each Pedersen commitment.

3.4.2.1 Building Blocks

Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange, one of the earliest public-key protocols, was named after Whitfield Diffie and Martin Hellman [DH76]. This groundbreaking method facili-

tates the secure exchange of cryptographic keys over a public channel, remarkably without necessitating any prior secret information exchange between the involved parties.

The operational mechanics of the Diffie-Hellman Key Exchange are delineated as follows:

Initialization Phase: - Commencement of this process involves selecting and publicly disclosing one large prime number, denoted as p, and the group generator, denoted as g. Here, p represents a prime number, while g functions as a primitive root modulo p, also known as a generator for the multiplicative group of integers modulo p.

Key Generation Phase: In this phase, each party independently and confidentially generates a private key. Specifically, Party A selects a private key a, and Party B chooses b, both randomly drawn from the range 1 to $p - 1$.

Public Key Computation: Each entity then computes their public key. This is achieved by exponentiating g with their respective private key and applying the modulus p to the resultant value. Party A calculates $A = g^a \mod p$ and conveys A to Party B. Conversely, Party B calculates $B = g^b \mod p$ and transmits B to Party A.

Shared Secret Derivation: Upon receiving the other party's public key, both parties engage in computing the shared secret. This involves exponentiating the received public key with their private key and taking the modulus p of this value. Party A arrives at $s = B^a \mod p$. Party B computes $s = A^b \mod p$.

Key Verification: Theoretically, if the process is executed without errors or external interferences, both Party A and Party B should derive an identical shared secret key s, since $g^{ab} \mod p$ is congruent to $g^{ba} \mod p$.

The underpinning security principle of the Diffie-Hellman Key Exchange is the computational complexity associated with the discrete logarithm problem. This involves the challenge of deducing a from known values of g, p, and $g^a \mod p$. Calculating $g^a \mod p$ is computationally straightforward, but reversing this process, especially for large values of p, is not.

It is crucial to acknowledge that the Diffie-Hellman Key Exchange in isolation does not guarantee the authentication of the parties involved and is vulnerable to man-in-the-middle attacks. Therefore, in practical applications, it is often employed in conjunction with additional cryptographic protocols that provide the necessary authentication.

Pedersen Commitment

The Pedersen Commitment [Ped92] is a cryptographic protocol that allows an individual to commit to a specific value while keeping it concealed until a later point when it can be disclosed. A key characteristic of Pedersen commitments is their additive homomorphic property, enabling the combination of multiple commitments to create a new commitment representative of the sum of their individual values.

Symmetric Encryption

Symmetric Encryption is a cryptographic technique where the same key is used for both the encryption and decryption of messages. The efficacy and security of this method are contingent upon the secrecy and integrity of the symmetric key.

3.4.2.2 The Scheme

1. **Transaction Setup:** Let S be the sender, R be the receiver, and D be the delegate of R's account or resulting UTXO. S and R engage in a DHKE to establish a shared secret k_{SR}. Simultaneously, S and D engage in a separate DHKE to establish a shared secret k_{SD}.

2. **Pedersen Commitment:** Sender S decides on the transaction amount a and selects a random blinding factor b. S computes the Pedersen commitment C to the amount a with blinding factor b, where $C = g^a h^b$ for publicly known group generators g and h.

3. **Encryption:** S encrypts the transaction amount a and blinding factor b using the shared secret k_{SR} as the symmetric key to create ciphertext $E_{SR}(a, b)$. S separately encrypts a and b using the shared secret k_{SD} as the symmetric key to create ciphertext $E_{SD}(a, b)$.

4. **Transaction Broadcasting:** S broadcasts the transaction, including the Pedersen commitment C and the two ciphertexts $E_{SR}(a, b)$ and $E_{SD}(a, b)$, to the network.

5. **Decryption and Verification:** Upon receiving the transaction, R uses k_{SR} to decrypt $E_{SR}(a, b)$ to recover a and b, and similarly, D uses k_{SD} to decrypt $E_{SD}(a, b)$.

6. **Proof of Stake Calculation:** D, knowing the sums of the amounts delegated to him, can prove the total delegated stake to the network by the additive homomorphism property of Pedersen commitments, enabling D to open the sum of all Pedersen commitments without revealing individual amounts.

This scheme is much more efficient than the previous one based on secret sharing, since each node can prove its own delegated proof of stake instead of using threshold decryption, and does not have the amount size limitation. However, the level of anonymity is not as strong, since delegates know the individual amounts of each account or utxo. This can be minimized by using stealth addresses, obfuscating the link between an account or utxo to a master address, or by a node being its own delegate.

3.5 Conclusion

In conclusion, Chapter 3 of this thesis offers a significant contribution to the discourse on privacy in digital transactions. In particular, the chapter introduces novel algorithms

and techniques aimed at enhancing sender and amount anonymity within cryptocurrency networks.

The development of the Simple Decoy Selection Algorithm (SimpleDSA) is presented as a means to improve sender anonymity by providing a predefined set of decoys that do not rely on past transaction histories, mitigating homogeneity and chain-reaction attacks. For amount anonymity, a multi-party computation tool compatible with various blockchain models is proposed. This tool employs a weighted threshold secret sharing scheme, a polynomial commitment scheme for verifiability, and a method for decrypting ciphertexts without reconstructing the private key, focusing on the use of an additive homomorphic ElGamal cryptosystem.

Furthermore, the chapter discusses the application of Diffie-Hellman key exchanges in conjunction with Pedersen commitments to preserve amount anonymity during transactions. This method allows for the validation of a delegate's total stake without revealing individual account balances, providing an efficient alternative to the previously described secret sharing-based approach.

Despite the advancements presented, the chapter acknowledges existing challenges and outlines areas for future research, such as the practicality of encrypted balance fields, the potential network congestion caused by decryption requests and the process of updating the delegated stakes of the network.

In summary, Chapter 3 not only demonstrates an ongoing commitment to privacy and security in the realm of cryptocurrencies but also sets the stage for further innovations that will bolster the trust and utility of decentralized financial systems.

Chapter 4

Consensus Layer

This chapter presents the research undertaken with the goal of designing a consensus algorithm for cryptocurrencies with lower latency than the current state-of-the-art while maintaining a level of throughput and scalability sufficient for real-world payments. The result is Nero, a new deterministic leaderless byzantine consensus algorithm in the partially synchronous model that is especially suited for Directed Acyclic Graph (DAG)-based cryptocurrencies. In fact, Nero has a communication complexity of O(n^3) and terminates in two message delays in the good case (when there is synchrony). The algorithm is shown to be correct, and we also show that it can provide eventual order. Finally, some performance results are given based on a proof of concept implementation in the Rust language.

4.1 Introduction

The consensus problem was first introduced in [PSL80]. It can be described as a set of entities that wish to agree on something, which could be a value, an action or a statement. At first glance, it appears a simple problem to solve, especially if all parties are honest. However, it becomes complex when some subset of those entities can behave unexpectedly, such as by not participating and leaving the consensus process without revealing their choice to the other parties or by acting maliciously by actively trying to sabotage the consensus scheme so that no consensus is reached or such that honest parties decide differently instead of arriving at a consensus.

This problem is typically studied in the context of computer science, where the entities are processes on computers and where the computers communicate with each other through an unreliable network such as the internet. However, this problem and the underlying principles can be applied to other areas, such as politics and game theory. In fact, one of the seminal works in this area is a metaphor for a group of generals on the battlefield who need to decide if they attack or not [LSP82], and this is where the term “byzantine” comes from, meaning any type of faulty behavior, also called “arbitrary” faults. Ever since, it has become one of the most studied problems in computer science due to its both theoretical and practical importance. From a theoretical standpoint, it has been shown to be equivalent to other problems, such as atomic broadcast and state machine replication, meaning that if you can solve one, you can also solve the others. From a practical standpoint, it is very useful in the deployment of real-world distributed systems.

Algorithms for reaching agreement started with leaderless synchronous consensus algo-

rithms [LSP82, DS83], but these had limited application in practice since distributed systems in the real world are not synchronous. This has led to the study of asynchronous distributed systems and to one of the most important theorems in the field: the Fischer Lynch Paterson FLP impossibility theorem [FLP85]. According to this theorem, it is not possible to have agreement and termination in the asynchronous model if one of the processes crash. Throughout the years, there have been many ways to circumvent this, such as randomization [MXC+16], modifying the model [DLS88] or the properties [CNV06] of the problem and byzantine fault detectors [KMMS03, HKD06].

Despite being an important field of research in computer science, distributed systems and particularly consensus algorithms gained new momentum with the emergence of Bitcoin [Nak09], the first decentralized-permissionless network that uses a blockchain as a ledger to store the transactions. Bitcoins consensus mechanism, the Nakamoto consensus [Ren19], led to the development of a whole new family of consensus algorithms based on Proof of Work (PoW).

However, soon the limitations of a design based on PoW became evident, such as low throughput and high latency, and new alternatives started to appear, such as Proof of Stake (PoS) [Sal20, KRDO17, LABK17] and PoS with a ledger structure based on directed graphs instead of a linear, chronological order.

Despite improving on the limitations of Bitcoin, new alternatives based on proof of stake also solved state machine replication [Sch90a] by sequentially executing consensus instances for agreeing on each block of transactions to append to the blockchain.

State Machine Replication (SMR) has long been a fundamental component in distributed systems [Sch90b]. Traditionally, it was used as a way to guarantee redundancy and, consequently, reliability in a centralized system. However, due to the emergence of cryptocurrencies and blockchain technology, recent research has focused mainly on SMR applied to decentralized systems, where consensus plays a critical role.

One of the most remarkable works in this area is the pivotal FLP (Fischer, Lynch, and Paterson) impossibility theorem [FLP85], asserting that no consensus algorithm can simultaneously be asynchronous, deterministic and terminate. Attempts to skirt around this theorem have culminated in the emergence of a variety of consensus algorithm typologies, including deterministic partially synchronous, randomized asynchronous, leader-based, and leaderless algorithms.

Notwithstanding these theoretical considerations, the most consequential factors in real-world settings pivot on practical utility and performance metrics. Primarily, these encompass scalability, latency and resilience, i.e., how they are impacted by faults and Byzantine behavior. Invariably, these elements exist in a state of delicate trade-off, as enhancements to one may inadvertently undermine another, requiring strategic design choices.

4.1.1 Technical Overview

In this chapter, we present two new leaderless deterministic consensus algorithms, called Nero and Echidna, making it suitable for DAG-based cryptocurrencies. Theoretical properties of the algorithms are derived, including a complexity analysis and correctness properties. The final contribution is an open-source implementation in the Rust language.

From the consensus algorithm, we have developed two methods to achieve efficient State Machine Replication (SMR): one leader based and a second leaderless one. Our design prioritizes low latency and robust resilience, followed by scalability, making it particularly suited for decentralized payment systems. We assert that to genuinely rival established centralized payment systems, a decentralized payment system must match, if not surpass, their performance. Therefore, our focus is on fostering an algorithm that delivers this capability.

The contributions of this chapter are summarized below:

- A new consensus algorithm called Nero based on voting that uses a termination mechanism by waiting for slow or lost votes to eventually arrive and detect potential double votes.

- A new consensus algorithm called Echidna, which is similar to Nero but uses a more efficient termination mechanism by adding a proof round to each vote. This allows vote conflicts to be resolved by choosing the vote that was produced in the earlier round.

- A new algorithm called Sphinx that solves the State Machine Replication problem with a leader-based approach by building upon Echidna. Due to its mechanism to handle conflicting values, Sphinx can have lower timeouts than other leader-based consensus algorithms, since conflicting proposals of different leaders are resolved by Echidna. Consequently, the latency of the consensus is lower.

- A new algorithm called Cerebrus that solves the State Machine Replication problem with a leaderless approach that uses Echidna as a building block. Cerebrus avoids the leader bottleneck, since any process can propose a set of values, but does not have the shortcomings of leaderless algorithms, such as low scalabilty, since the decided value is a set of at least $2f + 1$ proposals instead of a single proposal.

- We implement Sphinx and Cerebrus in Rust in an open source code, and benchmark it against the state of the art consensus algorithm Bullshark in the common case, under crash-faults and with conflicting values. We demonstrate that they have comparable latency and resilience and that they also have good scalability.

4.1.2 Related Work

This section provides an overview of some remarkable research related to SMR and consensus protocols.

Classical consensus protocols, such as Paxos [Lam01] and Raft [OO14], have been subjected to rigorous study given their critical role in distributed computing, but both only provide a solution to non-Byzantine faults within a network, i.e., crash faults.

The PBFT (Practical Byzantine Fault Tolerance) protocol [CL+99], with its pioneering approach, has fueled extensive investigations in the domain of BFT, though it falls short in terms of scalability.

Subsequent protocols improve on PBFT, HotStuff [YMR+19] and SBFT [GGAG+19], by offering improvements over PBFT's message complexity by requiring only a linear number of messages in the common case. HotStuff, despite having a better communication complexity, has high latency and its throughput drops to zero when the leader fails and until some view-change completes [VG19]. Tendermint [BKM18] improved the view-change complexity with a new termination mechanism. IBFT is another algorithm based on this design [Mon20].

Other categories of BFT protocols were also developed, such as optimistic BFT protocols, like Zyzzyva [KAD+07], with better performance in the good case; and probabilistic BFT protocols like Honeybadger [MXC+16] and BEAT [DRZ18], that achieve both safety and liveness in purely asynchronous networks.

In cryptocurrencies, the Nakamoto consensus protocol, [Ren19], adopted widely in blockchain systems, offers significant robustness against Sybil attacks, but suffers from high latency, low throughput, and substantial computational resource usage, stirring environmental concerns. Stellar [Maz15] and Algorand [GHM+17] try to solve those issues by employing a Byzantine agreement protocol with asymmetric quorums and using a probabilistic sampling mechanism for committee selection, respectively.

Other approaches use a multi proposers design instead of a single one, MirBFT [SDPV19], and Red Belly [CNG21], but the former still has a single point of failure in the leader, called "primary", and the latter is based on a binary consensus algorithm, which needs to be instantiated multiple times, while our design is built on a single instance of multi-valued consensus.

DBFT [SGSKK22a] does not require signatures and has the same communication complexity; however, it is not completely leaderless due to its requirement of a weak coordinator.

In order to improve scalability, another ledger structure called directed acyclic graph has gained popularity. This allows for consensus instances to be executed concurrently instead of sequentially and thereby improves the performance of the system[CZF+20]. Ex-

amples are [BL20, DKKSS22a, KKKNS21, CHK+18].

A DAG is a graph that is made up of a set of vertices (or nodes) connected by directed edges (or arcs) such that a closed chain is not possible. A blockchain is a type of acyclic graph where each vertex is a block of transactions and is connected to only another vertex sequentially. In a general DAG, each vertex can be connected to many vertices, and these connections are not made sequentially but concurrently instead. A blockchain can be seen as a DAG where only one edge can be connected to a vertex.

In the context of cryptocurrencies, this allows for a greater throughput and less latency when validating and adding transactions to the ledger compared to a blockchain. However, the downside is that it is much more difficult to order those transactions and, consequently, for the nodes of the network to have a common sense of time. A common sense of time is useful, for example, for synchronous network upgrades, for pruning the ledger and bootstrapping. Another problem is scalability and decentralization, which are connected to each other.

Many consensus algorithms are also used to assign a node the status of the leader. However, a leader-based approach has limitations, especially when nodes are geographically dispersed over a wide area network. Firstly, the latency for clients far from the leader is obviously increased. Furthermore, the leader can become a bottleneck or its network performance may degrade, leading to an overall decrease in system-wide performance decreases. Finally, if the leader fails, the whole system cannot serve new requests until an election of a new leader takes place, thereby affecting availability.

