Exploring Interoperability and Scalability in DAG-based Distributed Ledger Technologies

A Case Study of Relay Implementation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Philipp Greitbauer, BSc
Matrikelnummer 01325988

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr.-Ing. Stefan Schulte
Mitwirkung: Dr. William Sanders

Wien, 29. September 2023

Philipp Greitbauer	Stefan Schulte

Informatics

Exploring Interoperability and Scalability in DAG-based Distributed Ledger Technologies

A Case Study of Relay Implementation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Philipp Greitbauer, BSc
Registration Number 01325988

to the Faculty of Informatics

at the TU Wien

Advisor: Dr.-Ing. Stefan Schulte
Assistance: Dr. William Sanders

Vienna, 29th September, 2023

Philipp Greitbauer   Stefan Schulte

Erklärung zur Verfassung der Arbeit

Philipp Greitbauer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. September 2023

Philipp Greitbauer

Acknowledgements

At this point, I would like to express my gratitude to all those who have supported me in the creation of this diploma thesis and have accompanied me throughout my studies.

First and foremost, I want to thank my advisor Dr.-Ing. Stefan Schulte for his invaluable supervision of this thesis and for providing timely and comprehensive feedback.

Furthermore, I am thankful to Dr. William Sanders for his helpful insights and for connecting me with valuable resources and experts in the field.

However, my most profound gratitude is reserved for my wife, Ursula, for her patience during this challenging time and for my son, Johannes, who gave me the strength and motivation needed to successfully conclude this thesis.

Finally, I want to acknowledge my parents, whose initial support made my academic journey possible in the first place.

Kurzfassung

In den letzten Jahren sind Kryptowährungen wie Bitcoin und die zugehörige Blockchain-Technologie zu gängigen Themen in der Mainstream-Welt geworden. Diese erhöhte Aufmerksamkeit hat bedeutende Fortschritte im Bereich der Distributed-Ledger-Technologien (DLTs) ermöglicht. Gleichzeitig hat der erhöhte Datenverkehr auch die Bedeutung von Skalierbarkeit hervorgehoben und zur Entstehung verschiedener neuer Projekte und Techniken zur Verbesserung der Skalierbarkeit geführt. Viele dieser Systeme sind inkompatibel zueinander, was die Fragmentierung von DLTs verstärkt. Daher ist neben der Skalierbarkeit auch die Interoperabilität ein aktueller Forschungsschwerpunkt. Ein vielversprechender Ansatz zur Erhöhung des Transaktionsdurchsatzes besteht darin, die übliche Blockchain-Datenstruktur durch einen gerichteten azyklischen Graphen (directed acyclic graph, DAG) zu ersetzen. Solche Lösungen bringen jedoch ihre eigenen Herausforderungen in Bezug auf Dezentralisierung sowie Sicherheit mit sich und können immer noch an Skalierbarkeitsgrenzen stoßen. Aus diesem Grund darf die Forschung zur Skalierbarkeit solcher Systeme nicht vernachlässigt werden.

Die meiste Forschung zu Blockchain-basierten Systemen ist nicht unbedingt direkt auf DAG-basierte anwendbar. Dies gilt auch für sogenannte “Relays”, die Schemata zur vertrauenswürdigen Weiterleitung von Nachrichten von einer Ledger zu einer anderen sind. Diese Interoperabilitätslösungen können die Skalierbarkeit fördern, da sie es ermöglichen, von anderen Systemen zu profitieren, die beispielsweise bestimmte Aufgaben effizienter erledigen. In jüngster Zeit hat die Nutzung von sogenannten “Zero-Knowledge Succinct Non-Interactive ARguments of Knowledge” (zk-SNARKs) um solche Übertragungen effizient und sicher abzuwickeln viel Aufmerksamkeit erregt. Mit dieser Art von Protokoll ist es möglich, beliebige Berechnungen in kompakter Form zu beweisen, ohne Einzelheiten der Operation preiszugeben.

In dieser Arbeit geben wir einen umfassenden Überblick zu Trends der Skalierbarkeit sowie Interoperabilität in DLTs und diskutieren bestehende Relay-Lösungen. Basierend darauf präsentieren wir eine Relay-Implementierung für eine DAG-basierte DLT. Die Evaluierung konzentriert sich auf die Leistung des Relays bei der Überprüfung von echten Transaktionsdaten unter verschiedenen Szenarien. Zum Abschluss dieser Arbeit diskutieren wir mehrere Ansätze zur Nutzung von zk-SNARKs, um die vorgeschlagene Lösung weiter zu verbessern, und betonen die Wichtigkeit DLTs, unter Berücksichtigung von Interoperabilität und effizienter Anwendung von zk-SNARKs zu entwickeln.

Abstract

In recent years, cryptocurrencies like Bitcoin and the associated blockchain technology have become commonplace topics in the mainstream. This surge in attention has enabled significant advancements and research within the field of distributed ledger technologies (DLTs). Simultaneously, the increased traffic has underscored the importance of scalability, leading to the emergence of various new projects and techniques aimed at improving scalability. By this process, DLTs became even more fragmented due to several incompatible systems. Therefore, besides scalability also interoperability is a current focus of research.

A promising approach to achieve higher transaction throughput is to replace the common blockchain data structure with a directed acyclic graph (DAG) to construct a DAG-based DLT. However, such solutions come with their own challenges regarding decentralization as well as security and may still encounter scalability limits. For that reason, research on scalability specific to such systems should not be neglected.

Unfortunately, most research conducted in the field of DLT focuses on blockchain systems and may not directly be applicable to the ones based on a DAG. That is also the case for work on so-called "relays", which are schemes for relaying messages from one ledger to another in a trustworthy way. These interoperability solutions can promote scalability because they allow to benefit from other systems that, for example, handle specific tasks more efficiently. Recently, the utilization of zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) as an enabler for efficient and secure cross-chain transfers has attracted much attention. With this kind of protocol, it is possible to prove arbitrary computations in a concise way without giving away details of the operation.

In this thesis, we give a comprehensive overview of scalability and interoperability trends in DLT and discuss existing relay solutions. Based on these observations we present a relay implementation for a DAG-based DLT. The evaluation focuses on the relay performance for verifying real-world transaction data under various scenarios. Concluding this work, we discuss several approaches for using zk-SNARKs to further improve the proposed solution and stress the importance of DLTs that are designed under consideration of interoperability and efficient application of zk-SNARKs.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Cryptocurrencies like Bitcoin are usually realized using blockchains as a distributed ledger to log transactions in an immutable way. Using a consensus protocol, all participants settle on a unique transaction order. The downside of this approach is the relatively low throughput, mostly measured in transactions per second (TPS), which cannot easily be scaled up. For this reason, scalability is a heavily researched topic in the field of cryptocurrencies [WSNH19].

In recent years, blockchain-based cryptocurrencies have reached the mainstream and have become a more and more popular investment option. Hence, there is a need for a solution to handle the increased traffic. If the throughput per second cannot be increased, users have to wait longer for their transactions to be confirmed. At some point, the waiting time reaches a length where the usability of the system is heavily impacted [YWY+20; ZHZB20].

To avoid this, multiple ways for scaling an existing blockchain have been proposed in recent years. The solutions reach from optimizations of algorithms, data structures, and networking [DPS+20] to sharding and sidechains [WSNH19; ZHZB20].

Sharding, a technique originally used for databases, is a way of dividing the work (here: transactions) among groups of processing nodes. Accordingly, a blockchain gets partitioned into multiple parts, called "shards", while each node only stores and handles transactions for some of these shards. This distribution of work allows handling different transactions in parallel, improving the overall transaction throughput and also reducing the communication and storage overhead for all nodes [WSNH19; YWY+20].

Another solution for scalability works by creating additional sidechains having specific properties, like a faster block time, and also providing means of transferring native assets between the sidechains and the mainchain. This approach also makes the system extensible by just adding more sidechains with different properties, avoiding the complex process of changing the implementation of the mainchain [GKO20].

Furthermore, distributed ledger technologies (DLTs) applying other data structures than blockchains exist. For instance, IOTA is based on a directed acyclic graph (DAG) called “the Tangle” [PMC+20; Pop18], leading to a better transaction throughput because it allows attaching transactions to various nodes instead of only one. While scalability approaches for blockchains are heavily researched and in practical use, so far, little work has be done in the context of sharding or sidechains for DAG-based DLTs to even further improve the performance in terms of transaction throughput [ZHZB20].