Therefore, a ledger based on a DAG design is most suited for leaderless consensus algorithms, which can be probabilistic [MPP+22, ava18] or deterministic [ADG+21]. In fact, Leaderless State Machine Replication (SMR) offers appealing properties with respect to leader-driven approaches. Protocols are faster in the best case, suffer from no downtime when the leader fails, and distribute the load among participants.

Narwhal [DKKSS22b], Bullshark [SGSKK22b] have very good performance, but they are specifically designed for a Directed Acyclic Graph (DAG) and they have a more complex implementation than traditional blockchains.

Archipelago [ADG+21] is the only other leaderless deterministic consensus algorithm and has the same communication complexity; however, it has a greater latency.

In table 4.1, we compare some relevant deterministic consensus algorithms with Nero and Echidna using different parameters, such as the message complexity during normal case and view change, the latency (message delays) and if they are leaderless or not.

Table 4.1: Comparison of different consensus algorithms.

	Normal Case	View Change	Message Delays	Leaderless
PBFT [CL+99]	O(n^3)	O(n^4)	3	No
HotStuff [YMR+19]	O(n^2)	O(n^2)	8	No
IBFT [Mon20]	O(n^2)	O(n^2)	3	No
Tendermint [BKM18]	O(n^3)	-	3	No
DBFT [SGSKK22a]	O(n^3)	-	4	No
Archipelago [ADG+21]	O(n^3)	-	5	Yes
Echidna	O(n^3)	-	2	Yes
Nero	O(n^3)	-	2	Yes

4.2 Preliminares

4.2.1 System Model

The system consists of a set P of n asynchronous processes, namely $P = p_1, \ldots, p_n$, where asynchronous means that each process proceeds at its own speed, which can vary with time and remains unknown to the other processes. Up to f processes among $n = 3f + 1$ are byzantine, meaning they can fail or behave arbitrarily.

We use a variant of the partially synchronous system model [DLS88] where the system can be in one of two states in a given time: GST (Global Stabilization Time) and non-GST. In the first, there is a known bound Δ to all sent messages, meaning that a message broadcasted by a correct process at time t will be delivered by all correct processes before $t + \Delta$. In a non-GST state, this is not guaranteed to happen; however, it is assumed that the system reaches GST at least once.

In this section, we first give an overview and intuition behind the design of the Nero algorithm and describe the data structures necessary for the execution of the algorithm. Then we present the algorithm, prove its correctness and finally describe the mechanism to satisfy the eventual order.

4.2.2 Data Structures

We abstract the data structures and refer to only the contents relevant to the operation of the consensus algorithm.

Block. Data containing the value transferred from the sender account to the receiver account.

Transaction. To simplify the presentation, we abstract the contents of a transaction that are not relevant to the consensus algorithm, and we assume that a transaction is composed

of the origin block hash and the new block hash.

Round. A simple integer that starts at 0 and ends in the round where the value of an election is decided.

Elections. Each correct process maintains a hashmap of the active elections, where the key is a block hash (corresponding to the origin block of a transaction) and the value is the state of the election. An election can have multiple competing transactions (called forks) if they have the same origin block and the purpose of the consensus is that all correct processes validate one of the transactions or none.

Election State. The state of an election is represented by a hashmap, where the key is the election hash and the value a set of round states.

Round State. The state of a round is represented by a hashmap, where the key is the round number and the value the tally of the messages of that round. It also contains the validated and pending messages.

Tally. Each correct process computes a tally of the messages received in each round, with the total number of votes or commits in each value (block hash or nil).

Messages. A message is composed of a:

- Type: VOTE or COMMIT,
- Value: a block hash or a nil value,
- Round.

A message m can be valid, pending or invalid. A message m of round r is considered valid by a correct process p if there is a set of at least $2f + 1$ messages received by p in the previous round $r - 1$ that are compatible with the value and type of message m. A message is considered pending if there is no set compatible, but it is still possible to have that set by receiving the remaining messages of round $r - 1$. It is considered invalid if it is not possible to have a compatible set with the message.

Timer. At the start of each round, a correct process starts a timer t_r, where r is the round.

New round. A correct process starts a new round $r+1$ when it has received at least $2f+1$ messages in round r and the timer t_r has expired. During GST, all messages from correct processes will be received before the timer expires; however, a correct process has no way of knowing that it is in GST, i.e., if it receives $2f + 1$ messages and the timer expires, it is not guaranteed that those messages are all from correct processes.

4.3 Multi-Valued Consensus

We use a variant of the multi-valued byzantine consensus problem called Validity Predicate-based Byzantine consensus [CGLR17], with the following properties.

- Agreement: No two correct processes decide on different values.
- Termination: All correct processes eventually decide on a value.
- Validity: A decided value is valid, i.e., it satisfies the predefined predicate denoted valid().

Correctness. An algorithm satisfies the Multi-valued Consensus if it satisfies validity, agreement and termination.

4.3.1 The Nero Algorithm

Validation rules. After receiving a message m of round r, a correct process validates it and decides one of the following statuses:

- Valid if there is a set of at least $2f + 1$ messages of round $r - 1$ that are compatible with the value and type of m.
- Pending if there is not a set of at least $2f + 1$ messages of round $r - 1$ that are compatible with the value and type of m but it is still possible to receive new messages of round $r - 1$ to produce that set.
- Invalid if there is not a set of at least $2f+1$ messages of round $r-1$ that are compatible with the value and type of m, and it is not possible to receive new messages of round $r - 1$ to produce that set.

Decision rules. With the start of a new round $r + 1$, a new decision has to be made to decide the message that the process is going to send in the new round. This message is based on the messages received in the previous round r (at least $2f + 1$ valid messages in total), in the following way (only one rule can be applied):

- If process p has received at least $2f + 1$ valid messages of type COMMIT with the same value, decide that value. There is no need to send a new message because all the other correct processes will eventually receive the same messages and decide the same value as well.
- Else if process p has received at least one valid message of type COMMIT with some value v, send a message of type COMMIT with value v. (There cannot be two commits of different values in the same round).

- Else if process p has received at least $2f + 1$ valid messages of type VOTE with the same value, send a message of type COMMIT with that value.
- Else if process p has received at least one valid message of type COMMIT with some value v in round r and has already committed to another value v' in round r', send a message of type COMMIT with value v only if round $r > r'$. If $r < r'$, send a message of type $COMMIT$ with the previous committed value v'.
- Else if process p has received at least $f + 1$ valid messages of type VOTE with value nil, send a message of type VOTE with value nil.
- If no block hash value has at least $f + 1$ votes, send a message of type VOTE with value nil.
- Else if process p has not received at least $f + 1$ valid messages of type VOTE with value nil, send a message of type VOTE with the value with most votes.

Execution. In each round r, a correct process waits for at least $2f + 1$ valid messages and for the timer t_r to expire. Every time it receives a valid message, it does the following:

- Inserts it in the respective election state.
- Updates the tally of the election.
- Tries to validate previous pending messages based on the validation rules.
- Broadcasts m to the network.
- Computes a new message m' of round $r + 1$ based on the received messages and the decision rules.
- Broadcasts the message m' to the other processes.

4.3.1.1 Correctness

In this section, we discuss the correctness of the Nero consensus algorithm.

Lemma 1. *There cannot be two or more valid commits with different values in a given round.*

Proof. Since, per the consensus algorithm, a correct process never votes or commits more than one block hash in the same round in the same election, it is not possible to have two sets of $2f+1$ of different values without at least one correct process voting for two different values, so we prove by contradiction. □

Lemma 2. *If a correct process commits a value v in round r and it receives a valid commit of another value v' in round r', $r' > r$, no correct process could have decided v in round r.*

Proof. If a correct process decides value v in round r, it means that it has received at least $2f + 1$ commits with value v in round $r - 1$. Since at least $f + 1$ of those commits are from correct processes, they will not change its value and so the $2f + 1$ valid messages of value v' of type VOTE needed to commit value v' will not be produced. Inversely, if there was a commit of value v', it means that value v was not decided in a previous round. □

Lemma 3. *The Nero algorithm satisfies Termination.*

Proof. We need to prove that all correct processes decide. For that, we have to prove that all correct processes eventually commit a value and then that all correct processes eventually decide a value. The proof follows from lemmas 4 and 5. □

Lemma 4. *All correct processes eventually commit a value.*

Proof. First, we prove that at least one correct process eventually commits. Since a byzantine process can send different valid votes (with different values) to different correct processes in a given round, it is possible that no correct process commits during the non-GST phase.

However, during the GST, all correct processes will eventually vote the same value in a given round r because they will all receive the same votes in round $r - 1$. □

Lemma 5. *All correct processes eventually decide a value.*

Proof. First, we prove that at least one correct process eventually decides, which happens because all correct processes will eventually commit the same value.

During the GST, all correct processes will eventually decide as well since they will receive the valid commits in which the correct process based its decided value. □

Lemma 6. *The Nero algorithm satisfies Agreement.*

Proof. We need to prove that all correct processes decide the same value. Since we already proved in lemma 5 that all correct processes eventually decide, we need to prove that they all decide the same value.

Suppose that a correct process p decides value v in round r. This means that it has received at least $2f+1$ commits with value v in round $r-1$. At most there are f byzantine processes, so at least $f + 1$ of those commits are from correct processes, which means that they will only change their commit if they receive a valid commit with value v' in a round r', $r' > r - 1$.

If $r' = r$, this cannot happen due to lemma 1. If $r' > r$, for a valid commit with value v' to happen, there needs to be at least $2f + 1$ valid votes in a previous round $r' - 1$, which is not possible since at least $f + 1$ correct processes will not change their commit with value v in round r and vote with value v' since $r' - 1 >= r$. Therefore, we prove by contradiction.

lemma 2 ensures that a correct process can change its commit to a subsequent valid commit while maintaining safety. □

Lemma 7. *The Nero algorithm satisfies Validity.*

Proof. A correct process only sends messages with a valid value so only valid values can be decided. □

Theorem 2. *The Nero algorithm achieves a multi-valued consensus, i.e., Termination, Validation and Agreement.*

Proof. Follows from lemmas 3, 6 and 7. □

4.3.2 The Echidna Algorithm

An instance of the consensus protocol, see algorithm 9, starts with each correct process broadcasting a vote on a specified value v, which is acquired externally from the consensus instance.

Subsequently, a timer for that round is initiated, and the process waits for at least 2f + 1 and the timer's expiration to generate a new message in the subsequent round.

If the process receives at least of 2f + 1 commits on value v, it decides that value (line 27).

In a situation where the process receives at least 2f + 1 votes on value v in round r, and no value has been committed yet, it commits that value with proof round r and broadcasts a COMMIT message containing that information (line 15).

Otherwise, if there is no committed value, it votes for the value that has received the most messages (both votes and commits). If there is a tie, it votes for the value with the highest hash (line 20).

Given the possibility that distinct processes may commit different values and consensus may not terminate, it becomes necessary to safely alter a committed value if required. There are two scenarios that may trigger this change:

- The process receives at least 2f + 1 messages with a different value v' in a round r' that is more recent than the proof round pr of its committed value v (line 15).

- The process receives at least f + 1 COMMIT messages with a different value v' in a round r' that is more recent than the proof round pr of its committed value v (line 32).

Line 36 and the timers between rounds guarantee that the algorithm terminates, since GST will eventually be reached and all correct processes will receive the same messages and, consequently, produce the same response messages. Since the actual delay of the network is unknown, the timer can be increased in each round.

4.3.2.1 Correctness Proofs

We now prove formally that Echidna solves the Multi-valued Consensus problem.

Lemma 8. *For all $f \geq 0$, any pair of process sets that have a voting power of 2f + 1 or more, there must be at least one correct process shared between them.*

Proof. Given that the sum of all voting power is represented by n, where n is equal to 3f + 1, we can see that double the voting power, or 2(2f + 1), is equivalent to n + f + 1. What this implies is that if we consider the intersection between two process sets that each have voting power equal to 2f + 1, they must have an overlap of at least f + 1 voting power. This overlap will contain at least one correct process, because the total voting power that can be contributed by faulty processes is only f. This direct reasoning leads us to our stated result. □

Lemma 9. *Echidna satisfies Agreement.*

Proof. Let r be the first round such that some correct process p decides v. We now prove that if some correct process q decides v' in some round $r' \geq r$, then $v = v'$.

In case $r = r'$, q has received at least $2f + 1$ commits of v v' in r r', while p has received at least $2f + 1$ commits of v v in r r.

By Lemma 1, two sets of messages of voting power $2f + 1$ intersect in at least one correct process. As a correct process sends only one message per r, then $v = v'$.

In case $r' > r$, by the decision rules, p has received at least $2f + 1$ voting power equivalent of commits of value v in round $r - 1$ with proof round pr.

On the other side, q has received at least $2f + 1$ voting power equivalent of commits of value v' in round $r' - 1$ with proof round pr'. By Lemma 1, that means that at least one correct process c changed its committed value from v to v'.

According to the algorithm, a correct process only changes its committed value iff it receives at least $f + 1$ commits of a different value with a more recent proof round than pr

Algorithm 9: Echidna

Initialization:
  $instance_p := 0$
  $round_p := 0$
  $committedValue_p := nil$
  $decision_p[] := nil$
  $voteTimer := nil$
  $value_p := nil$

Function $Start(instance_p, value)$:
  if $round_p = 0$
    $value_p \leftarrow value$
    broadcast <VOTE, $instance_p, round_p, v$>
    $voteTimer \leftarrow running$
    schedule $OnTimeoutVote(instance_p, round_p)$ to be executed **after** $timeoutVote(round_p)$
    $round_p \leftarrow round_p + 1$

upon $2f + 1$ <$*, instance_p, r, *, v$> **with** $r > round_p$ **AND** ($r > proofRound_p$ **OR** $committedValue_p = nil$) **do**
  $round_p \leftarrow r$
  $committedValue_p \leftarrow v$
  $proofRound_p \leftarrow r$
  broadcast <COMMIT, $instance_p, round_p, v$>

upon $2f + 1$ <$*, instance_p, round_p, *, *$> **AND** $voteTimer = expired$ **do**
  $round_p \leftarrow round_p + 1$
  if $committedValue_p \neq nil$ **then**
    broadcast <COMMIT, $instance_p, round_p, proofRound_p, committedValue_p$>
  else
    $value_p \leftarrow$ max($value$.count()) (if tie, the $value$ with the highest id)
    broadcast <VOTE, $instance_p, round_p, nil, value_p$>

upon $2f + 1$ <COMMIT, $instance_p, r, v$> **while** $decision_p[instance_p] = nil$ **do**
  if $valid(v)$ **then**
    $decision_p[instance_p] = v$
    $instance_p \leftarrow instance_p + 1$
    $StartRound(0)$

upon $f + 1$ <COMMIT, $instance_p, r, pr, v$> **with** $r > round_p$ **AND** ($pr > proofRound_p$ **OR** $pr = -1$) **do**
  $round_p \leftarrow r$
  $committedValue_p \leftarrow v$
  $proodRound_p \leftarrow r$

upon $f + 1$ <$*, instance_p, r, *, *v$> **AND** $r \geq proofRound_p > -1$ **AND** $committedValue_p \neq *v$ **do**
  rebroadcast proof of $committedValue_p$ from round $proofRound_p$

Function *OnTimeoutVote*($height, round$) :
  $voteTimer \leftarrow expired$

(line 32) or at least $2f + 1$ messages of a different value in a more recent round than pr (line 15).

This would only be possible if some correct process that committed value v other than c changed its vote, but that process would need the same conditions to change, which is a contradiction. □

Lemma 10. *Echidna satisfies Termination.*

Proof. There are two possible cases:

1. No correct process has committed a value v before GST is reached. When GST is reached all correct processes will vote, commit and decide on the value with the highest count.

2. Some correct processes p have committed value v' before GST in round r with proof round pr. There are two possible cases:

   (a) $p \leq f$. There are at least $f+1$ correct processes q that can vote or commit value v, with v.count() $> v'$.count() (if v'.count() $> v$.count() they can never commit v because at least $f + 1$ will never change from value v' to v).

   If q commit, p will eventually receive the proof of that commit and switch their commits from v' to v because the round r is more recent than proof round pr (line 32). Eventually all correct processes decide v.

   If q do not commit, they will eventually receive the messages from round pr, because p will rebroadcast it (line 37), and commit v' because $commited_q = nil$ (line 23). Eventually all correct processes decide v'.

   (b) $p > f$. All correct processes will eventually commit v' because they cannot commit other value because p do not change their commits and either the rule of line 15 or the rule of line 32 will eventually be triggered. Eventually all correct processes decide v'.

□

Lemma 11. *Algorithm 1 satisfies Validity.*

Proof. Trivially follows from the rule at line 16 which ensures that only valid vs can be decided. □

Theorem 3. *Echidna satisfies the Validity Predicate-based Byzantine Consensus.*

Proof. Trivially follows from Lemma 2, Lemma 3 and Lemma 4. □

4.4 State Machine Replication

Our goal is to achieve State Machine Replication (SMR) [Sch90b], which is a commonly used method for creating copies of services modeled as deterministic state machines. The fundamental principle of this method is to ensure that all processes initiate from an identical state and process requests from clients in the same sequence. This way, it is assured that the states of the processes stay consistent and never deviate.

This problem is composed of the following two properties:

- Agreement: All correct processes should receive all requests.
- Order: The sequence of received requests remains consistent across all processes.

This section presents two different approaches to achieve efficient State Machine Replication (SMR) with very low latency, Sphinx and Cerebrus, making them specially suited for decentralized payment systems.

Sphinx has a leader-based design with a single proposer, while Cerebrus employs a leaderless strategy with multiple proposers. Both designs were implemented and their performance was evaluated in comparison with state-of-the-art decentralized solutions. Results showed that Sphinx and Cerebrus not only deliver comparable performance to them in terms of latency, but also present significant robustness under faults, while achieving enough scalability to handle typical usage levels of centralized payment systems.

This section presents the algorithms Spinx and Cerebrus, also with formal proofs of correctness, a description of how they can be applied to decentralized payment systems and an implementation and evaluation of both algorithms.

4.4.1 Sphinx: Leader-Based State Machine Replication

In every round of the Sphinx algorithm 10, a leader is chosen who has the responsibility of suggesting a value. This suggested value is then transferred to the associated Echidna consensus instance. If the leader fails to suggest a value before the deadline, a new round begins, electing a new leader until a value is suggested.

Depending on the network conditions, it is feasible that proposed values may conflict; however, the Echidna consensus will resolve this conflict. When the related Echidna consensus instance delivers a value, the Sphinx consensus instance concludes, making this value the final decision. Following this, a fresh instance commences.

4.4.1.1 Correctness Proofs

We now prove formally that Sphinx solves the State Machine Replication problem.

Algorithm 10: Sphinx: Leader-Based State Machine Replication

```
1  Initialization:
2      instance_p := 0
3      decisions_p[] := nil
4      proposeTimer := nil
5      proposal_p := nil
6  upon start do Start(0)
7  Function Start(round) :
8      round_p ← round
9      if proposer(instance_p, round_p) = p then
10         proposal ← getValue()
11         broadcast <PROPOSAL, instance_p, proposal>
12     else
13         schedule OnTimeoutPropose(instance_p) to be executed after
       timeoutPropose()
14 upon <PROPOSAL, instance_p, proposal> for the first time do
15     proposal_p ← Echidna.StartRound(instance_p, proposal)
16     decisions_p[instance_p] ← proposal_p
17     instance_p ← instance_p + 1
18     reset initial values and empty message log
19     StartRound(0)
20 Function OnTimeoutPropose(height) :
21     StartRound(round_p + 1)
```

Lemma 12. *Sphinx satisfies Agreement.*

Proof. Since Echidna satisfies Agreement and Termination, all correct processes will receive the same decided values. □

Lemma 13. *Sphinx satisfies Order.*

Proof. Echidna consensus instances are executed sequentially (line 17), so all correct processes will receive the same decided values in the same order. □

Theorem 4. *Sphinx satisfies State Machine Replication.*

Proof. Trivially follows from Lemma 5 and Lemma 6. □

4.4.2 Cerebrus: Leaderless State Machine Replication

Contrary to Sphinx, in Cerebrus (algorithm 23) every correct process can propose a value. Once a correct process has received at least 2f + 1 proposals and the timer has expired, it passes that set of proposals as a value to the corresponding Echidna consensus instance, which will resolve conflicts between different sets of proposals and return only one. This will be the decided value of the consensus instance and a new one will start.

4.4.2.1 Correctness Proofs

We now prove formally that Cerebrus solves the State Machine Replication problem.

Lemma 14. *Cerebrus satisfies Agreement.*

Proof. Since Echidna satisfies Agreement and Termination, all correct processes will receive the same decided values. □

Lemma 15. *Cerebrus satisfies Order.*

Proof. Echidna consensus instances are executed sequentially (line 19), so all correct processes will receive the same decided values in the same order. □

Theorem 5. *Cerebrus satisfies State Machine Replication.*

Proof. Trivially follows from Lemma 7 and Lemma 8. □

Algorithm 11: Cerebrus: Leaderless State Machine Replication

1 **Initialization:**
2 $instance_p := 0$
3 $decisions_p[] := nil$
4 $proposals_p[] := nil$
5 $proposeTimer := nil$
6 $value_p := nil$
7 **upon** start **do** *Start*()
8 **Function** *Start()* :
9 $v \leftarrow getValue()$
10 **broadcast** <PROPOSAL, $instance_p, v$>
11 $proposeTimer \leftarrow running$
12 **schedule** $OnTimeoutPropose(instance_p)$ to be executed **after** $timeoutPropose()$
13 **upon** <PROPOSAL, $instance_p, proposal_q$> **do**
14 **add** $proposal_q$ to $proposals_p[]$
15 **upon** $2f + 1$ <PROPOSAL, $instance_p, *$> **AND** $proposeTimer = expired$ for the first time **do**
16 $value_p \leftarrow proposals_p[]$
17 $value_p \leftarrow$ Echidna.$StartRound(instance_p, value_p)$
18 $decisions_p[instance_p] \leftarrow value_p$
19 $instance_p \leftarrow instance_p + 1$
20 **reset** initial values and **empty** message log
21 $StartRound()$
22 **Function** *OnTimeoutPropose*($height$) :
23 $proposeTimer \leftarrow expired$

4.4.3 Eventual State Machine Replication

In this section, we adapt the Nero consensus algorithm to achieve a variant of state machine replication suited for DAG-based systems called eventual state machine replication. Concretely, we change the definition of replica coordination of [Sch90a] to all non-faulty replicas *eventually* receive and process the same sequence of requests.

This property can be decomposed into two parts, Agreement and Eventual Order: Agreement requires all (non-faulty) replicas to receive all requests, and Eventual Order requires that the order of received requests is eventually the same at all replicas.

In this section, we propose a way to eventually order the transactions while maintaining the properties of the consensus algorithm. We do this by modifying the contents of the messages sent (in this case, the transactions), with the client adding a timestamp to it, and modifying the valid() predicate accordingly. In addition to the previous validation conditions, the node now has to also validate the timestamp of the transaction according to its own local timestamp when it receives it in order to discard invalid timestamps. Concretely, a timestamp t of a transaction is considered valid if $l - d < t < l + d$, where l is the local timestamp and d is the value to account for the local timestamps desynchronization of the different nodes and the delay of broadcasting a message through the network. The local timestamps can be synchronized using a decentralized protocol such as Network Time Protocol (NTP) [Mil06] or Network Time Security (NTS) [LD22].

This is not meant to provide an accurate timestamp of when a given transaction happened but only provide a global ordering of the ledger and a common sense of time to the nodes of the network. There is a probability of valid transactions being discarded when the network is not on GST, but they can always be resubmitted with a new timestamp. In order to minimize this, a node can vote on a transaction with an "invalid" timestamp (in its view) if at least $f + 1$ nodes voted on that transaction, meaning that at least one correct process received the transaction in a timely manner.

The ordering of the messages is then performed by its timestamp first and, if there is more than one message, with the same timestamp, by the hash of the message using a hash with the properties of [MVO96]: compression, one way, weak collision resistance and strong collision resistance.

Proof. Assuming the agreement and termination properties of the underlying consensus algorithm, all correct processes will eventually decide on the same set of values. In the case that the values all have different timestamps, the order is naturally done. In the case that different values have the same timestamp, the hash of those values assures a unique deterministic order because any value is hashable (compression property), and each hash is unique in practice (weak collision resistance property).

4.4.4 Application to Decentralized Payment Systems

Both Cerebrus and Sphinx can be employed to achieve consensus in a decentralized payment system. Depending on the specific implementation, the values passed on to the underlying Echina consensus could be valid transactions or ids (hashes) of valid transactions.

The consensus definitions used can be adapted to this use case, in the following way:

- Sphinx satisfies Validity Predicate-based Byzantine Consensus [SGSKK22a] if the decided value satisfies a given predicate valid(), which in this context would be a proposal containing only valid transactions.
- Cerebrus satisfies Set Byzantine Consensus [CNG21] if the decided value is a set of transactions and this set is a valid non-conflicting subset of the union of the decided set of proposals.

In the case of Cerebrus it is still necessary to reconcile the decided set of proposals, removing conflicting transactions, to prevent double-spends.

To ensure a fair process, the initial proposal in the reconciliation process must be randomized, or at the very least, sequenced in a round-robin manner, as suggested in [CNG21].

4.4.5 Implementation and Evaluation

For the purpose of our study, we created two separate branches from a forked version of the Narwhal project [1] in Rust, one for Sphinx and the other for Cerebrus, and we also open-sourced the implementations [2].

Narwhal proved to be an ideal foundation for our work, given its comprehensive benchmarking scripts for performance measurement under a variety of conditions. It is close to a production system, incorporating networking, cryptography, and persistent storage.

We implement three different benchmarking tests with the same number of processes and transaction load but under different conditions: in the common case, under crash-faults and with conflicting transactions. We start by submitting a fixed rate transaction load of 5000 and double up until 80000 transactions for a duration of 1 minute each, with every transaction having 532 bytes and we define 200 ms for the timeout between different rounds. Each process is represented by a e2-medium instance in Google Cloud Platform located in Europe.

In the comparative evaluation, Sphinx and Cerebrus are evaluated against the Bullshark branch, the highest performing algorithm on the repository.

[1] https://github.com/facebookresearch/narwhal
[2] https://github.com/Fiono11/narwhal

4.4.5.1 Benchmark in the common case

When operating under saturation, both Cerebrus and Sphinx display similar performance characteristics due to the former having a dynamic unlimited block size. This implies that Sphinx, even with its single proposer, can accommodate the same volume of transactions as all the proposers of Cerebrus.

The results are shown in figure 4.1 and the data is presented in table 4.2, so that the graph is easier to read. For instance, for the case of 4 Nodes and a load of 80000 transactions per second we have have a latency of approximately 12 seconds per transaction for Cerebus with a throughput of 42k whereas Sphinx has a latency of almost 23 seconds and a throughput of only 20K. Bullshark on the other hand for 4 nodes with the same load has a latency of less than one second and a throughput of 70k.

Summarizing the results in figure 4.1, Cerebrus exhibits superior scalability compared to Sphinx because it distributes the transaction load across all processes rather than consolidating it on a singular leader. Even though Sphinx and Cerebrus do not match Bullshark in terms of scalability, their performance is sufficient for real-world applications such as decentralized payment systems.

It is also worth noting that the saturation point is reached more quickly when there are fewer processes. This is an inherent design feature of the benchmarking tests employed, which distribute the same transaction load across all processes. Thus, with fewer processes, the individual transaction load per process increases, resulting in earlier saturation.

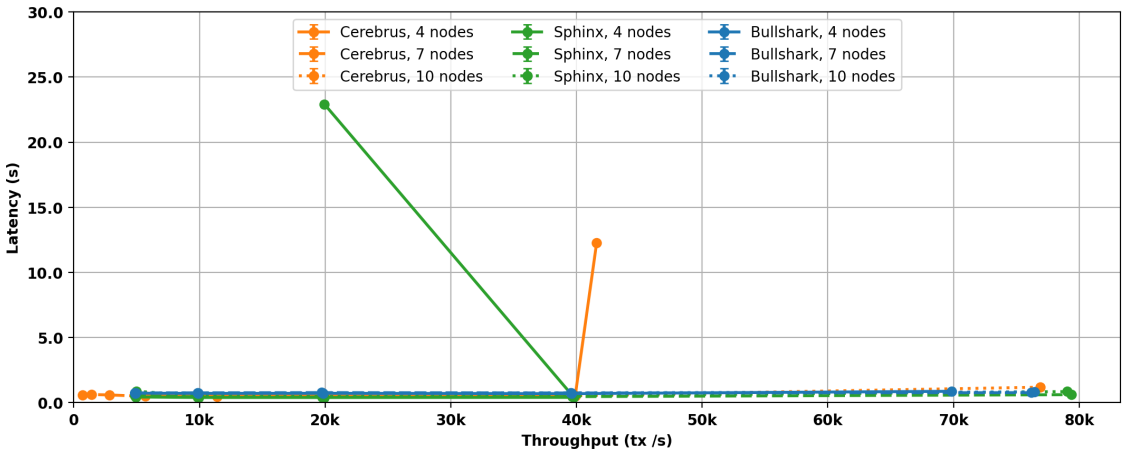

Figure 4.1: Benchmark in the common case

Table 4.2: Figure 4.1 data

Nodes/tps	Cerebrus lat	Cerebrus tps	Sphinx lat	Sphinx tps	Bullshark lat	Bullshark tps
4, 5000	439	1005	508	4994	526	4992
7, 5000	531	1999	504	4914	507	4904
10, 5000	885	4994	499	4728	708	4068
4, 10000	583	5649	504	9807	482	9791
7, 10000	439	3091	418	9986	504	9994
10, 10000	509	9016	477	9124	603	9408
4, 20000	519	12206	495	19317	525	19996
7, 20000	426	19121	508	19312	495	19303
10, 20000	465	19078	538	19212	645	19602
4, 40000	574	39134	499	39033	505	39161
7, 40000	465	39996	511	39992	463	39745
10, 40000	529	40103	492	39851	598	39996
4, 80000	11891	42036	23104	19995	509	69992
7, 80000	615	39991	598	78996	494	75998
10, 80000	854	76992	548	79003	634	75991

4.4.5.2 Benchmark under crash-faults

As can be seen in figure 4.2, Cerebrus demonstrates higher resilience compared to Sphinx when handling crash-faults, primarily because it does not rely on a single leader, thereby eliminating a single point of failure. Furthermore, under conditions of saturation, Cerebrus's performance remains largely unaffected, showing a level of robustness comparable to the state-of-the-art Bullshark consensus algorithm. This underlines Cerebrus's potential as a viable, resilient solution for decentralized systems.