An important field of study for DLTs is the relaying of messages in general and the transfer of funds between chains. Recently, the utilization of zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) as an enabler for such cross-chain transfers has attracted much attention [GKO20]. With this kind of protocol, it is possible to prove arbitrary computations in a succinct constant size without giving away details of the operation. The resource-intensive calculation and proof generation can happen off-chain and only the more lightweight verification is done on-chain [WE20].

By combining a DAG-based DLT, with additional sharding and sidechain solutions and the use of zk-SNARKs for cross-chain transfers, a highly scalable, secure and decentralized system for multiple use cases could be built. It remains to show if state-of-the-art zk-SNARKs or similar proof systems are efficient enough to allow all users to equally participate in all network operations.

1.2 Aim of the Work

Extensive research on possible scalability and interoperability solutions for blockchains and also on zero-knowledge proof schemes with different properties exists, but there have not been many efforts to apply these methods to other DLTs. Common, ways to scale popular existing blockchain systems such as Bitcoin receive the most attention in scientific work [ZHZB20] but also novel DLTs built from scratch with both scalability and interoperability in mind are proposed [KJG+18; PMC+20]. Although replacing the blockchain with a DAG can increase the throughput, such a solution also has limitations [BKLM19; POK19]. This is why research on scalability and interoperability, as it facilitates the former, specific to such systems should not be neglected.

Therefore, the initial aim of this thesis is to give an overview of current research in the fields of sharding, sidechains, relays as well as other related scalability and interoperability solutions for DLTs. Subsequently, the gained knowledge will be used to analyze and compare the found approaches. The results can then be used to evaluate the applicability to DAG-based DLTs and to derive a design for a relay solution. Before realizing the solution design, choices for the tech stack, as well as the potential usage of zk-SNARK libraries will be assessed. Relays are a popular field of research in DLT, as they bring together the topics of interoperability and scalability. The following paragraphs detail the different aims of the work and their expected outcomes.

Scalability and Interoperability Approaches

Various techniques can be employed to reach the goal of better scalability and interoperability. This thesis focuses on relay solutions but also looks into the usage of zk-SNARKs or similar proof systems. For other aspects, like reducing the network communication overhead or optimizing consensus protocols, only an overview is given.

The aim of the thesis is to find solutions applicable to DAG-based DLTs. Much work is dedicated to blockchain technology only. Therefore, it will be investigated whether the identified approaches are suitable for that goal and how they can be used.

zk-SNARK Schemes and Implementations

Although in recent years many new zk-SNARK proof schemes were proposed, only some concrete implementations are available. Additionally to an overview of the approaches and their practical usage, an assessment of the libraries implementing them is needed. This is especially important as proof generation generally needs significant computing resources that are not available for all participants due to constrained devices, for example in the Internet of Things (IoT).

Relay Implementation

The final output of the work is a proof of concept implementation of a relay solution for a DAG-based source ledger and an evaluation of its performance using transaction data. Furthermore, an analysis of selected security aspects and a detailed perspective for future enhancements will be given.

1.3 Methodology and Approach

The methodological approach can be grouped into four steps. Due to the fact that the topics of this thesis are still undergoing heavy research and development, novel approaches are proposed regularly. Analyzing these cutting-edge methods and their implementations is a highly exploratory work. In consequence, the listed phases are not followed in a strictly linear manner. Rather, a partly iterative process is necessary.

Literature Review

First, a literature review on the research fields of DLT scalability and interoperability with a focus on relays needs to be conducted to gain deeper knowledge about current problems and limitations. Additionally, recent advancements in (DAG-based) DLT and also in the area of zero-knowledge proofs have to be evaluated to identify possible approaches to be used as parts of the relay implementation case study.

Requirements Analysis and Solution Design

Before working on the implementation, the requirements need to be clearly defined. Based on these, a design is derived and the proposed solution is explained in detail. This part also contains reasoning about exchangeable components and the potential use of zk-SNARK proof systems.

Implementation

After analyzing existing approaches and stating the requirements, suitable options need to be chosen to fulfill them with the developed proof of concept. On that account, care must be taken when deciding on options such as the programming language to use. The selection will be limited by the support of the targeted DLT and the availability of zk-SNARK libraries, as these are the most important and complex parts of the realization. Porting such codebases is not feasible within the scope of this work.

Evaluation

Finally, the created relay implementation and possibly its different options are evaluated under various scenarios based on a suitable metric such as its "gas" costs. For zk-SNARKs, CPU or RAM resource usage, as well as the computational time for proof creation and verification, are the most important factors. To get meaningful results, real-world transaction data of a DAG-based DLT, for instance, IOTA, is used as input.

1.4 Structure

After the previous introduction, the remaining thesis is structured into the following key chapters that collectively provide a comprehensive exploration of current trends in DLT and the development of a relay scheme for DAG-based source ledgers.

In Chapter 2, the foundation is laid with an extensive overview of DLT, covering both blockchain and DAG-based cryptocurrencies. Concepts like sharding and zero-knowledge proofs, particularly zk-SNARKs, are thoroughly examined. Chapter 3 investigates existing research in the field. It addresses the topics of scalability, interoperability, and their challenges. This review of related work sets the stage for the thesis's original research.

Chapter 4 presents the requirements and design approach of a relay scheme for DAG-based source ledgers, on the concrete example of IOTA. While drawing inspiration from blockchain-based relay solutions, it highlights the specific design considerations taken for this work. Chapter 5 follows with a detailed exploration of the implementation aspects of the relay solution described in Chapter 4. It discusses the technology stack, the relay contract functionality as well as the testing approach. Chapter 6 evaluates the solution, aligning it with the design criteria specified in Chapter 4. Furthermore, data collection and analysis, benchmarking, and a security analysis are presented in this chapter.

Finally, Chapter 7 serves as the conclusion, summarizing key findings and insights gained throughout the thesis. It also looks ahead to potential research directions following this work and also within the broader field of research, providing a fitting closure to this comprehensive exploration of DLTs and relay schemes.

CHAPTER 2

Background

Chapter 2 provides a comprehensive overview of DLT. It covers blockchain-based, as well as DAG-based cryptocurrencies, and the concept of sharding. The chapter also explores zero-knowledge proofs, focusing on zk-SNARKs and their applications. A summary concludes the chapter, highlighting the key points discussed and their significance in the context of DLTs.

2.1 Blockchain-based Cryptocurrencies

Since the Bitcoin whitepaper was published by Nakamoto [Nak08] in 2008, DLTs have evolved and become more and more accepted by the public. This evolution of blockchains can roughly be divided into three generations, explained in the following paragraphs [Do23; KS22; Ten]. The actual classification of generations in literature varies slightly depending on the source and its intended audience and goal.

Bitcoin and its direct derivatives can be labeled as first-generation blockchains (or DLTs). They were proposed with the goal of building a decentralized *peer-to-peer electronic cash system* [Nak08] based on cryptographic functions and financial incentives, later commonly referred to as "cryptocurrencies".

With Ethereum, initially described by Buterin [But21b] and later specified in a more formal way by Wood [Woo21a], the second generation of blockchains began. Note that both of these papers are still updated. The initial versions were published in 2013 and 2014. Smart contracts, the major feature introduced by Ethereum, allow to create applications that are directly executed in the network and can control digital assets, based on the code written in a Turing-complete programming language. For this reason, second-generation blockchains are not just cryptocurrencies but rather platforms that other projects can build upon. Developers can implement various decentralized apps (dapps), for instance, financial services, games, and supply chain management. In Bitcoin, the available balance of a user is the total value of so-called unspent transaction outputs (UTXOs), meaning coins of a transaction not yet used in another one, the user can prove ownership for. Ethereum instead opted to store the balance of each account in the global state. This approach is named “account model” and was chosen because of a more simple handling in smart contracts and storage space savings [Eth20].

The new possibilities also meant a steady increase of transactions waiting to be executed, resulting in much research on how to scale the efficiency of the system. Third-generation blockchains are designed with inherent scalability solutions to tackle this problem. Second-generation projects also work on integrating scalability solutions. For instance, Ethereum transitioned to a PoS consensus and prepared for future scaling enhancements through an update called “The Merge” that occurred in September 2022 [Eth23a].

Because there are already a huge amount of DLT projects covering several use cases and one general-purpose blockchain cannot fulfill all of them with the same efficiency, interoperability between different systems is of utmost importance. This is the second topic third-generation blockchains are working on besides scalability. Well-known projects include Cosmos [KB19] and Polkadot [BCC+20; Woo16]. With their SDKs, both projects facilitate to build “application-specific blockchains”. Instead of executing a decentralized application on top of a general-purpose virtual machine, like the EVM, it directly is run as part of the chain logic. In the past, creating such a customized blockchain required forking an existing DLT codebase, such as Bitcoin. Adaptions beyond simple configuration changes were quite challenging as it was not intended for it.

Building an application as an application-specific blockchain has multiple benefits [ICF20; Mar19]:

- Flexibility: Smart contracts usually require specific programming languages and are therefore limited to a few choices. Compared to that developing app-specific blockchains is possible with multiple general-purpose languages, although there still is the limitation of SDK availability. With the rise of WebAssembly VMs allowing many different programming languages for smart contract implementations, language choice on its own is not an advantage of application-specific blockchains anymore. Even Cosmos has a module for adding Wasm smart contract capabilities to a blockchain built with its SDK[1]. Nevertheless, a customized blockchain allows you to weigh trade-offs such as validator count, throughput, the balance of safety and availability, and the account model to make a decision that fits your use case best. Another benefit over smart contracts is the possibility to automatically trigger code execution, e.g., at the beginning or end of each block.

- Performance: Although the performance of VMs regularly improves and often used functionality is integrated in an efficient way, native execution in general will always be ahead in terms of performance. Furthermore, an app-specific blockchain

[1]CosmWasm. https://docs.cosmos.network/v0.46/CosmWasm/, Last accessed: 2023-03-15

allows one to choose a consensus engine and avoids competition over storage or computation resources between applications.

- Sovereignty: With an app-specific blockchain, the developer has full sovereignty over the application and is not dependent on the governance of the network. On the one hand, this allows to easily fix occurring issues but on the other hand reduces trust in the network.
- Security: Without a VM, the attack surface is vastly reduced as there no longer is the need to reason about its mechanisms. In addition to, more or less, free choice of programming language also any available cryptographic library can be utilized instead of being dependent on the limited functionality available in smart contracts.

Of course, also first-gen blockchains worked on improving their efficiency by various means. Some of them are explained in Section 3.1. Additionally, there are projects that do not directly fit into the described generations. For instance, IOTA (further described in Section 3.5.1) does not use a blockchain and also does not support smart contracts. Although the blockchain data structure has found a novel use case with cryptocurrencies, it has limitations, such as not being suitable for parallel attachments of new blocks, causing slow confirmation and low throughput [WYCX20].

Blockchains save state changes in the system by storing transactions in a block. These blocks linearly build upon their predecessors, forming an immutable chain. Therefore, to not cause conflicts, only one block can be added at a time. The performance bottleneck is caused by that in combination with the consensus mechanism determining which participant is allowed to add the next block. Because the amount of transactions fitting into one is limited, the others are pooled until the following round. The race to be the miner to attach the next block can lead to conflicts and wasted computations [IOT22; WYCX20].

For the given reasons, research on replacing or at least complementing blockchains with other data structures started early. A popular concept is that of using a tree-like representation. First ideas came up to accelerate Bitcoin by structuring blocks in trees [Tod14]. The next step was to avoid blocks altogether and use transactions to directly confirm multiple others, constructing a DAG by that process.

2.2 DAG-based Cryptocurrencies

In contrast to most DLTs, which are based on a blockchain, systems using a DAG to attach transactions come with the intrinsic advantage of fast confirmation and high scalability. Blockchain systems rely on keeping the transactions/blocks linearly in a single chain. Therefore, there is always only one point (not counting temporary forks) to attach a succeeding block, causing competition over it. This ultimately leads to slow confirmation. On the other hand, structuring transactions in a DAG provides multiple attachment points enabling parallel processing of transactions [WYCX20].

This distributed structure also allows for a more efficient utilization of network resources, because transactions can be processed concurrently by different nodes in the network. In general, performance is improved by reducing the communication, computation, as well as storage overhead. Additionally, the lack of a single, central chain in a DAG-based DLT theoretically eliminates the need for nodes to reach consensus on a linear global ordering of transactions, resulting in improved scalability [WYCX20].

However, this shift in data model and transaction handling also calls for new consensus algorithms and adjusted incentives to retain decentralization and security. Traditional blockchain systems often rely on proof-of-work (PoW) or more recently on proof-of-stake-based (PoS) procedures to validate and order transactions.

In PoW, participants (miners) have to solve complex mathematical puzzles in order for them to be allowed to validate transactions and add them to the blockchain. These puzzles, for example providing an input to a hash function that leads to an output with specific properties, require significant computational power and energy.

PoS is an alternative in which validators are chosen to create new blocks and confirm transactions based on the amount of cryptocurrency they hold and are willing to *stake* as collateral, reducing the need for energy-intensive computations seen in PoW.

Another type is Byzantine Fault Tolerance (BFT) based consensus, where nodes vote on the validity of transactions or proposed changes to the system. Through multiple rounds of communication and voting, nodes must reach a threshold of agreement, typically two-thirds, to ensure that only valid transactions are included in the system's ledger, even in the presence of faulty or malicious nodes.

In DAG-based DLTs, novel consensus algorithms leverage concepts like voting, gossip protocols, or random walks with cumulative weights to ensure the integrity of the ledger and prevent double-spending [WYCX20].

Furthermore, the use of a DAG in DLTs introduces some unique challenges. One challenge is the potential for conflicts or conflicting transactions within the graph, which requires additional measures to resolve. Moreover, the security of a DAG-based DLT relies heavily on the honesty and reliability of the participants in the network. Since transactions can be confirmed by different nodes independently, malicious actors could potentially create multiple conflicting transactions or attempt to gain advantages in the network by other means. Some attack scenarios known from classic blockchains can also be applied to DAG-based systems. Additionally, various novel threats have surfaced due to the inherently different approaches taken in these DLTs. Therefore, robust mechanisms for transaction validation, reputation systems, and anti-spam measures are crucial in ensuring the integrity and security of the ledger [AVD20; CKN+22].

DAG-based systems, compared to blockchains, still have not reached widespread commercial acceptance due to their inconsistent designs, lack of standards, less certain security, mixed performance, and implementations not matching the initial vision. For example, IOTA employs a centralized auxiliary construct instead of relying on consensus based on the heaviest DAG, such as the probabilistic and leaderless protocols proposed by Popov and Buchanan [PB21] or Müller et al. [MPP+22]. In general, DAGs may enhance scalability and performance but often lead to sacrifices in terms of consistency, finality, or decentralization [WYCX20].

2.3 Sharding

Sharding techniques originated from horizontal database partitioning schemes that separate very large databases into many smaller parts (shards). This increases performance and makes database management easier [WSNH19].

When transactions access multiple shards, coordination protocols are required to ensure properties such as atomicity and isolation [DDL+19]. In their work, Dang et al. define the term sharding as the combination of replication and partitioning. By replicating each partition, distributed database systems can achieve fault tolerance and scalability. To keep the content consistent, consensus protocols are needed, which could be combined with transaction management in a well-thought-out way to reach better overall performance, compared to just layering the two systems.

The goal of sharding in general is to scale out communication, storage and computation. To scale out (also referred to as horizontal scaling) means to grow the respective capacity with the number of nodes [YWY+20].

Yu et al. describe two fundamental issues that need to be addressed by any sharding mechanism.

2.3.1 Intra-Consensus Safety

Sharding increases the throughput but also paves the way for different types of attacks, especially when naive sharding techniques are used. For instance, the 1% attack, named after the example that the whole network consists of 100 shards and one miner has 1% of the total amount of mining power. This is by far not enough to directly attack the whole network because usually more than 50% would be needed. An honest actor striving to maximize his rewards would distribute the mining power over multiple shards, but if he chooses to just concentrate on one he can completely control that shard. When more shards are added, the issue gets worse as the needed mining power to attack one decreases further. Therefore, especially a PoW-based consensus protocol under this assumption is insecure. BFT-based consensus algorithms as an alternative can solve the issue. The most important requirement for that type of algorithm is a source of secure randomness that is unpredictable and unbiasable. It is then used to allocate nodes to shards both initially and for each reshuffling phase, deciding shard leaders and also choosing where cross-shard transactions should be broadcast to [YWY+20].