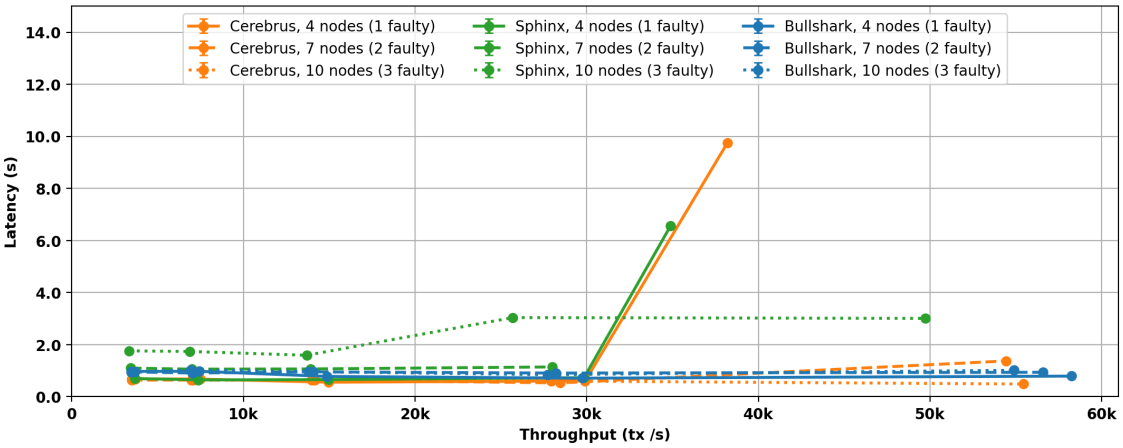

Figure 4.2: Benchmark under crash-faults

4.4.5.3 Benchmark with conflicting values

Figure 4.3 reveals that the scalability of Sphinx and Cerebrus remains on a par when dealing with conflicting transactions. This is attributed to all the proposers in Cerebrus in-

troducing conflicting transactions, thereby negating the advantage of multiple proposers over a single one. However, Sphinx exhibits lower latency, given that all its processes are correct and there is no proposal timeout during the proposal phase, unlike Cerebrus.

Despite demonstrating lower scalability in this scenario, both Sphinx and Cerebrus outperform Bullshark in terms of latency.

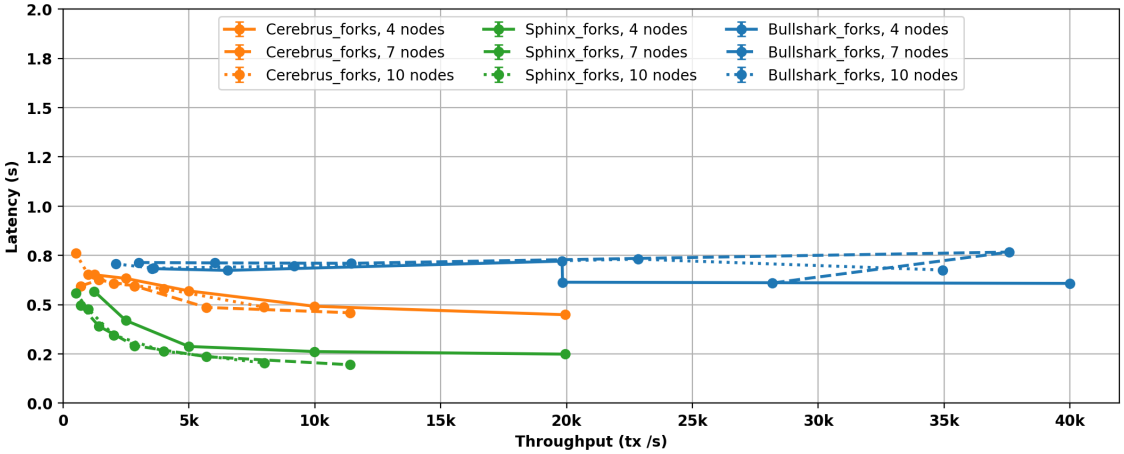

Figure 4.3: Benchmark with conflicting values

4.5 Conclusion

In conclusion, this chapter presented Echidna and Nero, two groundbreaking algorithms for multi-valued consensus, and two distinct strategies based on it, Sphinx and Cerebrus, designed to deliver efficient SMR with extremely low latency. These algorithms show exceptional promise for their use in decentralized payment systems. Sphinx, with its leader-based single proposer design, and Cerebrus, with a leaderless multi-proposer approach, have been implemented and evaluated against current decentralized solutions.

The results indicate that both Sphinx and Cerebrus not only match state of the art in terms of latency but also display substantial resilience in the face of faults. Additionally, they possess the necessary scalability to handle the usage levels typically seen in centralized payment systems. This study underscores the potential of these innovative algorithms in enhancing the efficiency and robustness of decentralized payment systems, laying a solid foundation for future research and development in this field.

Chapter 5

Adamastor: a New Protocol for Permissionless Decentralized Anonymous Payments System

5.1 Introduction

This chapter presents Adamastor, a new low latency and scalable decentralized anonymous payment system, which is an extension of Ring Confidential Transactions (RingCT) that is compatible with consensus algorithms that use Delegated Proof of Stake (DPoS) as a defense mechanism against Sybil attacks.

Adamastor is implemented and evaluated using the Nero consensus algorithm, demonstrating significantly lower latency compared to Proof of Work based cryptocurrencies. Adamastor also exhibits ample scalability, making it suitable for a decentralized and anonymous payment network.

5.1.1 Related Work

Initial attempts to provide anonymized Bitcoin transaction involved mixing the unspent outputs to obfuscate the sender, like TumbleBit [HAB+17] and Coinshuffle [RMSK14], but they offer limited anonymity. Because of that, new cryptocurrencies were developed, such as Monero [Alo18, AHJ17].

Monero was first described in [vS13] and combines linkable ring signatures [TW05] with confidential transactions [PBF+18] into ring confidential transactions. Reference [SALY17] gives the first formal syntax of RingCT and improves it with a new version and [YfSL+19, LRR+19, NG20] improve on the size and the efficiency of the linkable ring signature component and [BBB+18] on the range proof. Compatibility with smart-contracts was achieved in [BAZB20].

ZCash uses zero-knowledge Succinct Non-interactive AR-guments of Knowledge (zk-SNARKs) to construct a decentralised payment scheme [SCG+14], but requires a trusted setup. Since then, other zero-knowledge proofs for arithmetic circuits were developed by improving on efficiency, decreasing the amount of ”trust” required [MBKM19] or increasing the scope of use by adding smart contracts [BCG+20].

Other relevant approaches to anonymous cryptocurrencies are Zerocoin [MGGR13] its evolution Lelantus [Jiv19], Dash, that uses masternodes to provide some degree of anonymity

[**?**], and Ouroboros Crypsinous, the first formal analysis of a privacy-preserving proof-of-stake blockchain protocol [KKKZ19b]. Mimblewimble [SBCL21] uses a different approach to achieve privacy and scalability. It leverages confidential transactions and combines it with a new technique that allows the pruning of old, unspent transaction outputs.

Besides anonymous decentralized payment systems, there has also been research focusing on the development of frameworks to execute smart contracts privately, like Hawk [KMS+16]. Our work, Adamastor, addresses many of the limitations of these works, providing scalable, efficient, and privacy preserving transactions without relying on a trusted setup. It also introduces an effective Decoy Selection Algorithm to counter deanonymization attacks and prune spent outputs, offering an advancement in the secure implementation of privacy-preserving cryptocurrencies.

In the realm of blockchain technology, the intersection of privacy and consensus mechanisms, particularly in Proof-of-Stake (PoS) systems, presents a complex and evolving landscape of research. This work situates itself amidst this intricate domain, drawing upon a rich tapestry of related studies.

The concept of privacy in blockchain systems has been a focal point of earlier research endeavors. Notably, Zcash emerged as a pioneering platform, implementing transactional anonymity and confidentiality through zero-knowledge proofs, albeit on a Proof-of-Work (PoW) basis [SCG+14]. This development, set a precedent for the integration of these cryptographic techniques in blockchain privacy, like Zerocoin [MGGR13].

Parallel to these developments, the PoS consensus mechanism garnered attention, primarily for its energy efficiency, offering a stark contrast to the computationally intensive PoW approach. The foundational principles of PoS, where the consensus is derived from stakeholding as opposed to computational power, were first articulated by PPCoin [KN12]. Later, other works tried to combine this type of consensus with privacy [GOT19].

5.1.2 Technical Overview

In a general way, a transaction protocol defines a set of rules for transferring data (coins) between different parties in a network, which are recorded in a ledger. This typically follows the account model or the UTXO model [ABLZ18], the one that we use in our protocol. These rules are enforced trough a consensus mechanism and are the following:

- Membership: Coins spent in a transaction must already exist in the ledger.
- Ownership: Coin must only be spent by its owner.
- Conservation of Value. The total amount of the coins consumed in a transaction must be equal to the amount of the coins created (plus an optional fee).
- No Double Spending: Coins spent in a transaction must not have been already spent.

A decentralized anonymous payment scheme tries to enforce these rules in a decentralized way and in a way that the amounts of the coins and the relationship between transactions are not revealed. Our goal is to make it compatible with the DPoS family of consensus without breaking these rules, where the weight that a node has on the consensus of the network is proportional to its stake (total value of the coins) on the network and that stake can be delegated to another node, effectively transferring its weight on the consensus to that node.

With this in mind, the concept of stake delegation is introduced by representing the amount of a coin as a Pedersen Commitment (PC), which is additive homomorphic. Both the receiver and the delegate of the coin need to know what is the amount of the coin, the first to be able to spend that coin and the second to be able to prove its delegated stake to the network.

To do that, each output has two ciphertexts that encrypt the value of the corresponding coin, one for the receiver and the other for the delegate, where the key for decrypt the ciphertext is constructed with the transaction public key and the secret key of the receiver or the delegate. This can be seen as a three party Diffie-Hellman exchange [DH76], since there is a three-way shared secret (the value of the coin) between the sender, the receiver and the delegate of the transaction output.

All the inputs (outputs of a previous transaction) of a transaction need to have the same delegate and SimpleDSA chooses the decoys for a given spend transaction. A delegate is able to know the amount of an output that delegated to it, but it does not know what are the sender and receiver master addresses of a transaction.

Since the PC is additive homomorphic and the consensus algorithm only needs to know the total amount of delegated stake to a given delegate, he can reveal the total value of the delegated coins without revealing the individual value of each coin in a reveal transaction.

The owner of a coin can change its delegate anytime in a delegate transaction by producing a new PC together with a Non-Interactive Zero-Knowledge (NIZK) proof that the old and the new commitments have the same value, assuring that no new coins are created.

This chapter is organized as follows: Section II describes the building blocks that are used to develop the Adamastor protocol. Sections III gives a formal definition.Section IV details SimpleDSA. In Section V, the implementation details and evaluation results of Adamastor are presented and the work is concluded in Section VI.

5.2 Building Blocks

5.2.1 Digital Signature Scheme

A Digital Signature scheme is composed of the following algorithms:

- $\text{Setup}(1^\lambda) \rightarrow pp$. A probabilistic algorithm that receives as input a security parameter λ and outputs public parameters pp.
- $\text{KeyGen}(pp) \rightarrow (pk, sk)$. A probabilistic algorithm that receives as input pp, and outputs a public key pk and a secret key sk.
- $\text{Sign}(sk, m) \rightarrow \sigma$. A probabilistic algorithm that receives as input a secret key sk and a message m, and outputs a signature σ.
- $\text{Verify}(pk, m, \sigma) \rightarrow 1/0$. A deterministic algorithm that receives as input a public key pk, a message m, and a signature σ, output "1" if the signature is valid and "0" if it is not.

Existential Unforgeability under Chosen Message Attacks (EUF-CMA). A signature scheme is EUF-CMA secure if no efficient (polynomial-time) adversary can forge a valid signature for a new message, even if it has been able to see valid signatures for messages of its choosing.

5.2.2 Homomorphic Commitment Scheme

A Homomorphic Commitment scheme is composed of the following algorithms:

- $\text{Setup}(1^\lambda) \rightarrow pp$.
- $\text{Com}(m) \rightarrow c$.
- $\text{Open}(c, m, r) \rightarrow 1/0$.

Homomorphism. For any commitment c, any $(m1, r1), (m2, r2) \in MxR$, we have $Com(m1, r1) + Com(m2, r2) = Com(m1 + m2, r1 + r2)$.

Binding. Once a commitment is made to a value, the committer cannot change their committed value later.

Hiding. The commitment does not reveal any information about the committed value before it is opened.

5.2.3 Symmetric Encryption Scheme

A symmetric encryption scheme is a cryptographic system where the same key is used for both encryption and decryption of data. Formally, a symmetric encryption scheme consists of three main algorithms:

- $\text{Setup}(1^\lambda)$: on input a security parameter λ, output public parameters pp.

- $\mathrm{KeyGen}() \rightarrow k$. This algorithm takes a security parameter as input and generates a secret key, denoted as k. The security parameter determines the key size and, consequently, the level of security provided by the encryption scheme.
- $\mathrm{Enc}(k, m) \rightarrow c$. The encryption algorithm takes the secret key k and a plaintext message m as inputs and outputs a ciphertext c. The ciphertext is the encrypted form of the plaintext message, and the process ensures that the information is secure from unauthorized access.
- $\mathrm{Dec}(k, c) \rightarrow m$. The decryption algorithm takes the secret key k and a ciphertext c as inputs and outputs the original plaintext message m. This process allows the intended receiver to recover the original information from the encrypted data.

Indistinguishability under Chosen Plaintext Attack (IND-CPA). A symmetric encryption scheme is IND-CPA secure if no polynomial-time adversary can distinguish between two ciphertexts that encrypt two plaintexts of its choosing, even with access to an encryption oracle.

5.2.4 Non-Interactive Zero-Knowledge Proof System

A Zero-Knowledge Proof (ZKP) is a cryptographic protocol that allows one party (the Prover) to convince another party (the Verifier) that a given statement is true, without revealing any information beyond the fact that the statement is true.

A Sigma protocol reflects the three phases of interaction between the Prover and the Verifier. In such protocols the Prover sends a commitment, often a random value, to the Verifier. The Verifier then responds with a random challenge. In the third phase the Prover computes and sends a response that potentially convinces the Verifier that the prover knows a secret without revealing it.

A Non-Interactive Zero-Knowledge Proof (NIZK) System is a cryptographic proof where the Prover can convince the Verifier that a statement is true in a single round of communication, without requiring any interaction between them after the proof is created.

A NIZK in the random oracle model can be constructed by applying the Fiat-Shamir transform [FS87] to the Σ-protocol by modeling a cryptographic hash function as a random oracle [KL14] and computing $e = \mathcal{H}(m)$. This not only removes interaction, but also strengthens honest-verifier zero-knowledge to full zero-knowledge (i.e. against malicious verifiers). It is composed of the following three polynomial time algorithms.

- $\mathrm{Setup}(1^\lambda) \rightarrow pp$. A probabilistic algorithm that receives as input the security parameter λ and outputs public parameters pp.
- $\mathrm{Prove}(x, w) \rightarrow \pi$. A probabilistic algorithm that receives as input a statement-witness pair (x, w), and outputs a proof π.

- $\text{Verify}(x, \pi) \rightarrow 1/0$. A deterministic algorithm that receives as input a statement x and a proof π, and outputs "0" if rejects and "1" if accepts.

5.2.5 Decoy Selection Algorithm

We adapt the definition of DSA to the context of an anonymous decentralized payment system, and it is composed of the following algorithms:

- $\text{Setup}(1^\lambda) \rightarrow pp$. It takes a security parameter $\lambda \in N$, and outputs the system parameters pp. All algorithms below have implicitly pp as part of their inputs.
- $\text{selectDecoys}(n, m, i) \rightarrow pp$. It takes as input the order i of the real output, the number of inputs in a transaction m and the number of outputs in a transaction n.