2.3.2 Cross-Shard Atomicity

Although distributing transactions over multiple shards helps with the throughput also the overhead and latency of communication between shards have to be considered. With an increasing number of shards, the probability of cross-shard transactions approaches 100%. Therefore, the communication overhead could outweigh the initial advantage of sharding resulting in an inefficient technique that does not reach the desired scalability.

Atomicity guarantees that a transaction is successfully executed across all different shards and thereby ensures consistency and prevents partial execution or inconsistent states in the distributed ledger. It needs to be fulfilled not only for simple transactions, for instance, value transfers but also for more complex statements, such as the execution of smart contracts.

To achieve cross-shard atomicity, coordination mechanisms and protocols must be established. These mechanisms enable the coordination and synchronization of transactions across different shards, ensuring that they are executed atomically. Yu et al. discuss various approaches, based on lock/unlock schemes as well as asynchronous lock-free solutions [YWY+20].

2.4 Zero-Knowledge Proofs

The first notion and definition of zero-knowledge proofs (ZKPs) was given more than three decades ago by Goldwasser, Micali, and Rackoff [GMR89]. In their work, they introduce interactive proof systems which are needed so that a prover can convince a verifier that he knows the secret, without giving it (or any other knowledge) away, by repeatedly exchanging information. They also give credit to Babai and Szemeredi who invented the so-called Arthur-Merlin proof systems which are similar to interactive proof systems. As an application of a zero-knowledge interactive proof system Goldwasser et al. describe the usage for cryptographic protocols in which it could securely replace a trusted middleman.

Further work in this field was done by Goldreich, Micali, and Wigderson, showing a ZKP for the graph coloring problem under the assumption of secure encryption schemes (and the existence of one-way functions) [GMW86; GMW91]. Since every problem in NP can be reduced to this NP-complete one, they proved that ZKPs can be constructed for every problem in NP.

Later Quisquater et al. published the famous, illustrative example known as the "Ali Baba cave" [QQQ+90] still used nowadays to explain ZKPs in a comprehensible way. This special cave has one entry and leads to a circular path that is blocked by a gate midway through. The gate only opens for someone knowing the secret words. Now imagine two persons, Peggy (prover) and Victor (verifier). Peggy wants to prove to Victor that she can pass the door, without revealing how. This can be done with an interactive zero-knowledge protocol as follows.

First Peggy enters the cave and chooses one direction while Victor waits outside so he cannot observe which way Peggy went. Next Victor, waiting at the fork, tosses a coin and depending on the outcome tells Peggy to come back via the left or right path. If she truly knows the secret to open the door she can fulfill that request no matter the given direction. Of course, if Peggy does not know the secret, there is a 50% chance for her to choose the correct side in advance. Hence the described procedure needs to be repeated n times until the probability $1/2^n$ of always guessing correctly is so low that Victor is convinced that Peggy really knows the secret. Otherwise, if she shows up on the wrong side once, Victor immediately knows that Peggy is dishonest and has no knowledge of the secret.

The property of being able to notice dishonest behavior in such a proof system with a high probability is called *soundness*. Another property of interactive (zero-knowledge) proofs is *completeness*.

Together they can be informally described like this [GMW91; Gol01]:

1. Completeness: If the prover knows the secret (statement is true) the verifier can be convinced of this fact with high probability.

2. Soundness: If the prover does not know the secret (statement is false) the verifier can only be fooled into believing otherwise with negligible probability.

3. Zero-knowledge: The proof does not yield knowledge beyond the fact that the prover knows the secret (i.e., just the validity of the statement).

It is important to differentiate between *knowledge* and *information*. Goldreich describes this in *Foundations of Cryptography* [Gol01] to motivate the definition of zero-knowledge and explain what a gain in knowledge is. In the sense of information theory, there is no difference between asking, for instance, about different properties of a publicly known large graph. Knowledge however is related to computational difficulty. Therefore someone does not gain knowledge by asking for a fact that could have been efficiently determined by oneself, e.g., if the graph is Eulerian. One gains knowledge if after learning the answer something can be easily computed, which was not possible efficiently before. In the context of the graph example, this could be the knowledge if the graph contains a Hamilton path, which is an NP-complete problem and therefore cannot be computed efficiently.

According to Blum et al. [BDMP91], interactive ZKPs consist of the following building blocks:

(1) Interaction: Prover and verifier communicate with each other recurringly.

(2) Hidden Randomization: The verifier's source of randomization (e.g., the coin toss) is hidden from the prover.

(3) Computational Difficulty: The proof embeds a computational difficulty of some problem.

In a quest to cut down these requirements, Blum et al. investigated the possibility to implement ZKPs without recurrent interaction between prover and verifier. Getting rid of this communication overhead would considerably enhance the practical applicability of ZKPs.

Considering that uni-directional ZKPs without any shared information are only existing for trivial statements, the new approach uses *shared randomness* as a solution. This means that prover and verifier know the same, short, randomized string, commonly named “reference string”. Therefore the building block (2) is obsolete, as the verifier does not need to do any coin tosses. Additionally, as now only one single message is sent from prover to verifier (1) is not needed anymore leading to the class of non-interactive zero-knowledge proofs (NIZKs) [Gol04].

In the work of Blum et al., a new reference string is needed for each proof and it cannot be reused without implications on the zero-knowledge property. As a general open problem, they mention finding more efficient proof systems and even more important, reducing the complexity assumptions needed. The question whether many provers could share the same reference string without any drawbacks regarding the properties completeness, soundness, or zero-knowledge was in succession positively answered in “Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String” by Feige, Lapidot, and Shamir [FLS90].

2.4.1 zk-SNARKs

More research in the field of NIZKs led to more practicable and efficient proofs with desirable properties, such as succinctness. Bitansky et al. [BCC+17; BCCT12] first coined the term *zero-knowledge Succinct Non-Interactive ARguments of Knowledge* (zk-SNARKs) for describing such enhanced proof systems. Gennaro et al. also worked on succinct NIZKs by constructing a SNARK and making it zero-knowledge [GGPR13]. Subsequently, efforts were made to create concrete, universal implementations solving previous limitations and helping to bring zk-SNARKs into practical use [BCTV14; PGHR13]. Details on selected state-of-the-art research and open challenges can be found in Section 3.4.

The properties building the acronym SNARK are described in the following [But21a; Rei16]:

- Succinct: The length (or size) of the proof sent to the verifier and the time needed for the verification is small in relation to the original statement’s size and computation time. Therefore, the proof can be verified with considerably less effort than it takes to build it.

  In other words: the size of the proof, as well as the time required for its verification, grows substantially slower than the computation to be verified.

- Non-interactive: Instead of recurring interactions, only a trusted setup and a single message from the verifier to the prover are needed. Additionally, many SNARKs satisfy the *public verifier* property, allowing them to be confirmed by multiple verifiers without further messages. This is especially helpful in the context of DLTs.

- ARguments: Also known as computational soundness, the verifier is only protected against maliciously crafted proofs under the assumption of a computationally bounded prover. An adversary with unlimited/enough computation power could craft proofs that would be accepted by the verifier even if they include wrong statements. However, with such computation resources also public-key encryption could be broken.

- of Knowledge: A prover can only construct a proof when knowing a witness for it. The preimage x of a hash function h(x) is an example of a witness. Hence, the verifier does not only know that a witness exists but also that the prover really knows one.

Unfortunately, most SNARK schemes require a one-time trusted setup to produce reference strings [GKM+18; Set19].

There are different types of reference strings used [GKM+18; ZKP19]:

- "Common Reference String" (CRS): The umbrella term for both types (URS and SRS). The string, composed of proving and verification parameters (also referred to as keys), is known to the prover and verifier.

- "Uniform Random String" (URS): In earlier literature, it was also called "Common Random String" but due to the same acronym as the Common Reference String this name is avoided nowadays. URS is a special case of CRS. The reference string in this case is sampled from a uniform space without involving secrets. Such a setup, where the prover and verifier access the URS, obtained from a common source of randomness, is called *transparent*.

- "Structured Reference String" (SRS): An SRS is created based on a complex distribution with a sampling algorithm using internal randomness which must be kept secret. If this data is revealed, the creation of proofs for false statements would be possible. For that reason, a trusted setup is needed for SRS creation. Groth et al. use the term *structured CRS* [GKM+18].

In environments without trusted authorities, for instance, in DLTs, such a setup is often infeasible or at least tied to the huge effort of a secure multi-party computation (MPC) ceremony[2]. The result of an MPC is secure as long as one of the participants is honest. Some MPC protocols result in a CRS for a fixed circuit. In case of an upgrade, for new

[2]Example of a complex MPC procedure [Zcab]

functionality or better performance, and therefore a changed circuit, a whole new setup ceremony is needed. A circuit is a representation of a computation consisting of inputs, gates, wires and outputs. The Rank-1 Constraint System (R1CS) is a concrete way to define a circuit using mathematical equations and inequalities describing how the input is transformed to the output [Mal19].

Although this requirement hinders the widespread practical usage of zk-SNARKs, such an upgrade procedure was performed by Zcash [Zcab]. The initial ceremony happened in 2016 and required pre-selected participants who needed to be available for the entire process. If one participant would have aborted, the whole procedure would have failed. Since then MCP protocols have improved and scale better with respect to the number of participants as they now can leave the protocol immediately after their work is done.

Promising solutions to avoid additional MPCs come with common reference strings that are *universal* or *updatable* [CHM+20; GKM+18; MBKM19].

A universal SRS does not depend on specific relations and therefore is usable for all relations (of a bounded size), while an updatable one can be changed to cope with the new requirements. Updates require a proof of correctness and can be done by a single user or multiple ones in succession. The updated SRS can be trusted if this proof can be verified successfully under the additional condition that either the old string was trustable or the updater was honest.

Universal SRS can be used for any circuit but in turn, they result in lower performance compared to using a fixed SRS. To mitigate this drawback, more efficient, specialized SRS for concrete problems can be derived. Such derivation algorithms only need public information as input and therefore can be used whenever needed. Besides resulting in a smaller reference string for a concrete circuit, the time complexity of verification and proving can also be reduced. For instance, in [GKM+18], a universal quadratic size SRS is specialized to a linear-size SRS with linear-time prover computation. This procedure however requires an expensive computation on its own. To set the size of the SRS from Groth et al. [GKM+18] in a practical context: Zcash has a circuit with 2^{17} multiplication gates leading to an SRS size in the order of terabytes. Some SNARKs like for instance *Sonic* [Mal19] do not need to be specialized to be linear in size.

Furthermore, in recent work, Setty proposed a *transparent* SNARK that does not require a trusted setup [Set19].

Bulletproofs, an alternative proving system to zk-SNARKs, do not require a trusted setup and their proof size grows logarithmically. On the other hand, the verification time scales linearly, even when applying additional techniques like batching to prove multiple statements as a whole. Therefore Bulletproofs are only suitable for relatively simple relations [MBKM19].

Zero-knowledge Scalable Transparent ARguments of Knowledge (zk-STARKs), first constructed by Ben-Sasson et al. [BBHR18], have similar properties to Bulletproofs. They are transparent, meaning they do not need a trusted external setup phase and share the proof size complexity as well. The verification time only increases logarithmically and therefore is shorter than that of Bulletproofs. When compared, zk-STARKs only are at an advantage when the setup phase of zk-SNARKs is included in the measurement. If the same computation with different inputs is anyway repeated many times or the resources needed for the pre-processing do not hinder the intended use case, zk-SNARKs are beneficial due to the constant verification time and proof size. As Ben-Sasson et al. state, zk-STARK proofs are about 1000 times longer than those of zk-SNARKs. In contrast to the zk-STARK construction in [BBHR18] many zk-SNARKs in practical use are based on the discrete logarithm assumption and therefore are not quantum-resistant. However new post-quantum secure zk-SNARK schemes have been proposed recently [NYI+20].

2.4.2 Applications of ZKPs

This subsection gives an overview to the power and versatile use cases of ZKPs. Besides their usage in cryptocurrencies, there are also more practical and easier graspable scenarios in the insurance sector.

DLTs

While zk-SNARKs can be used in many ways, one of the first practical applications of the technique was *Zcash*. Zcash is a blockchain-based cryptocurrency built on the original work done for the Zerocash protocol [BCG+14; BCT20]. It allows for transparent transactions similar to Bitcoin as well as shielded ones where the addresses and the transaction amount are kept private by using zk-SNARKs. For reasons of compliance or auditory needs, the owner of a shielded address can choose to reveal transaction details to trusted third parties with the help of viewing keys[3] [Zcaa].

Monero, another privacy-focused cryptocurrency, employs Bulletproofs for range proofs (which are simple statements [BNTT20]) to secure the amount sent in transactions. By replacing the previous type of range proofs[4] with Bulletproofs they could notably reduce the transaction size and therefore costs [Mon].

Besides the privacy-focused aspect, zk-SNARKs can also be used to realize scalability features for DLTs. For instance, instead of publishing data for each transfer, the data from these transfers is first collected and then processed in bulk at once, which is more efficient in terms of computation and storage resources, resulting in significantly lower fees [Ethb; TSH22].

Data Collaboration

An interesting use case for ZKPs is using them for building a privacy-protected and regulatory-compliant data collaboration platform. This is for instance explored in the insurance industry [DeS21]. To not give their competitors an advantage and protect

[3]https://electriccoin.co/blog/explaining-viewing-keys/, Last accessed: 2023-03-15
[4]Schnorr and later Borromean signatures

customer data, insurers usually use their data independently. However, by exchanging data, some fraudulent activities may be detectable while this is impossible in a siloed environment.

A simple example is identifying cases where the same invoice has been handed in multiple times for a refund of expenses. Detecting such behavior in an automated way while not revealing sensitive information is a benefit to all stakeholders. Insights from internal data can be monetized, risks mitigated and losses prevented. Additionally, the amount of resource-consuming investigations needed to uncover fraud cases is reduced. Analyzing if companies have mutual customers, without compromising user privacy, allows them to develop better targeted marketing campaigns.

Altogether, zero-knowledge data collaboration in the insurance industry not only greatly improves efficiency by various means, especially at fraud detection and business growth, but also enables whole new possibilities such as data monetization.

Verifiable Computation

In general, zk-SNARKs are just a way of achieving (zero-knowledge) verifiable computation, where two parties collaborate on the computation of a function. When talking about ZKPs, these parties are usually called verifier and prover, but in the context of verifiable computation, the terms client and worker are used. The correctness of the result returned by the worker can then be verified by the client [PGHR13]. Therefore, this topic covers nearly all major use cases. It can be used for different types of outsourcing, for instance realizing blockchain-based computation offloading [KSF+21] or doing computations, e.g., SQL queries, over outsourced data [ZGK+17].

Furthermore, verifiable computations can solve the trust issues regarding the integrity of federated learning results. Heiss et al. [HGT+22] propose a concrete solution utilizing zk-SNARKs for verifiable off-chain computation together with DLT to build a decentralized and trustworthy federated learning system.

Anonymous credentials are another practical example made possible by verifiable computation. Instead of revealing their whole credentials, users can selectively disclose identity attributes, or prove statements about them, while protecting other privacy-sensitive ones [CG12; SKSB19].

Besides the proofs for cryptography-focused statements used in context of DLTs, where proof size is crucial, there is emerging interest in highly complex statements of other nature. In particular DARPA's SIEVE program researches statements about software to prove vulnerabilities in code without revealing the actual exploit and about sociotechnical interactions, for instance, if a computation is compliant to data protection regulations [Bar; BNTT20; DAR21]. Additionally, the program works on improving the efficiency of post-quantum zk-SNARK schemes.

2.4.3 Standardization

Because the interest in ZKP technologies is growing fast and novel work with slightly different notions is published regularly, it is crucial to settle on a standard in this field. For this reason "ZKProof Standards"[5] has been formed in 2018[6]. ZKProof is an initiative consisting of renowned researchers, already mentioned in this work such as Goldwasser, Chiesa, Groth, and well-known industry partners, for instance, Microsoft, Google, and the Ethereum as well as the Zcash foundation. Their aim is to mainstream and standardize ZKP cryptography in a community-driven way. A standard for this advancing technology fosters trust and enables faster adoption in the industry. Workshops held annually are used to discuss the latest proposals and projects. The outcome of these efforts is documented in the ZKProof Community Reference [ZKP19] which should become the preferred source of trusted specifications and definitions for anyone seeking to implement ZKPs. To facilitate interoperability and flexibility the work gives guidelines on the implementation as well as the representation of ZKPs. It also includes best practices regarding the benchmarking of proof systems with the aim of fair and unbiased comparisons. Additionally, the protocol design steps for three ZKP use cases (identity management, asset transfer, and regulation compliance) are presented. Besides these practical aspects also the theory and security assumptions of ZKP systems are covered.