Security model. A Digital Signature Algorithm (DSA) is considered secure if it is computationally infeasible for an adversary to distinguish the real input (the actual coin being spent) from the decoys in a ring signature, except with negligible probability in the security parameter λ. This security definition focuses on preventing an adversary from identifying the true transaction among the set of possible transactions included in the ring.

Various attacks aim to compromise this security by exploiting patterns or information leakage in the ring signature. Two notable variants are:

- **Homogeneity Attack.** It is infeasible for an adversary A to determine the historical transaction t' where the real input of a transaction t originated ($t' < t$), except with negligible probability in the security parameter λ.
- **Chain-reaction Analysis.** It is computationally infeasible for an adversary A to determine if an input (decoy) of a transaction t has been previously consumed in a transaction t' ($t' < t$), except with negligible probability in the security parameter λ.

5.2.6 Ring Confidential Transactions Protocol

A RingCT protocol [YfSL+19] consists of the following algorithms:

- $\text{Setup}(1^\lambda) \rightarrow pp$. It takes a security parameter $\lambda \in N$, and outputs the system parameters pp, which include the number of inputs (being one real and the others decoys) and outputs of a transaction. All algorithms below implicitly contain these system parameters, pp, as part of their inputs.
- $\text{LongTermKeyGen}() \rightarrow (ltsk, ltpk)$. This algorithm outputs a long term secret key $ltsk$ and a long term public key $ltpk$.

- $\text{OneTimePKGen}(ltpk) \rightarrow (pk, aux)$. Given a long term public key $ltpk$ as input it outputs a one-time public key pk and the corresponding auxiliary information aux.
- $\text{OneTimeSKGen}(pk, aux, ltsk) \rightarrow sl$. On input of a one-time public key pk, an auxiliary information aux and a long term secret key $ltsk$, it outputs the one-time secret key sk.
- $\text{Mint}(pk, a) \rightarrow (cn, ck)$. Given a one-time public key pk and an amount a as input it outputs a coin cn and the associated coin key ck.
- $\text{AccountGen}(sk, pk, cn, ck, a) \rightarrow (act, ask)$. It takes as input a user key pair (sk, pk), a coin cn, a coin key ck and an amount a. It returns $\perp$ if ck is not the coin key of cn with amount a. Otherwise, it outputs the account $act = (pk, cn)$ and the corresponding account secret key is $ask = (sk, ck, a)$.
- $\text{Spend}(m, ask, A_{\text{in}}, O) \rightarrow (A_{\text{out}}, \pi_{\text{range}}, s, CK_{\text{out}})$. It takes as input a message m, an account secret key ask and a set A_{in}, of input accounts in which the spending account is included and a set of output public keys, O, with the corresponding output amounts. It returns a set of output accounts A_{out}, a range proof π, a serial number s and a set of output coin keys CK_{out}
- $\text{Verify}(m, A_{\text{in}}, A_{\text{out}}, \pi, s) \rightarrow 1/0/-1$. It takes as input a message m, a set of input accounts A_{in}, a set of output accounts A_{out}, a range proof π_{range} and a serial number s, the algorithm outputs -1 if the serial number was spent previously. Otherwise, it checks if the range proof π_{range} is valid for the transaction, and outputs 1 or 0, meaning a valid or invalid spending respectively.

Security model. A RingCT protocol is secure if it is anonymous against receiver and ring insider, unforgeable, equivalent, linkable and non-slanderable [YfSL+19].

5.3 Definition of Adamastor

5.3.1 Data Structures

Ledger. The Adamastor protocol operates on top of a ledger L, which is a publicly accessible and append-only database. At any given time t, all users have access to L_t, which is a sequence of transactions. If $t < t'$, state of L_t is previous to the state of $L_{t'}$.

Public parameters. These include the group in which the algorithms perform operations, generators of the group, cryptographic hash functions and parameters regarding transactions, namely:

- a, which specifies the maximum possible number of coins that the protocol can handle. Any balance and transfer must lie in the integer interval $[0, a]$.

- m, the maximum number of input accounts used as decoys in a spend transaction.
- n, the maximum number of output coins of a spend transaction.

Address. A public key pk. It can be used to transfer the ownership of coins in a spend transaction or to delegate coins in a delegate transaction.

Coin. A commitment of an amount a and a randomness coin key ck.

Account. A data structure composed of a coin cn, a one-time public key of the receiver pk_r, a public key of the delegate pk_d, a ciphertext for the receiver c_r, a ciphertext for the delegate c_d and an auxiliary information aux. The auxiliary information aux allow both the owner of the account (receiver) and its delegate to reveal the amount of the coin and also can be used by the receiver to compute the secret key sk_r of the account, allowing it to spend it. The account secret key ask is composed of the secret key sk_r, the amount a and the coin key ck.

Delegate. A delegate of a coin can stake its value and use it to participate in the consensus of the network to validate transactions. Its weight on the consensus is proportional to the total amount of delegated stake.

Transactions. Structured blocks of data that are validated through the consensus mechanism and recorded on the ledger in the following four types of transactions.

- **Spend.** A spend transaction transfers the ownership of coins by creating new ones with the same value. All input and output coins of a spend transaction must have the same delegate so that the anonymity of the sender and the amounts is maintained.
- **Mint.** A mint transaction creates a new unspent coin through the consensus process.
- **Delegate.** A delegate transaction allows the owner of a coin to change its delegate.
- **Reveal.** A reveal transaction is used by a receiver or a delegate to reveal the amount of an account. It can be used by a delegate to prove its total delegated stake to the network since the commitments used to represent coins are homomorphic.

5.3.2 Algorithms

The Adamastor protocol is composed of the following polynomial-time algorithms.

- $\mathrm{Setup}(1^\lambda) \rightarrow pp$. On input a security parameter λ, it generates the public parameters pp of Adamastor, which are implicit in the rest of the algorithms.
- $\mathrm{LongTermKeyGen}() \rightarrow (ltpk, ltsk)$. Returns the long term public and secret keys $ltpk$ and $ltsk$.

- $\text{OneTimePKGen}(ltpk) \rightarrow (pk, aux)$. On input a long term public key $ltpk$, it outputs a one-time public key pk and the auxiliary information aux.

- $\text{OneTimeSKGen}(pk, aux, ltsk) \rightarrow sk$. On input a one-time public key pk, auxiliary information aux and a long term secret key $ltsk$, it outputs the one-time secret key sk.

- $\text{Mint}(pk_r, a) \rightarrow (cn, ck)$. It takes as input the receiver public key pk_r and an amount a, outputs a coin cn and the associated coin key ck.

- $\text{AccountGen}(sk_r, pk_r, ltpk_d, cn, ck, a, aux) \rightarrow (act, ask) / \perp$. It takes as input a receiver key pair (sk_r, pk_r), a delegate long term public key $ltpk_d$, a coin cn, a coin key ck, an amount a and the auxiliary information aux. It returns $\perp$ if ck is not the coin key of cn with amount a. Otherwise, it outputs the account $act = (pk_r, cn, pk_d, c_r, c_d, aux)$ and the corresponding account secret key is $ask = (sk_r, ck, a)$.

- $\text{Spend}(m, i, ask, O, L) \rightarrow (A_{out}, \pi_{\text{range}}, s, CK_{out}) / \perp$.

  It takes as input a message m, the index i of the account to be spent in the ledger L, the corresponding account secret key ask, the set of output accounts with the corresponding output amounts O and the ledger L. It outputs $\perp$ if the sum of output amounts in O is different from the input amount or the ask is not valid. Otherwise it returns the output accounts A_{out}, a range proof π_{range}, a serial number s and a set of output coin keys CK_{out}.

- $\text{VerifySpend}(m, A_{\text{in}}, A_{\text{out}}, \pi_{\text{range}}, s) \rightarrow$ 1/0/-1. it takes as input a message m, a set of input accounts A_{in}, a set of output accounts A_{out}, a range proof π_{range} and a serial number s, the algorithm outputs -1 if the serial number has been spent previously. Otherwise, it checks if the range proof π_{range} is valid for the transaction, and outputs 1 or 0, meaning a valid or invalid spending, respectively.

- $\text{Delegate}(m, act, ask, ltpk'_d) \rightarrow (act', aux', \pi, \sigma)$. On input a message m, an account $act = (pk_r, cn, ltpk_d, c_r, c_d, aux, ma)$, the corresponding account secret key $ask = (sk_r, ck, a)$ and a new delegate long term public key $ltpk'_d$, it outputs a new account $act' = (pk_r, cn', ltpk'_d, c_r, c'_d, aux')$, a proof of equivalence π and a signature σ.

- $\text{VerifyDelegate}(act, \sigma) \rightarrow$ 1/0. It receives an account act and a signature σ and returns "1" if it is is valid and "0" if not.

- $\text{Reveal}(m, ltsk_{r/d}, act) \rightarrow (a, ck)$. On input a message m, a receiver or delegate long term secret key $ltsk_{r/d}$ and an account act, it returns the amount a and the coin key ck of the coin of that account. This algorithm serves as a proof of delegated stake for the delegate.

- $\text{VerifyReveal}(cn, a, ck) \rightarrow$ 1/0. On input a transaction a coin cn, it outputs "1" if the amount a and coin key ck correspond to the coin cn and "0" if not.

5.3.3 Security Model

We use the security model of [YfSL+19] and extend it to reflect the additions of Adamastor. The model consists of a polynomial time adversary that tries to break some security property by winning the corresponding security game. In the games, the adversary interacts with oracles that simulate the Adamastor protocol, which are the following:

- LongTermPKGen(i). On input a query number i, it runs the algorithm LongTermKeyGen() $\rightarrow (ltsk_i, ltpk_i)$, adds $(i, ltsk_i, ltpk_i)$ to an initially empty list and returns the long term public key $ltpk_i$.

- OneTimePkGen(i, j). It retrieves $(i, ltsk_i, ltpk_i)$ from U' and runs OneTimePKGen $(ltpk_i) \rightarrow (pk_{i,j}, R_{i,j})$. It adds $(i, j, *, pk_{i,j}, aux_{i,j})$ to an initially empty list and returns $pk_{i,j}$.

- OneTimeSkGen(i, j). It retrieves $ltsk_i$ from U and $pk_{i,j}$ and $aux_{i,j}$ from U' and runs OneTimeSKGen($pk_{i,j}, aux_{i,j}, ltsk_i$) $\rightarrow sk_{i,j}$. It updates (i, j, $sk_{i,j}$, $pk_{i,j}$, $aux_{i,j}$) the corresponding list.

- ActGen($pk, a, ltpk_d$). On input a receiver public key $pk_r \in U$, an amount a and a delegate long term public key $ltpk_d \in U'$, it runs the algorithm Mint($a, pk_r, ltpk_d$) $\rightarrow (cn, ck)$, and outputs (act, ck) for address pk_r. It adds (act, a, ck) to an initially empty list G. If $(sk, pk) \in U$, then the associated secret key with account act is $ask = (sk, ck, a)$. It adds act and (act, ask) to initially empty lists.

- Corrupt(act): on input an account act, it retrieves $(act, ask) \in L$. It adds act to an initially empty list, and finally returns ask.

- Spend(m, i, O, L): takes in a message m, the index i of the account act to be spent in the ledger L and the output set O. It gets act from the ledger, retrieves (act, ask) from list I and runs Spend$(m, i, ask, O, L) \rightarrow (A_{\text{out}}, \pi, s, CK_{\text{out}})$ and returns the result after adding it to an initially empty list.

- Delegate($act, ltpk'_d$). On input an account act and a delegate long term public key $ltpk_d$, it retrieves the associated secret key ask and runs the algorithm Delegate(act, ask, $ltpk'_d$).

- Reveal(act). On input an account act, it retrieves the associated receiver or delegate long term secret key and runs the algorithm Reveal($ltsk_{r/d}, act$).

Anonymity against receiver. The definition of anonymity against receiver ensures that, without the knowledge of the spender's account secret key or the delegate's secret key, the spender's account is successfully hidden among all the honestly generated accounts, even if the output accounts and the output amounts are known.

Definition 1. *The Adamastor protocol is anonymous against receiver if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger $\mathcal{CH}$:*

1. **Setup:** $\mathcal{CH}$ runs Adamastor.Setup(1^λ) $\rightarrow pp$ and sends pp to A.
2. **Pre-challenge queries:** The adversary can query all previously defined oracles.
3. **Challenge:** In the challenge phase, the adversary gives to $\mathcal{CH}$.

   $(m, pk_{k,i}, pk_{out,j})$ $k \in [1, M], i \in [1, n], j \in [1, N]$

   The adversary is given $A^*_{out}, \pi^* = (\pi_{range}, \sigma_{ring}), S^*$ and A_{in}, where S^* is the set of serial numbers and A_{in} is the set of input accounts; also n is the size of the ring N is the number of outputs.
4. **Post-challenge queries:** After receiving the challenge, A can continue querying the same oracles, except corrupt accounts that were used in the challenge phase.
5. **Guess:** The adversary outputs $(k^*, {ind_k}^*)$. With probability $1/n$, $k' = k^*$. The adversary wins with probability $1/n$, if ${ind_k}^* = {i_k}^*$.

Anonymity against ring insider. The definition of anonymity against ring insider ensures that, without the knowledge of the output accounts secret keys or the delegate's secret key, the spender's account is successfully hidden among all corrupted accounts.

Definition 2. *The Adamastor protocol is anonymous against ring insider if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger $\mathcal{CH}$:*

1. **Setup**: $\mathcal{CH}$ runs Adamastor.Setup(1^λ) $\rightarrow pp$ and sends pp to A.
2. **Pre-challenge queries:** The adversary can query all previously defined oracles.
3. **Challenge:** In the challenge phase, the adversary gives $(m, act_{k,i}, pk_{out,j})$ to $\mathcal{CH}$, $k \in [1, M], i \in [1, n], j \in [1, N]$. The adversary is given $A^*_{out}, \pi^* = (\pi_{range}, \sigma_{ring}), S^*$ and A_{in}, where S^* is the set of serial numbers and A_{in} is the set of input accounts.
4. Post-challenge queries: After receiving the challenge, A can continue querying the same oracles, including the corrupt oracle, but the accounts that these oracles return cannot be corrupted, i.e., the inputs cannot be the accounts of the challenge.
5. **Guess:** The adversary outputs $(k^*, {ind_k}^*)$. The adversary wins with probability $1/(n - n^*)$, if ${ind_k}^* = {i_k}^*$ and Ain $\cap$ C = $\emptyset$, where there are n^* distinct values of i such that $act_{k^*,i} \in C$.

Unforgeability. The unforgeability property ensures that no polynomial time adversary can forge a Spend transaction, a Delegate transaction or a Reveal transaction if all input accounts are not corrupted.

Definition 3. *The Adamastor protocol is unforgeable if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger $\mathcal{CH}$:*

1. **Setup**: $\mathcal{CH}$ runs Adamastor.Setup$(1^\lambda) \rightarrow pp$ and sends pp to A.
2. **Queries:** The adversary can query all previously defined oracles.
3. **Forge:** A outputs a spend, delegate or reveal transaction such that no input account is corrupted and wins if the transaction is valid.

Equivalence. The equivalence property ensures that no polynomial time adversary can output a valid signature and coin keys where the input amount is different from the sum of the output amounts, even if the accounts are corrupted.