2.4.4 Summary

In Chapter 2, we provided a comprehensive background to establish the necessary foundation for our research. We explored various key topics, including blockchain-based as well as DAG-based DLTs, sharding, and ZKPs.

We summarized the evolution of DLT in Section 2.1 by describing the issues and motivations leading to the creation of the respective next generation, which itself opened up other areas that have to be improved. Due to the fast innovation in the field, even quite novel solutions appear to always be one step behind and sometimes fail to deliver their promises in time to stay relevant.

DAG-based DLTs (Section 2.2) in theory allow for better scalability by replacing the blockchain with a graph but have drawbacks in other areas. Furthermore compared to blockchain-based solutions they are in the minority and therefore get less attention in industry and research. In addition, because of their wholly different approach, partly even between DAG-based solutions, they have higher effort to incorporate advancements in the field.

Section 2.3 describes the concept of sharding, which originated from the field of databases, as a combination of replication and partitioning. The goal of sharding is to horizontally scale a whole system while considering the safety of each part and the atomicity of transactions across.

[5]https://zkproof.org/, Last accessed: 2023-03-15
[6]Steering Committee has been formed 2018-01-31
https://zkpstandard.github.io/zkproof.github.io/index.html, Last accessed: 2023-03-15

Interestingly, the original research for ZKPs goes back more than three decades. Due to recent advancements, this field of research gained a lot of traction again. ZKPs in their various manifestations can be used in many scenarios. They allow the construction of proofs for statements over a secret that can be verified without giving the secret away. For example, we could prove the possession of the pre-image of a hash without revealing it. This characteristic makes them a valuable construct for privacy-related goals. A family of ZKPs called zk-SNARKs has the property of succinct proofs, meaning that the proof is small and fast to verify compared with the statements it proves. Therefore it is primed for usage in DLTs, where it is beneficial to do as much work as possible off-chain and have a low storage and computation footprint on-chain. More details on the evolution, the properties and uses of ZKPs are given in Section 2.4.

Currently, projects with a focus on interoperability as well as application-specific blockchains, and the utilization of ZKPs seem to be the most promising approach.

The insights gained from this chapter form the basis for our exploration of related work in the subsequent chapter, where we examine the existing research and practical developments of the described concepts.

CHAPTER 3

Related Work

This chapter provides an overview of the existing research in the field of distributed ledger technology. We explore the topics of scalability, sharding in DLTs, distributed ledger interoperability, zk-SNARKs, and DAG-based DLTs. Each section discusses the key concepts, notable advancements, and challenges associated with these areas. This review of related work serves as a foundation for our own research.

3.1 Scalability

In general, cryptocurrencies face the “Scalability Trilemma” (or “Blockchain Trilemma”), stating that a system cannot be perfectly scalable, decentralized, and secure at the same time [CZZ20; KJG+18; ZHZB20]. Figure 3.1 illustrates that concept with a triangle, where the corners represent the three conflicting properties. Commonly, DLT projects focus on one side and therefore two properties, but they have to make compromises on the third one. For instance, traditional blockchains, such as Bitcoin or Ethereum, are located on the bottom edge, as they are decentralized and secure but lack scalability.

According to Buterin [But21d], simple techniques only allow for two out of those three desired properties. However, he describes sharding as a complex solution to realize a scalable, decentralized, and secure DLT. Zhou et al. [ZHZB20] state that, although these three properties will not be perfectly achieved together, a future DLT has to balance them well to be suitable for mainstream adoption. Therefore the trilemma can be relaxed to say that it just is very hard for a system to be reasonably scalable, decentralized, and secure at the same time, instead of not being possible at all.

In their work, Zhou et al. [ZHZB20] also categorize several scalability solutions and assign them to the respective layer they are applied to. Layer 1 describes solutions that are employed on-chain and therefore need changes in the system itself. A simple example are changes related to block data, for instance, expanding the block size to be able to fit

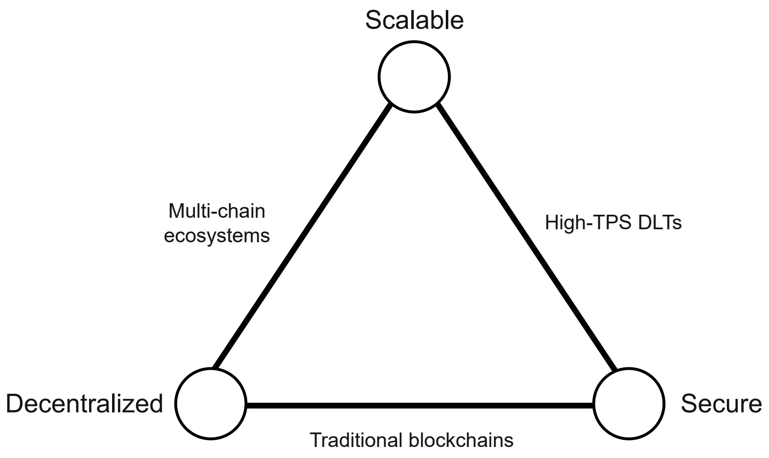


Figure 3.1: The scalability trilemma demonstrating the conflicting requirements of DLTs. Based on [But21d].

more transactions into one. The downside of this approach is that larger blocks are also more difficult to propagate efficiently because of the limited intra-blockchain bandwidth. Additionally, verifying the increased amount of transactions in one block within the interval is hard for individual nodes and could lead to further centralization. The original Bitcoin network only indirectly increased the amount of transactions that fit in a block through the SegWit [LLW21] upgrade [ZHZB20]. Some Bitcoin forks however increased the limit, for instance, Bitcoin Cash to 32 MB [ZHZB20]. Bitcoin SV even removed the fixed limit completely and replaced it by a configurable consensus rule[1], leading to a block reaching a size of 2 GB, that earned more by fees than by block reward[2].

Because block exchange efficiency is also important for the whole network, Bitcoin reduced the data that needs to be sent for a block by implementing a compact block relay (BIP152 [Cor20]). Instead of sending the whole block with all the transaction data, only identifiers are sent. Other nodes already got most of these previously unconfirmed transactions relayed. Therefore, they can reconstruct the block from data they still have in memory. In case some transaction data is not present, it can be fetched with an additional message exchange [ZHZB20].

In a blog post [But21c] Buterin gives an overview on the limits of scalability and reasons why just increasing parameters is not a sustainable solution. He stresses the importance of regular users being able to run a full node and gives estimates on the maximum requirements for nodes regarding computing power, bandwidth and storage.

By replacing the classic blockchain data structure with a DAG, multiple blocks or

[1] https://github.com/bitcoin-sv-specs/protocol/blob/master/updates/genesis-spec.md#block-size-consensus-rule, Last accessed: 2023-03-15

[2] https://coingeek.com/new-1-25gb-2gb-record-blocks-prove-bsvs-economic-model/, Last accessed: 2023-03-15

transactions could be processed concurrently and therefore result in increased throughput. However, such a change opens up challenges regarding consensus, ordering and other scalability issues not necessarily present for blockchains. It may also lead to drawbacks in terms of security or decentralization.

For example, IOTA, in its current version, opted for scalability and security but is not fully distributed because it relies on a central entity, called “The Coordinator”, to reach consensus and to ensure the network’s security. Work to replace it by a distributed approach is ongoing [PMC+20]. More details on IOTA are given in Section 3.5.1.

Part et al. [POK19] describe that even DAG-based DLTs face scalability issues. If the incoming transaction rate rises, the width of the graph grows, leading to an increased duration until the transaction is considered final. To keep the system healthy, they propose to dynamically adjust processing parameters and fee policies. An additional analysis of efficiency and fairness in DAG-based ledger implementations identifying structural limits under high transaction load is given in [BKLM19].

Another way better scalability can be reached is by a change in the consensus strategy. One option is to adjust the way the consensus utilizes PoW. For instance, Eyal et al. [EGSV16] proposed Bitcoin-NG that splits time into epochs and elects a single leader for each epoch through PoW. During the epoch the leader can create multiple blocks containing transactions without the need for additional PoW, resulting in improved scalability and reduced transaction confirmation time [ZHZB20]. Projects actually using Bitcoin-NG as consensus protocol are æternity[3] and Waves[4].

Because of the computational overhead and amount of energy needed for PoW, an alternative mechanism called PoS is used for many DLTs created in recent years. The rationale behind PoS is that users with more stake (coins they hold) in the system are not likely to harm it because that would lead to financial loss [ZHZB20].

Delegated PoS (DPoS) is a variant in which stakeholders elect a small group of delegates by staking. The voting weight is proportional to the amount of coins on stake. Delegates will then produce and validate blocks. Without further measures, DPoS can lead to centralization because the network is under the control of only a small number of nodes that hold the most stake [ZHZB20].

In practice, (D)PoS schemes greatly differ in their workings. For instance, Polkadot uses an own PoS scheme named Nominated PoS (NPoS) [BCC+20]. A limited number of validators are elected for roughly one day based on the stake from nominators. When nominating multiple validators, the stake is distributed evenly among them. In case of a validator misbehaving, the penalties also affect the stake of nominators.

Ethereum 2.0 also uses a PoS-based consensus protocol, where validators each need a stake of 32 ETH and are assigned (pseudo-)randomly to propose a block or to be part of

[3] https://aeternity.com/, Last accessed: 2023-03-15

[4] https://docs.waves.tech/en/blockchain/waves-protocol/waves-ng-protocol#_1-3-brief-summary-of-bitcoin-ng, Last accessed: 2023-03-15

a committee to attest it [Fan21; Jos20].

This assignment is regularly shuffled to avoid attacks on single shards (or blocks) when the attacker controls less than one third of all validators. More details about this consensus algorithm are given in Section 3.2.3.

Off-chain solutions (Layer 2) can be divided into payment channels, off-chain computation and cross-chain communication. Payment channels are utilized for temporary off-chain trading to reduce the amount of transactions in the DLT network and to enable low-cost (micro-)payments. As a first step, a payment channel needs to be opened between two parties, meaning that an amount of tokens is locked on the blockchain. Now, trading via the off-chain channel can happen. In the end, the trades are settled by closing the channel and posting the outcome as a transaction on the DLT network. To avoid establishing new direct channels, it is also possible to build a payment channel network (PCN) allowing for indirect trading between two parties by routing over other participants. For details about how such functionality can be implemented, the reader is referred to descriptions of the Lightning (Bitcoin) [PD16] and the Raiden Network (Ethereum) [Bra]. An issue with PCNs is that users have to be online simultaneously in order to trade and therefore to update the transaction outcome. The generalized term "state channels" describes the same approach as payment channels but instead of just payments, it includes all kinds of state-changing operations conducted in a DLT [CHX18]. A framework for building applications on that idea for Ethereum is *statechannels*[5].

Cross-chain solutions can be separated into sidechain solutions connecting two chains together and overall interoperability projects enabling the communication of multiple different DLTs. Plasma [Etha] (consisting of multiple different variations), originally proposed by Poon and Buterin [PB17], allows to create sidechains for Ethereum with the use of smart contracts and Merkle trees. Plasma chains are working independently and can themselves have chains, resulting in a tree structure. The main chain gets reports from the side chains periodically and is also used for dispute-settling. By taking operations and their data off the Ethereum main chain, Plasma helps with scalability. Additionally, resource-intensive computations can be done on Plasma chains because of lower fees and faster execution compared to the main chain. A downside is that users have to put up with long waiting periods to exit from Plasma chains, because of a challenge period allowing any user to provide a proof that the withdrawal is invalid.

Further promising scalability solutions, such as Sharding, DLT interoperability, and various applications of zk-SNARKs (e.g., rollups and off-chain computation) are explained in the following sections.

3.2 Sharding in DLTs

A promising approach to overcome the scalability/performance problems in blockchain technology is the usage of sharding. In the context of DLT, this works as follows: So-called

[5]https://github.com/statechannels/statechannels, Last accessed: 2023-03-15

shards, consisting of multiple nodes in the network, are defined. Instead of giving all nodes the same transactions to process, they are distributed among these groups. This has several positive effects on the blockchain system. Apart from the obvious improvements in processing throughput due to parallelization, it significantly reduces the communication and storage overhead [WSNH19]. These overheads are the reason why the naive attempt to increase the performance by just increasing the block size is not a proper solution to the scalability problem in DLTs [ZHZB20].

However, sharding, like other scalability solutions for DLTs, cannot easily scale with regards to all required properties without any drawbacks, such as sacrifices in terms of security. Therefore, as stated by Wang et al. [WSNH19], a central, unanswered research question is whether there is a DLT design that achieves scalability for throughput, storage efficiency, and security at the same time.

A more practical definition, further away from the initial notion of sharding, is that it is a term for scalability solutions that are built with multiple interacting sub-networks [Cam21]. Generally, sharding can be realized with a hierarchical or non-hierarchical structure. Further, it can be categorized into two types, homogeneous and heterogeneous sharding.

3.2.1 Hierarchical Sharding

Hierarchical sharding resembles a tree-like structure with one central root element where all other ones descend from. This means that all children respond to the root in some way, whether directly or indirectly by layered interaction working the way up to the top. Therefore, in the end, the root shard is responsible for keeping everything in order. This hierarchical organization not only simplifies the development of the distributed ledger system but also brings significant advantages in terms of error tolerance and scalability.

In terms of error tolerance, the hierarchical sharding approach offers a robust mechanism. Since the root shard plays a pivotal role in overseeing the entire structure, it can detect and mitigate errors effectively. If a child shard experiences an issue, the root shard can step in to maintain the overall integrity of the ledger.

Furthermore, the hierarchical sharding model provides scalability benefits. As the ledger grows and the need for additional capacity arises, new child shards can be created under the root shard. This scalable approach allows the network to expand seamlessly, accommodating increased transaction volumes and data storage requirements. Each child shard can operate independently, parallelizing transactions and computations, thus enhancing the overall throughput.

However, it is important to note that while the hierarchical structure simplifies development and offers error tolerance and scalability advantages, it is also constrained by the capabilities of the root shard.

3.2.2 Non-hierarchical Sharding

In non-hierarchical sharding approaches, there is no central element and no hierarchical tree-like (communication) structure. Hence, the system is much more complex and requires elaborate strategies, especially regarding security and communication. A non-hierarchical approach has no obvious bottleneck caused by its topology and theoretically should be more scalable.

However, unlike hierarchical sharding, where data flows through a predefined structure, non-hierarchical networks must manage communication paths in a more flexible and adaptive manner. This flexibility is essential for distributing workloads effectively, but it also demands sophisticated bandwidth control mechanisms to prevent congestion and ensure efficient data transfer. Even if everything is perfectly efficient, the throughput will still be limited by available bandwidth for the needed communication.

3.2.3 Homogeneous Sharding

Homogeneous sharding works by creating multiple equally functioning shards, meaning that they use the same consensus, state transition functions, and other configurations.

One example is the sharding approach initially envisioned for Ethereum 2.0. Note that the ideas described in the following were partly replaced with other solutions at a later point in time. Shard chains are no longer a planned feature and data sharding will be implemented by the so-called "Danksharding"[6]. . However, the initial approach is still outlined as an example of homogeneous sharding: With the original sharding approach, there would have been 64 "shard chains", which in the first step only provide more capacity for data storage and access. The crucial problem to solve for this "data sharding" is that of data availability. Shards store data and serve attestations for the availability of it in parts. Using these attestations for data availability verification it is possible to build a secure and efficient data layer that can be used in Layer 2 protocols, like rollups [But20]. Data sharding is a pre-requisite to also scale dapps and is easier to accomplish[7].

In a later version handling of transactions and smart contracts could be added although it is still discussed if this extension is needed. According to Buterin, there are three possible ways forward: First, state execution is not needed and shards continue to be just used for data storage needs. Second, as a compromise instead of only one execution shard, a subset of around four to eight of the 64 shards get extended functionality. The third option is to wait for a more matured zk-SNARK technology and adapt the shards to support their usage for e.g., private transactions [Eth21].

A main-chain called Beacon Chain is used as the consensus layer that could potentially also secure shards and enable communication between them. The Beacon Chain is already live and introduced a PoS consensus mechanism to Ethereum Mainnet by an update in September 2022 [Eth23a].

[6] https://ethereum.org/en/roadmap/danksharding/, Last accessed: 2023-09-25

[7] https://twitter.com/VitalikButerin/status/1312009922771443713, Last accessed: 2023-03-15

Although the following paragraphs were written before this upgrade and with the integration of sharding in mind they still convey the workings of the current consensus layer. As DLTs are a fast-paced field of research and development some details may be outdated.

The Beacon Chain's life is divided into epochs. An epoch provides 32 slots, each lasting 12 seconds.