Definition 4. *The Adamastor protocol is equivalent w.r.t. insider corruption if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger $\mathcal{CH}$:*

1. **Setup**: $\mathcal{CH}$ runs Adamastor.Setup$(1^\lambda) \rightarrow pp$ and sends pp to A.
2. **Queries:**: The adversary can query all previously defined oracles.
3. **Output:** A outputs a spend transaction with an input amount different than the sum of the output amounts and wins if the transaction is valid.

Linkability. The linkability property ensures that no polynomial time adversary can output two valid spend transactions with the same serial number.

Definition 5. *The Adamastor protocol is linkable w.r.t. insider corruption if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger$\mathcal{CH}$:*

1. **Setup**: $\mathcal{CH}$ runs Adamastor.Setup$(1^\lambda) \rightarrow pp$ and sends pp to A.
2. **Queries:** The adversary can query all previously defined oracles.
3. **Output:** A outputs two spend transactions with the same serial number and wins if they are both valid.

Non-Slanderability. The definition of non-slanderabiliy ensures that no polynomial time adversary can make a spend transaction that is linkable with another spend transaction of an honest user, i.e., an adversary cannot produce a valid spend transaction with the same serial number of a previously honest spending.

Definition 6. *The Adamastor protocol is non-slandarable if a polynomial time adversary A has a negligible probability of winning the following security experiment with a challenger $\mathcal{CH}$:*

1. **Setup**: $\mathcal{CH}$ runs Adamastor.Setup(1^λ) $\rightarrow pp$ and sends pp to A.
2. **Queries:** Same as in **Definition 1**.
3. **Output:** A outputs a spend transaction with the same serial number of a previously honest spending and wins if the transaction is valid.

5.4 Instantiation of Adamastor

An instantiation of Adamastor from the building blocks presented in section 2 is now given, in the following way.

- Let HC = (Setup, Com, Open, Prove, Verify) be a Homomorphic Commitment scheme, instantiated by the Pedersen Commitment scheme [Ped92].
- Let SE = (Setup, KeyGen, Enc, Dec) be a Symmetric Encryption scheme, instantiated by AES/GCM.
- Let S = (Setup, KeyGen, Sign, Verify) be a Signature scheme, instantiated by the Schnorr Signature scheme [Sch04].
- Let NIZK = (Setup, CRSGen, Prove, Verify) be a NIZK proof system for the equivalence of two discrete logarithms, formally defined as $(g_1,h_1,g_2,h_2) \mid \exists\ w \in Z_p \quad$ s.t. $\log_{g1}\ h1 = w = \log_{g2}\ h2$, instantiated by the Chaum-Pedersen scheme [CP92].
- Let RCT = (Setup, KeyGen, Mint, Spend, AccountGen, Verify) be a Ring Confidential Transaction, instantiated by the RCT3.0 scheme [YfSL$^+$19].
- Let DSA = (Setup, KeyGen, Mint, Spend, AccountGen, Verify) be a Decoy Selection Algorithm, instantiated by the Simple Decoy Selection Algorithm.
- Let H be a cryptographic hash function H : M x $G \rightarrow Z_p$, where M denotes the message space, instantiated by SHA-256.

The protocol is composed of the following algorithms.

- $\text{Setup}(1^\lambda) \rightarrow pp$. On input a security parameter λ, it generates the public parameters pp for HC, SE, NIZK, S, RCT and DSA, and returns them. They are implicit in the rest of the algorithms.
- $\text{LongTermKeyGen}() \rightarrow (ltpk, ltsk)$. It runs $\text{RCT.LongTermKeyGen}() \rightarrow (ltpk, ltsk)$.
- $\text{OneTimePKGen}(ltpk) \rightarrow (pk, aux)$. It runs $\text{RCT.OneTimePKGen}(ltpk) \rightarrow (ltpk, ltsk)$.
- $\text{OneTimeSKGen}(pk, aux, ltsk) \rightarrow (sk)$. It runs $\text{RCT.OneTimeSKGen}(pk, aux, ltsk) \rightarrow (ltpk, ltsk)$.
- $\text{Mint}(pk, a) \rightarrow (cn, ck)$. It takes as input a public key pk and an amount a, and runs:
  - Picks $ck \xleftarrow{R} Z_p$.
  - $\text{HC.Com}(a, ck) \rightarrow cn$.

  It outputs a coin cn for pk as well as the associated coin key ck.
- $\text{AccountGen}(sk_r, pk_r, ltpk_d, cn, ck, a, aux) \rightarrow (act, ask) / \perp$: it takes as input a user key pair (sk, pk), a coin cn, a coin key ck, an amount a, a delegate long-term public key pk_d. It runs:
  - $pk_d \leftarrow ltpk_d.0$.
  - $\text{SE.Enc}(k_r, m) \rightarrow c_r$, where $k_r = sk_r * aux$.
  - $\text{SE.Enc}(k_d, m) \rightarrow c_d$, where $k_d = sk_d * aux$.

  It returns $\perp$ if ck is not the coin key of cn with amount a. Otherwise, it outputs the account $act = (pk, aux, cn, c_r, c_d)$ and the corresponding account secret key is $ask = (sk, ck, a)$.
- $\text{Spend}(m, i, ask, O, L) \rightarrow (A_{\text{out}}, \pi_{\text{range}}, s, CK_{\text{out}})$.

  It takes as input some transaction string $m \in 0, 1^*$, the index i of account act in the ledger L together with the corresponding account secret key ask, an arbitrary set A_{in} of groups of input accounts containing act, a set O of output public keys with the corresponding output amounts and a ledger with past confirmed transactions L. It runs:
  - $\text{DSA.selectDecoys}(n, m, i) \rightarrow js$, where n and m are the number of outputs and the number of inputs in the spend transaction, respectively, and they are both public parameters of Adamastor .
  - $\text{L.get}(i) \rightarrow act$.
  - $\text{L.get}(js) \rightarrow A_{\text{in}}$.
  - $\text{RCT.Spend}(m, ask, A_{\text{in}}, O) \rightarrow (A_{\text{out}}, \pi_{\text{range}}, s, CK_{\text{out}})$.

It outputs a set of output accounts A_{out}, a range proof π, a serial number s and a set of output coin keys CK_{out}.

- $\text{VerifySpend}(m, A_{\text{in}}, A_{\text{out}}, \pi_{\text{range}}, s) \rightarrow 1/0/-1$: it takes as input a message m, a set of input accounts A_{in}, a set of output accounts A_{out}, a range proof π_{range} and a serial number s.

  It runs $\text{RCT.Verify}(m, A_{in}, A_{out}, \pi_{\text{range}}, s)$ and outputs -1 if the serial number was already spent previously. Otherwise, it checks if the range proof π_{range} is valid for the transaction, and outputs 1 or 0, meaning a valid or invalid spending respectively.

- $\text{Delegate}(m, act, ask, ltpk'_d) \rightarrow (\pi_{\text{equal}}, \sigma)$. On input a message m, an account act, the secret key of the account ask and the new delegate public key $ltpk'd$, it runs:

  - $pk'_d \leftarrow ltpk'_d.0$.
  - Picks a random new coin key ck' and computes the new coin $cn' \leftarrow \text{HC.Com.}(act.a, ck')$.
  - $\text{NIZK.Prove}((h, cn, h, cn'), (ck - ck')) \rightarrow \pi_{\text{equal}}$.
  - $\text{SE.Enc}(k_r, act.c_r) \rightarrow m$, where $k_r = act.pk_r * act.aux$.
  - $\text{SE.Enc}(k_d, m) \rightarrow c_d$, where $k_d = pk'_d * act.aux$.
  - $\text{S.Sign}(sk, m) \rightarrow \sigma$.

  It returns the equivalence proof π_{equal}, the new delegate ciphertext c'_d and the signature σ.

- $\text{VerifyDelegate}(act, act', \pi_{\text{equal}}, \sigma) \rightarrow 1/0$. On input two accounts act and act', an equivalence proof π_{equal} and a signature σ it runs:

  - $\text{NIZK.Verify}((h, act.cn, h, act'.cn), \pi_{\text{equal}}) \rightarrow 1/0$.
  - $\text{S.Verify}(act.pk, \pi_{\text{equal}}, \sigma) \rightarrow 1/0$.

  It returns "1" if both return "1" and "0" otherwise.

- $\text{Reveal}(m, act, sk_{r/d}, aux) \rightarrow (a, ck)$. On input an account act, a receiver or delegate one-time secret key $sk_{r/d}$, it runs:

  - $\text{SE.Dec}(k_{r/d}, act.c_{r/d}) \rightarrow m$, where $k_{r/d} = aux * sk_{r/d}$.
  - H("key", m) $\rightarrow ck$.
  - $ma \oplus_8$ H("amount", m) $\rightarrow a$. The masked amount ma is also included in the transaction by the sender.
  - $\text{S.Sign}(sk_{r/d}, m) \rightarrow \sigma$.

  It returns the amount a and the coin key ck together with the signature σ.

- $\text{VerifyReveal}(cn, a, ck, pk_d, \sigma) \rightarrow 1/0$. On input a coin cn, an amount a, a coin key ck, a delegate public key pk_d and a signature σ, it runs:
  - $\text{HC.Verify}(act.cn, a, ck) \rightarrow 1/0$.
  - $\text{S.Verify}(pk_d, m, \sigma) \rightarrow 1/0$.

  It returns 1 if both algorithms return 1, and 0 otherwise.

5.4.1 Security Proofs

We begin by explaining the proofs informally and defer the formal proofs to the next section.

The intuition for the anonymity, linkability, equivalence and non-slanderability is the following:

Since the underlying RCT component defines these properties in the same way, i.e., the output of the adversary is the same, and these are proven secure, we want to reduce the security experiment of Adamastor to the one of RingCT. If we can make this reduction so that the view of the adversary A is not altered, the probability that A has of winning will be the same in both experiments. Since this probability is negligible for the experiment of the RingCT's property, it will also be negligible for the experiment of the Adamastor's property, proving its safety.

Since the only difference between the two security experiments is the access to new oracles that mimic the added functionality to the protocol, we can simulate these oracles so that they produce the same results but they do not use the real added underlying algorithms because the challenger $\mathcal{CH}$ of the experiment of RingCT does not have access to those algorithms.

The property of unforgeability is different because it captures the unforgeability of all transactions (Spend, Delegate and Reveal) and some of the output that A can produce is directly related to the added functionality of Adamastor.

If the output is a Spend transaction, we can follow the same idea as the other security properties. If the output is a Delegate or Reveal transaction, we cannot do that because RingCT does not have that functionality, so we reduce the security experiment to some underlying secure problem where the output of the Adamastor experiment can be used to try to break its security.

In the case of the Delegate transaction it is the Schnorr Signature scheme and in the case of the Reveal transaction it is the hiding property of the Pedersen Commitment. To do that we need to be able to simulate all the other components of Adamastor.

Theorem 6. *Assuming the security of the S component, the security of the* RCT *component, the security of the* DSA *component, the security of the* SE *component, the security*

of the HC *component and the zero-knowledge property of* NIZK*, our* Adamastor *construction is anonymous against ring insider.*

Proof. We proceed via a sequence of games. Let S_i be the probability that A wins in Game i.

Game 0. The real experiment for anonymity against ring insider as described in the security model of section 3.3.

Game 1. The same as Game 0 except that the oracles of Adamastor that are not present in the security model of RCT are simulated by $\mathcal{CH}$ instead of using the real algorithms, in the following way:

- Delegate: $\mathcal{CH}$ can easily simulate the NIZK and the encryption part. The signing part is simulated as described in the security proof of the Integrated Signature Encryption scheme in [CMTA20], which itself includes a sketch of the security proof due to [PS00].
- Reveal: $\mathcal{CH}$ stores retrieves the amount a and coin key ck previously stored when running the ActGen oracle.

$\mathcal{A}$'s view of Game 1 is identical to Game 0 because $\mathcal{CH}$ perfectly mimics the oracles that use the simulated components of Adamastor and those components are secure. So, we have:

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 16. *Assuming the anonymity against ring insider of the* RCT *component, Pr[*G_1*] is negligible in* λ *for any* polynomial time *adversary A.*

Proof. We prove this lemma by showing that if there exists a polynomial time adversary $\mathcal{A}$ that has non-negligible advantage in Game 1, we can build a polynomial time adversary $\mathcal{B}$ that breaks the anonymity against ring insider of the RCT component with the same advantage since $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the queries related to RCT to its own challenger and simulating the rest of the components of Adamastor. The guess of $\mathcal{A}$ is used by $\mathcal{B}$ in its own game and wins with the same probability as $\mathcal{A}$.

The security of the DSA component ensures an adversary cannot increase its probability of winning by eliminating decoys from the ring signature of a spend transaction. □

The proof of the theorem follows directly from the lemma. □

Theorem 7. *Assuming the security of the* S *component, the security of the* RCT *component, the security of the* SE *component, the security of the* HC *component and the*

zero-knowledge property of NIZK*, our* Adamastor *construction is anonymous against receiver.*

Proof. We proceed via a sequence of games. Let S_i be the probability that A wins in Game i.

Game 0. The real experiment for anonymity against receiver as described in the security model of section 4.3.

Game 1. The same as Game 0 except that the oracles of Adamastor that are not present in the security model of RCT are simulated by $\mathcal{CH}$ instead of using the real algorithms, in the following way:

- Delegate: $\mathcal{CH}$ can easily simulate the NIZK and the encryption part. The signing part is simulated as described in the security proof of the Integrated Signature Encryption scheme in [CMTA20].
- Reveal: $\mathcal{CH}$ stores retrieves the amount a and coin key ck previously stored when running the ActGen oracle.

$\mathcal{A}$'s view of Game 1 is identical to Game 0 because $\mathcal{CH}$ perfectly mimics the oracles that use the simulated components of Adamastor and those components are secure. So, we have:

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 17. *Assuming the anonymity against ring insider of the* RCT *component, Pr[G_1] is negligible in λ for any* polynomial time *adversary A.*

Proof. We prove this lemma by showing that if there exists a polynomial time adversary $\mathcal{A}$ has non-negligible advantage in Game 1, we can build a polynomial time adversary $\mathcal{B}$ that breaks the anonymity against ring insider of the RCT component with the same advantage since $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the queries related to RCT to its own challenger and simulating the rest of the components of Adamastor. The guess of $\mathcal{A}$ is used by $\mathcal{B}$ in its own game and wins with the same probability as $\mathcal{A}$.

The security of the DSA component ensures an adversary cannot increase its probability of winning by eliminating decoys from the ring signature of a spend transaction. □

The proof of the theorem follows directly from the lemma. □

Theorem 8. *Assuming the security of the S component, the security of the RCT component, the security of the SE component, the security of the HC component and the zero-knowledge property of NIZK, our Adamastor construction satisfies unforgeability.*

Proof. We proceed via a sequence of games. Let Si be the probability that A wins in Game i.

Game 0. The real experiment for unforgeability as described in the security model of section 3.3.

Game 1. The same as game 0 except for:

2. Queries: in one query to the ActGen oracle $\mathcal{CH}$ picks a random public key pk^* without knowing the corresponding secret key sk^* and the coin cn^* is a random commitment. It outputs the account act^*.

   If the Spend oracle is queried with an account act whose account secret key ack is not known to $\mathcal{CH}$, the Spend transaction is simulated as in the proof of unforgeability of [YfSL$^+$19] (the ciphertexts are picked at random) and the Delegate and Reveal oracles are simulated as in the previous anonymity proof.

A's view in Game 0 and Game 1 are identical because $\mathcal{CH}$ can perfectly mimic the Spend, Delegate and Reveal oracles.

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 18. *Assuming the EUF-CMA security of the Schnorr Signature scheme, the hiding property of the Pedersen Commitment scheme and the unforgeability of the RCT3.0 scheme, Pr[G_1] is negligible in λ for any polynomial time adversary $\mathcal{A}$.*

Proof. We prove this claim by showing that if there exists a polynomial time adversary A has non-negligible advantage in Game 1, we can build a polynomial time adversary B that breaks either the EUF-CMA security of the Schnorr Signature, the hiding property of the Pedersen Commitment scheme or the unforgeability of the RCT3.0 scheme with the same advantage if the forge of A is a Delegate, a Reveal or a Spend, respectively. □

The proof of the theorem follows directly from the lemma. □

Theorem 9. *Assuming the security of the S component, the security of the RCT component, the security of the SE component, the security of the HC component and the zero-knowledge property of NIZK, our Adamastor construction satisfies equivalence w.r.t. ring insider.*

Proof. We proceed via a sequence of games. Let S_i be the probability that A wins in Game i.

Game 0. The real experiment for anonymity against ring insider as described in the security model of section 3.3.

Game 1. The same as Game 0 except that the oracles of Adamastor that are not present in the security model of RCT are simulated by $\mathcal{CH}$ instead of using the real algorithms, in the following way:

- Delegate: $\mathcal{CH}$ can easily simulate the NIZK and the encryption part. The signing part is simulated as described in the security proof of the Integrated Signature Encryption scheme in [CMTA20].
- Reveal: $\mathcal{CH}$ stores retrieves the amount a and coin key ck previously stored when running the ActGen oracle.

$\mathcal{A}$'s view of Game 1 is identical to Game 0 because $\mathcal{CH}$ perfectly mimics the oracles that use the simulated components of Adamastor and those components are secure. So, we have:

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 19. *Assuming the anonymity against ring insider of the* RCT *component, Pr[G_1] is negligible in λ for any* polynomial time *adversary $\mathcal{A}$.*

Proof. We prove this lemma by showing that if there exists a polynomial time adversary $\mathcal{A}$ has non-negligible advantage in Game 1, we can build a polynomial time adversary $\mathcal{B}$ that breaks the anonymity against ring insider of the RCT component with the same advantage since $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the queries related to RCT to its own challenger and simulating the rest of the components of Adamastor. The guess of $\mathcal{A}$ is used by $\mathcal{B}$ in its own game and wins with the same probability as $\mathcal{A}$. □

The proof of the theorem follows directly from the lemma.

□

Theorem 10. *Assuming the security of the S component, the security of the RCT component, the security of the SE component, the security of the HC component and the zero-knowledge property of NIZK, our Adamastor construction satisfies linkability w.r.t. ring insider.*

Proof. We proceed via a sequence of games.

Let S_i be the probability that A wins in Game i.

Game 0. The real experiment for anonymity against ring insider as described in the security model of section 3.3.

Game 1. The same as Game 0 except that the oracles of Adamastor that are not present in the security model of RCT are simulated by $\mathcal{CH}$ instead of using the real algorithms, in the following way:

- Delegate: $\mathcal{CH}$ can easily simulate the NIZK and the encryption part. The signing part is simulated as described in the security proof of the Integrated Signature Encryption scheme in [CMTA20].

- Reveal: $\mathcal{CH}$ stores retrieves the amount a and coin key ck previously stored when running the ActGen oracle.

$\mathcal{A}$'s view of Game 1 is identical to Game 0 because $\mathcal{CH}$ perfectly mimics the oracles that use the simulated components of Adamastor and those components are secure. So, we have:

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 20. *Assuming the anonymity against ring insider of the* RCT *component, Pr[G_1] is negligible in λ for any* polynomial time *adversary A.*

Proof. We prove this lemma by showing that if there exists a polynomial time adversary $\mathcal{A}$ has non-negligible advantage in Game 1, we can build a polynomial time adversary $\mathcal{B}$ that breaks the anonymity against ring insider of the RCT component with the same advantage since $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the queries related to RCT to its own challenger and simulating the rest of the components of Adamastor. The guess of $\mathcal{A}$ is used by $\mathcal{B}$ in its own game and wins with the same probability as $\mathcal{A}$. □

The proof of the theorem follows directly from the lemma. □

Theorem 11. *Assuming the security of the S component, the security of the RCT component, the security of the SE component, the security of the HC component and the zero-knowledge property of NIZK, our Adamastor construction satisfies non-slanderability.*

Proof. We proceed via a sequence of games.

Let S_i be the probability that A wins in Game i.

Game 0. The real experiment for anonymity against ring insider as described in the security model of section 3.3.

Game 1. The same as Game 0 except that the oracles of Adamastor that are not present in the security model of RCT are simulated by $\mathcal{CH}$ instead of using the real algorithms, in the following way:

- Delegate: $\mathcal{CH}$ can easily simulate the NIZK and the encryption part. The signing part is simulated as described in the security proof of the Integrated Signature Encryption scheme in [CMTA20].

- Reveal: $\mathcal{CH}$ stores retrieves the amount a and coin key ck previously stored when running the ActGen oracle.

$\mathcal{A}$'s view of Game 1 is identical to Game 0 because $\mathcal{CH}$ perfectly mimics the oracles that use the simulated components of $\mathrm{Adamastor}$ and those components are secure. So, we have:

$$|Pr[G_1] - Pr[G_0]| \leq negl(\lambda)$$

Lemma 21. *Assuming the anonymity against ring insider of the* RCT *component, Pr[*G_1*] is negligible in* λ *for any* $\mathrm{polynomial\ time}$ *adversary A.*

Proof. We prove this lemma by showing that if there exists a $\mathrm{polynomial\ time}$ adversary $\mathcal{A}$ has non-negligible advantage in Game 1, we can build a $\mathrm{polynomial\ time}$ adversary $\mathcal{B}$ that breaks the anonymity against ring insider of the RCT component with the same advantage since $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the queries related to RCT to its own challenger and simulating the rest of the components of $\mathrm{Adamastor}$. The guess of $\mathcal{A}$ is used by $\mathcal{B}$ in its own game and wins with the same probability as $\mathcal{A}$. □

The proof of the theorem follows directly from the lemma. □

5.4.2 Implementation and Evaluation

We implemented a prototype of Adamastor in Rust [1] with the Nero consensus algorithm chapter 4 of the document. It was bench-marked using the Microsoft Azure platform [2]. Each node corresponds to a Standard_DS1_v2 instance, the instance with the lowest specs available with an Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz processor and 3.5 GiB of memory.

The results are plotted in figure 5.1. The numbers next to each data point represent the number of members used in the ring signature of a transaction (16, 32, 64, 128, 256, 512 and 1024). The greater the ring size the stronger the level of anonymity, but also the higher the latency and the lower the throughput.

Interestingly, we can observe that the latency does not change much with the number of nodes, and the throughput even increases with the increase of the number of nodes. The first is explained by the parallelism in the transaction verification between nodes and the second by how the broadcasting is done in the test, where in each run 1000 transactions are broadcast per second divided by the number of nodes and the consensus algorithm used, which batches votes. Each transaction has the number of members of the ring signature as inputs and only one output, since our protocol does not need fees (table 5.1).

[1] `https://github.com/Fiono11/Adamastor`
[2] `https://azure.microsoft.com/en-us/free/students/`

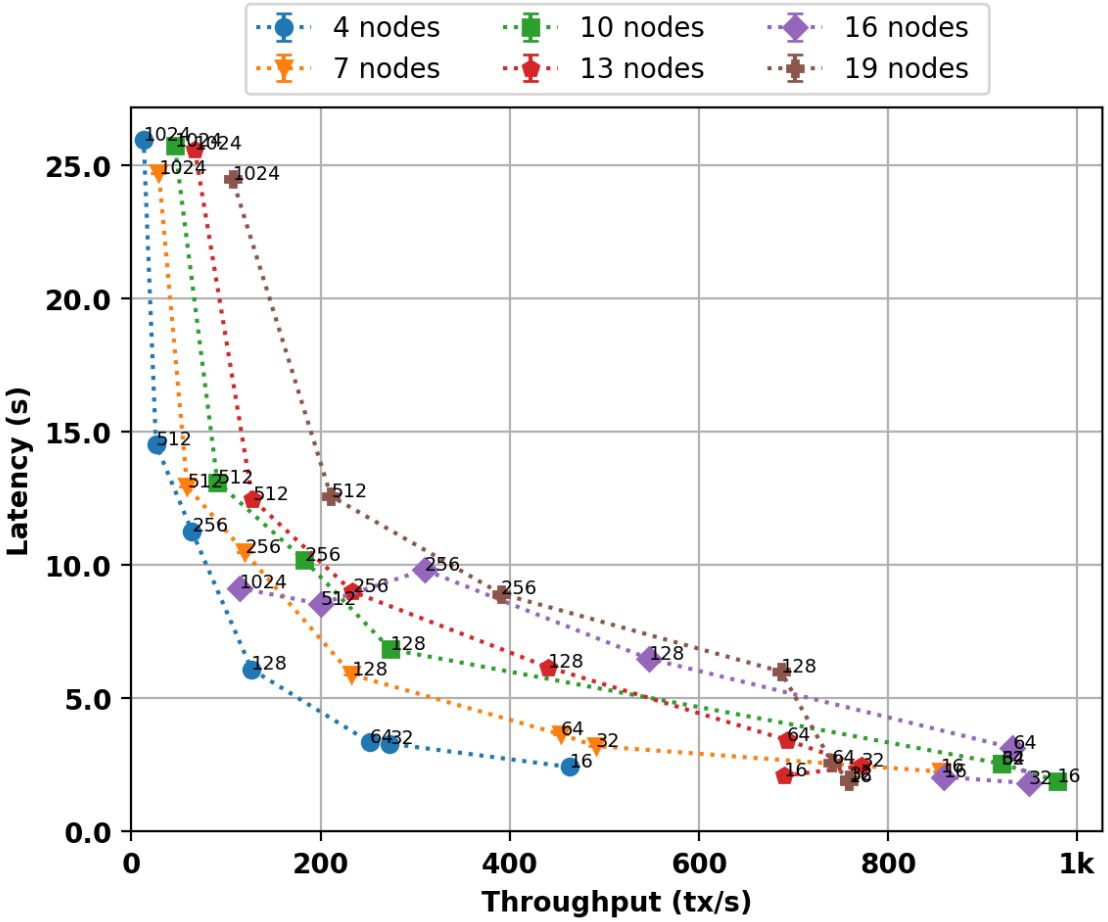

Figure 5.1: Benchmarking results of Adamastor

	Size ($\mathbb{F}$)	Size ($\mathbb{G}$)
Spend		$\Delta 3n + \Delta 3t$
Mint		$\Delta 3$
Delegate	4	3
Reveal	4	2

Table 5.1: Proof sizes of Adamastor transactions.

A spend transaction of Adamastor has an additional overhead of $3n + 3t$ and 3 elements of $\mathbb{G}$ when compared to the underlying RingCT scheme, respectively, corresponding to the ciphertext of the PKE scheme and the NIZK proof of encryption.

Delegation and PoDS transactions have proof size of $4\mathbb{F} + 3\mathbb{G}$ and $4\mathbb{F} + 2\mathbb{G}$, respectively.

5.5 Conclusion

In conclusion, the work in this section introduces Adamastor, an innovative decentralized anonymous payment system. By extending the capabilities of RingCT and incorporating a novel DSA, called SimpleDSA, we have developed a system that not only ensures robust security against homogeneity attacks and chain analysis, but also mitigates the problem of ever-increasing outputs in ring signature-based protocols. Our evaluation of Adamastor reveals remarkable improvements in latency while maintaining scalability and anonymity, setting a new standard in the realm of decentralized that use DPoS and paving the way for further advances in this fast-evolving field.

In Adamastor, delegating accounts to other nodes decreases the level of anonymity of the sender and the receiver, since the delegate knows the amount transferred in a spend transaction. However, the delegate still does not know the long term public key associated of both the sending and the receiving account. Moreover, you can always be your own delegate and maintain the level of anonymity of RingCT. Because of this the advantages of Adamastor greatly outweigh its disadvantages, namely the increase in the complexity of the protocol and its implementation.

Chapter 6

Conclusions and Future Work

In this thesis, we embarked on a comprehensive exploration of the evolving landscape of digital currencies, culminating in the conceptualization and development of a novel framework for a permissionless, decentralized digital currency based on Delegated Proof of Stake (DPoS). Our investigation traversed the multifaceted aspects of digital currencies, from their historical evolution to their technological intricacies and potential societal impacts.

Money, as a medium of exchange, has continually evolved, adopting forms that progressively address the limitations of their predecessors. Our research underscores a pivotal advancement in this evolution: the emergence of decentralized digital currencies. These currencies represent a groundbreaking shift, facilitating, for the first time, a global competition of currencies thanks to the internet. This phenomenon is not merely an economic development; it's a redefinition of financial paradigms.

Our stance is clear: a decentralized digital currency must be open and permissionless, uphold anonymity, and embody a neutral and agnostic monetary policy, devoid of fees and consensus rewards, because this leads to manipulation and centralization. If such fees or rewards are needed, they should be limited to the application layer. Such a currency should also demonstrate high performance, characterized by low latency and high throughput, and be scalable to meet global demands. In this thesis, we have endeavored to amalgamate the principles of DPoS with the desired characteristics of a robust digital currency.

In the communication layer, our research introduces innovative strategies for both vertical and horizontal scaling, addressing the critical need for scalability in digital currency networks. Relative to the first we proposed a peer management system to make the network more efficient and an inbound traffic prioritization system based on DPoS that serves as a novel alternative to fee-based models, aligning with our vision of a neutral monetary policy. We acknowledge that fees, while not fundamental at the communication level, can be incorporated at the application layer if deemed necessary. Outbound traffic is also prioritized based on the importance of the messages to reach consensus. Regarding horizontal scaling, we proposed a sharding architecture based on database sharding that uses a fallback mechanism to ensure consensus if one shard goes offline.

In the transaction layer, we have presented a general framework that makes DPoS compatible with anonymity. We have presented two ways to instantiate amount anonymity, one based on multi-party computation and other based on the Diffie-Hellman key exchange.

We have also proposed a new decoy selection algorithm for the sender anonymity, called SimpleDSA. These advancements are significant, as they uphold the crucial aspects of financial privacy in digital transactions.

The consensus layer of our framework introduces two new consensus algorithms, called Nero and Echidna, along with two distinct approaches to achieving state machine replication: leader-based and leaderless, called Sphinx and Cerebrus, respectively. These innovations are geared towards enhancing the performance of the digital currency network.

In the final chapter, we integrated the transaction and consensus layers to lay the groundwork for a permissionless decentralized network for digital currency based on DPoS called Adamastor. We have formalized this framework with security proofs and have moved towards its practical implementation.

Our research demonstrates the feasibility of developing a permissionless decentralized digital currency network based on DPoS, encompassing the characteristics we envisioned. This contribution is not only theoretical but paves the way for practical applications, offering a blueprint for future digital currency systems that prioritize openness, performance, and security.

While we have proposed novel ideas and laid a foundational framework, some of these concepts require further exploration, particularly in the context of a fully operational network. The complexity of implementing certain aspects of this thesis without a working network underscores the need for continued research and development.

The natural progression of this work is the implementation of a fully functional network that embodies the principles and innovations discussed in this thesis. Such an endeavor would not only validate the theoretical constructs but also provide invaluable insights into the practical challenges and opportunities of decentralized digital currency systems.

This thesis contributes to a transformative form of money that has the potential to change the world for the better. By exploring and developing the concept of a permissionless, decentralized digital currency based on DPoS, we offer a vision of a financial system that is more inclusive, efficient, and secure. The implications of this work extend beyond the realms of finance and technology, hinting at a future where digital currencies play a pivotal role in global economic and social structures.

Bibliography

[ABLZ18] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of bitcoin transactions. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security*, pages 541–560, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg. 82

[ADG+21] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. Leaderless consensus. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 392–402, 2021. 61, 62

[AGP+22] Tony Allen, R. G. Grant, Philip Parker, Kay Celtel, Ann Kramer, and Marcus Weeks. *Timelines of World History*. DK Publishing, New York, 2022. 2

[AHJ17] Kurt M. Alonso and Jordi Herrera-Joancomartí. Monero - privacy in the blockchain. *IACR Cryptol. ePrint Arch.*, 2018:535, 2017. Available from: `https://api.semanticscholar.org/CorpusID:32319935`. 81

[Alo18] Kurt M. Alonso. Zero to monero : First edition a. 2018. Available from: `https://api.semanticscholar.org/CorpusID:203561927`. 81

[ava18] Snowflake to avalanche : A novel metastable consensus protocol family for cryptocurrencies team rocket. 2018. Available from: `https://api.semanticscholar.org/CorpusID:198184325`. 61

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020 , Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*, page 423–443, Berlin, Heidelberg, 2020. Springer-Verlag. Available from: `https://doi.org/10.1007/978-3-030-51280-4_23`. 81

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018. 81

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. pages 947–964, 05 2020. 81

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. Available from: `http://arxiv.org/abs/1807.04938`. 60, 62

[BL20] Leemon Baird and Atul Luykx. The hashgraph protocol: Efficient asynchronous bft for high-throughput distributed ledgers. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–7, 2020. 61

[CBHG19] Guilhem Chaumont, Paul Bugnot, Zach Hildreth, and Balthazar Giraux. Dpops: Delegated proof-of-private-stake a dpos implementation under x-cash a monero based hybrid-privacy coin, July 2019. 37

[CGLR17] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *ArXiv*, abs/1702.03068, 2017. Available from: `https://api.semanticscholar.org/CorpusID:9779579`. 64

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 103–118, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. 52, 53

[CHK+18] Tai-Yuan Chen, Wei-Ning Huang, Po-Chun Kuo, Hao Chung, and Tzu-Wei Chao. DEXON: A highly scalable, decentralized dag-based consensus algorithm. *CoRR*, abs/1811.07525, 2018. Available from: `http://arxiv.org/abs/1811.07525`. 61

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I*, page 738–768, Berlin, Heidelberg, 2020. Springer-Verlag. Available from: `https://doi.org/10.1007/978-3-030-45721-1_26`. 51

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999. 60, 62

[CMTA20] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Allen Au. Pgc: Decentralized confidential payment system with auditability. In *European Symposium on Research in Computer Security*, 2020. 97, 98, 100, 101

[CNG21] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red belly: A secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 466–483, 2021. 60, 76

[CNV06] Miguel Correia, Nuno Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Comput. J.*, 49:82–96, 01 2006. 58

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Annual International Cryptology Conference*, 1992. 93

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260, 1997. 53

[CYD+20] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Verissimo. Exploring the monero peer-to-peer network. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020 , Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*, page 578–594, Berlin, Heidelberg, 2020. Springer-Verlag. Available from: https://doi.org/10.1007/978-3-030-51280-4_31. 14

[CZF+20] Bin Cao, Zhenghui Zhang, Daquan Feng, Shengli Zhang, Lei Zhang, Mugen Peng, and Yun Li. Performance analysis and comparison of pow, pos and dag based blockchains. *Digital Communications and Networks*, 6, 01 2020. 60

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 53, 83

[DKKSS22a] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 34–50, New York, NY, USA, 2022. Association for Computing Machinery. Available from: https://doi.org/10.1145/3492321.3519594. 61

[DKKSS22b] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 34–50, New York, NY, USA, 2022. Association for Computing Machinery. Available from: https://doi.org/10.1145/3492321.3519594. 61

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35, 12 1988. 58, 62

[DPS+21] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. Survey on blockchain networking: Context, state-of-the-art, challenges. *ACM Comput. Surv.*, 54(5), may 2021. Available from: https://doi.org/10.1145/3453161. 12

[DRZ18] Sisi Duan, Michael K. Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on*

Computer and Communications Security, CCS '18, page 2028–2041, New York, NY, USA, 2018. Association for Computing Machinery. Available from: https://doi.org/10.1145/3243734.3243812. 60

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12:656–666, 1983. Available from: https://api.semanticscholar.org/CorpusID:13592890. 58

[ECP21] Jean-Philippe Eisenbarth, Thibault Cholez, and Olivier Perrin. A comprehensive study of the bitcoin p2p network. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 105–112, 2021. 14

[Eic19] Barry Eichengreen. *Globalizing Capital: A History of the International Monetary System*. Princeton University Press, 3 edition, 2019. 2

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, Oct 1987. 50

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. Available from: https://doi.org/10.1145/3149.214121. 58

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. 85

[GBE+18] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 – March 2, 2018, Revised Selected Papers*, page 439–457, Berlin, Heidelberg, 2018. Springer-Verlag. Available from: https://doi.org/10.1007/978-3-662-58387-6_24. 14

[GGAG+19] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–580, 2019. 60

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017. 60

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, jun 2002. Available from: https://doi.org/10.1145/564585.564601. 32

[GOT19] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 690–719, Cham, 2019. Springer International Publishing. 82

[Gra11] David Graeber. *Debt: The First 5,000 Years*. Melville House, Brooklyn, NY, 2011. 1

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017. 81

[Hay78] Friedrich A. Hayek. *Denationalisation of Money: The Argument Refined*. The Institute of Economic Affairs, London, 1978. 3

[HKD06] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. The case for byzantine fault detection. In *Proceedings of the Second Conference on Hot Topics in System Dependability*, HotDep'06, page 5, USA, 2006. USENIX Association. 58

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, 2015. Available from: https://api.semanticscholar.org/CorpusID:1471218. 15

[HZD+19] Weifeng Hao, Jiajie Zeng, Xiaohai Dai, Jiang Xiao, Qiangsheng Hua, Hanhua Chen, Kuan-Ching Li, and Hai Jin. Blockp2p: Enabling fast blockchain broadcast with scalable peer-to-peer network topology. In *Green, Pervasive, and Cloud Computing: 14th International Conference, GPC 2019, Uberlândia, Brazil, May 26–28, 2019, Proceedings 14*, pages 223–237. Springer, 2019. 14

[ISTY19] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. Churn in the bitcoin network: Characterization and impact. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 431–439, 2019. 15

[Jiv19] Aram Jivanyan. Lelantus: Towards confidentiality and anonymity of blockchain transactions from standard assumptions. *IACR Cryptol. ePrint Arch.*, 2019:373, 2019. 81

[KAD+07] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. volume 51, pages 45–58, 01 2007. 60

[KKKNS21] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 165–175, New York, NY, USA, 2021. Association for Computing Machinery. Available from: https://doi.org/10.1145/3465084.3467905. 61

[KKKZ19a] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 157–174, 2019. 37

[KKKZ19b] Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 157–174, 2019. 82

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014. 38, 85

[KMMS03] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16–35, 2003. 58

[KMNS21] Markulf Kohlweiss, Varun Madathil, Kartik Nayak, and Alessandra Scafuro. On the anonymity guarantees of anonymous proof-of-stake protocols. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1818–1833, 2021. Available from: https://api.semanticscholar.org/CorpusID:233176420. 37

[KMS+16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. pages 839–858, 05 2016. 82

[KN12] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. Available from: https://api.semanticscholar.org/CorpusID:42319203. 82

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, Cham, 2017. Springer International Publishing. 58

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor,

Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. Available from: https://doi.org/10.1007/978-3-642-17373-8_11. 51

[LABK17] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 297–315, Cham, 2017. Springer International Publishing. 58

[Lam01] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, December 2001. Available from: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/. 60

[LD22] Kevin Lamshöft and Jana Dittmann. Covert channels in network time security. In *Proceedings of the 2022 ACM Workshop on Information Hiding and Multimedia Security*, IHamp;MMSec '22, page 69–79, New York, NY, USA, 2022. Association for Computing Machinery. Available from: https://doi.org/10.1145/3531536.3532947. 75

[LRR+19] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 31–48, New York, NY, USA, 2019. Association for Computing Machinery. Available from: https://doi.org/10.1145/3319535.3345655. 81

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, pages 382–401, July 1982. Available from: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/. 57, 58

[Maz15] David Mazières. The stellar consensus protocol: A federated model for internet-level consensus. 2015. Available from: https://api.semanticscholar.org/CorpusID:10674564. 60

[MBKM19] M. Maller, Sean Bowe, Markulf Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019. 81

[MCdS23a] Rui Morais, Paul Crocker, and Simao Melo de Sousa. Adamastor: a new low latency and scalable decentralized anonymous payment system, 2023. Available from: https://arxiv.org/abs/2011.14159. 10

[MCDS23b] Rui Pedro Bernardo Morais, Paul Andrew Crocker, and Simão Melo De Sousa. Echidna: A new consensus algorithm for efficient state machine replication. In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pages 82–88, 2023. 10

[MCL23] Rui Morais, Paul Crocker, and Valderi Leithardt. Nero: A deterministic leaderless consensus algorithm for dag-based cryptocurrencies. *Algorithms*, 16(1), 2023. Available from: https://www.mdpi.com/1999-4893/16/1/38. 9

[MCMdS20] Rui Morais, Paul Crocker, and Simão Melo de Sousa. A tool for implementing privacy in nano. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 159–163, 2020. 9

[Men05] Alfred Menezes. An introduction to pairing-based cryptography. *CONTEMPORARY MATHEMATICS*, 30:47, 2005. 50

[MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, 2013. 81, 82

[Mil06] D.L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, 2006. Available from: https://books.google.pt/books?id=pdTcJBfnbq8C. 75

[Mon20] Henrique Moniz. The istanbul bft consensus algorithm. *ArXiv*, abs/2002.03613, 2020. Available from: https://api.semanticscholar.org/CorpusID:211069668. 60, 62

[MPP+22] Sebastian Müller, Andreas Penzkofer, Nikita Polyanskii, Jonas Theis, William Sanders, and Hans Moog. Tangle 2.0 leaderless nakamoto consensus on the heaviest dag. 05 2022. 61

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018:143–163, 06 2018. 42

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., USA, 1st edition, 1996. 75

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery. Available from: https://doi.org/10.1145/2976749.2978399. 58, 60

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. Available from: http://www.bitcoin.org/bitcoin.pdf. 58

[NG20] Sarang Noether and Brandon Goodell. Triptych: logarithmic-sized linkable ring signatures with applications. *IACR Cryptol. ePrint Arch.*, 2020:18, 2020. 81

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association. 60

[OS03] Arthur O'Sullivan and Steven M. Sheffrin. *Economics: Principles in Action*. Prentice Hall, Upper Saddle River, New Jersey, 2003. 1

[PBF+18] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*, page 43–63, Berlin, Heidelberg, 2018. Springer-Verlag. Available from: https://doi.org/10.1007/978-3-662-58820-8_4. 81

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. 54, 93

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptol.*, 13(3):361–396, January 2000. Available from: https://doi.org/10.1007/s001450010003. 97

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the Association for Computing Machinery 27*, 2, April 1980. 2005 Edsger W. Dijkstra Prize in Distributed Computing. Available from: https://www.microsoft.com/en-us/research/publication/reaching-agreement-presence-faults/. 57

[Ren19] Ling Ren. Analysis of nakamoto consensus. *IACR Cryptol. ePrint Arch.*, 2019:943, 2019. Available from: https://api.semanticscholar.org/CorpusID:201659382. 58, 60

[RMSK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 345–364, Cham, 2014. Springer International Publishing. 81

[RNH12] Na Ruan, Takashi Nishide, and Yoshiaki Hori. Elliptic curve elgamal threshold-based key management scheme against compromise of distributed rsus for vanets. *Journal of Information Processing*, 20:846–853, 01 2012. 52

[Sal20] Fahad Saleh. Blockchain without Waste: Proof-of-Stake. *The Review of Financial Studies*, 34(3):1156–1190, 07 2020. Available from: https://doi.org/10.1093/rfs/hhaa075. 58

[SALY17] Shifeng Sun, Man Ho Allen Au, Joseph K. Liu, and Tsz Hon Yuen. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. *IACR Cryptol. ePrint Arch.*, 2017:921, 2017. Available from: https://api.semanticscholar.org/CorpusID:3699609. 81

[SBCL21] Adrián Silveira, Gustavo Betarte, Maximiliano Cristiá, and Carlos Luna. A formal analysis of the mimblewimble cryptocurrency protocol. *Sensors*, 21(17), 2021. Available from: https://www.mdpi.com/1424-8220/21/17/5951. 82

[SCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014. 81, 82

[Sch90a] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, dec 1990. Available from: https://doi.org/10.1145/98163.98167. 58, 75

[Sch90b] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, dec 1990. Available from: https://doi.org/10.1145/98163.98167. 58, 71

[Sch04] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 2004. 93

[Sch06] D.M. Schaps. *The Invention of Coinage in Lydia, in India, and in China*. International Economic History Association, 2006. Available from: https://books.google.pt/books?id=av-TkQEACAAJ. 1

[SDPV19] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-bft: High-throughput robust bft for decentralized networks, 2019. Available from: https://arxiv.org/abs/1906.05552. 60

[SGSKK22a] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2705–2718, New York, NY, USA, 2022. Association for Computing Machinery. Available from: https://doi.org/10.1145/3548606.3559361. 60, 62, 76

[SGSKK22b] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2705–2718, New York, NY, USA, 2022. Association for Computing Machinery. Available from: https://doi.org/10.1145/3548606.3559361. 61

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. Available from: http://doi.acm.org/10.1145/359168.359176. 50

[Ste11] David L. Stearns. *Electronic Value Exchange: Origins of the Visa Electronic Payment System*. Springer, London, 2011. 2

[SYYY21] Yuanyuan Sun, Biwei Yan, Yan Yao, and Jiguo Yu. Dt-dpos: A delegated proof of stake consensus algorithm with dynamic trust. *Procedia Computer Science*, 187:371–376, 2021. 2020 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI2020. Available from: https://www.sciencedirect.com/science/article/pii/S1877050921009236. 37

[TW05] Patrick P. Tsang and Victor K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In Robert H. Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, editors, *Information Security Practice and Experience*, pages 48–60, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 81

[VG19] Gauthier Voron and Vincent Gramoli. Dispel: Byzantine smr with distributed pipelining. *ArXiv*, abs/1912.10367, 2019. Available from: https://api.semanticscholar.org/CorpusID:209445044. 60

[vS13] Nicolas van Saberhagen. Cryptonote v 2.0. 2013. Available from: https://api.semanticscholar.org/CorpusID:2711472. 81

[WPNM23] Chenghong Wang, David Pujol, Kartik Nayak, and Ashwin Machanavajjhala. Private proof-of-stake blockchains using differentially-private stake distortion. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association. 37

[WZY+21] Taotao Wang, Chonghe Zhao, Qing Yang, Shengli Zhang, and Soung Chang Liew. Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain. *IEEE Transactions on Network Science and Engineering*, 8(3):2131–2146, 2021. 14

[YfSL+19] Tsz Hon Yuen, Shi feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. *IACR Cryptology ePrint Archive*, 2019:508, 2019. 81, 86, 87, 90, 93, 99

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 347–356, New York, NY, USA, 2019. Association for Computing Machinery. Available from: `https://doi.org/10.1145/3293611.3331591`. 60, 62