Every slot gives the possibility to add a block to the Beacon Chain. One epoch in advance, all validators are shuffled and evenly assigned to slots. Validators are active participants in the consensus and need to stake 32 ETH by sending them to a specific smart contract. Note that 32 ETH is the maximum stake a single validator can have. At the start of an epoch exactly one validator for each slot, out of all active validators, is randomly (weighted by their balance) chosen to propose a new beacon block. The validators assigned to the slot can then vote on what they see as head of the chain. A vote, also named attestation, is weighted by the validator's balance and based on the outcome of a fork-choice rule, called Latest Message Driven Greediest Heaviest Observed SubTree (LMD GHOST) [BHK+20; Jos20]. This algorithm selects the block as the head that is in the most "active" chain, meaning having the most votes, only considering the latest one per validator. In the following, the attestations are aggregated and included at the earliest in the next block and latest after a delay of 32 blocks [Edg].

Besides the LMD GHOST vote validators also have to vote for the so-called checkpoints of the current (target) and the previous epoch (source). This additional vote is called Casper Friendly Finality Gadget (FFG) vote It also can be described as referencing the transition from the previous checkpoint to the next. Checkpoints need at least 2/3 (the supermajority) of the total voting weight of active validators to be accepted. A target checkpoint (and all other previous blocks) is considered "justified" once it reached supermajority and "finalized" when it achieved the same as source checkpoint. The double-vote process is used to avoid additional communication between nodes which would hinder the scalability of an increasing amount of validators. Getting a block finalized on average takes the time of slightly over two epochs (around 14 minutes) and can be compared to the required amount of block confirmations used in PoW. Finalized blocks are added permanently to the blockchain and cannot be reversed [Fan21; Jos20].

In case the proposer did not provide a block, e.g. because it is offline or out of sync, the slot will stay empty and a new block will only be added in the next one. Multiple incentives and disincentives ensure good behavior of proposers and attestors (voters) [Jos20].

By the time shard chains were still part of the Ethereum roadmap it was planned to upgrade the Beacon Chain to randomly assign shards to committees for each slot [Eth21]. A committee is a group of at least 128 validators made up of the previously shuffled validators in a specific slot. This amount was chosen under statistical considerations to keep the probability of having 2/3 malicious participants in one committee at a reasonable level. As a result, it is improbable that an attacker in control of less than 1/3 of all validators can attack a shard. To process all 64 shard chains in each of the 32 slots in an epoch, there must be a minimum of $128 * 64 * 32 = 262144$ active validators. In case that number is not reached, any given shard will only be processed once every few slots. Validators in the committees also vote for the head of their respective shard chain in addition to the Beacon Chain's head [Jos20].

3.2.4 Heterogeneous Sharding

Usually, DLTs target specific features tailored to some use cases or try to fit as many as possible. Both cases lead to various tradeoffs. Decisions have to be made which approach and specific configuration to use for the transaction model, the consensus mechanism, and the ledger model (UTXO or account-based). This severely limits flexibility and forces enterprises to scatter their services over multiple DLTs suitable for the different business scenarios. Besides the high implementation effort, another downside is that a seamless interaction between the projects is difficult to realize or not possible at all. With traditional sharding, the scaling is done by adding shards with identical specifications which, for instance, are optimized to scale-out computation capacity but make drawbacks otherwise.

Heterogeneous sharding, on the other hand, allows each shard to be configured differently, e.g., in terms of consensus mechanism. The flexibility coming with heterogeneous sharding allows to adapt more easily to future demands. This concept is similar to using sidechains and cross-chain communication with the difference that it is more tightly coupled and also protects the shards against attacks, whereas this has to be done by every single other chain itself [BCC+20; Qua20].

Several projects in the field of distributed ledger interoperability adopt the notion of heterogeneous sharding. While they share many common properties, they also differ in implementation specifics, current feature completeness, and openness regarding joining the networks [KP19]. Examples of such systems are Polkadot and Cosmos, which are both explained in more detail in the following section. Koens and Poll [KP19] identified that they share the planned functionality and both aim to include any ledger (reach). Furthermore, they store the state in multiple places (scope) and have validators to ensure semantic and syntactic interoperability.

3.3 Distributed Ledger Interoperability

This field of research generates a wide variety of publications and grey literature with a multitude of different approaches and goals.

In their extensive systematic literature review, Belchior et al. classified the analyzed work into three categories: Public Connectors, Blockchain of Blockchains, and Hybrid Connectors, each with several sub-categories. Relevant sub-categories for this thesis are explained below. Additionally, the authors made a critical observation about interoperability in their work, highlighting that it not only combines flexibility and portability, but also promotes scalability by enabling transaction offloading to other ledgers, such as through sharding [BVGC21].

3.3.1 Public Connectors

Approaches in this category consist of strategies to achieve interoperability across public ledgers for use cases like cryptocurrency trades and moving assets.

Notary schemes

In that scheme a *notary* is connected to multiple ledgers. When an event occurs on one ledger the notary creates a corresponding transaction on another ledger. This means that notary schemes act as intermediaries between ledgers. A practical example of notary schemes are exchanges which can be either centralized (EX) or decentralized (DEX). An exchange is centralized if it holds the users' funds and executes trades, for example buying cryptocurrencies conditionally, on their behalf. If the trust is not put into a centralized entity and the exchange just provides a matching between buyers and sellers with the help of smart contracts it is considered decentralized. The tradeoff is comfort and speed (EX) versus security (DEX).

Relays

For a relay to work "relayers" keep track of the source ledger data and feed it into a smart contract running on the target ledger. The first example of such a system was the BTC Relay smart contract hosted on Ethereum to relay Bitcoin data. For each new block published on Bitcoin, the header data needs to be submitted to the contract. There the header gets validated and in the following can be utilized to verify Bitcoin on-chain information such as the inclusion of transactions in a block. This is made possible by the Merkle trees stored inside the header data. Now anyone can use the relay to verify a transaction and forward the confirmation to another smart contract which then acts on that information.

However, BTC Relay only accepts new block headers if they refer to a previously successfully transmitted and validated block header. Therefore it is required to submit every single intermediary header missing in the contract until the latest one can be added. Although there is an incentive mechanism in place to compensate the relayers it does not cover that overhead sufficiently. As a result, gaps will form over time and it will become less and less practical to use the relay, due to the high costs [FSS+20; WE20].

Frauenthaler et al. proposed "ETH Relay", a relay solution for Ethereum-based blockchains that avoids high operating costs by well-defined incentives and only validating blocks on demand [FSS+20]. Contrary to block validations for Bitcoin, the hash function needed for Ethereum is computationally expensive to compute on-chain. Therefore, in addition to the issues mentioned for BTC Relay, it would not be economically feasible to operate such a relay when fully validating each block.

ETH Relay works by optimistically accepting block headers. At first, it only validates if the header has not yet been submitted and the referenced parent block is already known.

Newly added blocks cannot immediately be used for verifying transactions but are locked for a certain amount of time.

During that time, they can be disputed. A reason could be that clients have noticed it is an invalid block header by monitoring the relay as well as the actual source blockchain. Only in the event of a dispute, a full validation of the header in question is done by the relay contract. If it fails, the disputed block together with its potential descendants is removed from the contract. In case the lock time of a block passes without a dispute it automatically is considered valid but still needs multiple confirming blocks until it can be used for checking the inclusion of transactions.

To make the described scheme work in practice an incentive structure compensates clients for submitting and disputing blocks. Submitting a block requires a stake that can be earned by disputants if it turns out the block is invalid. Additionally, clients receive a fee each time a valid block they have submitted is used.

“Dogethereum”, a relay between Dogecoin and Ethereum proposed by Teutsch, Straka, and Boneh utilizes Bulletproofs in order to validate Dogecoin’s PoW off-chain, as the execution of the underlying memory-hard *scrypt* function is infeasible in smart contracts. Although the validation of blocks happens in batches the proof size of Bulletproofs grows too fast, leading to excessive verification costs on-chain. As a consequence, an optimistic approach with an incentive scheme that rewards the detection of incorrect proofs is used to minimize on-chain computation [TSB19; WE20].

Two additional relays are described in later sections: “zkRelay” in Section 3.4 because it utilizes zk-SNARKs and “Verilay” a relay for PoS-based ledgers in Section 4.1.2 to guide the design of the IOTA relay proposed in this work.

3.3.2 Blockchain of Blockchains

The concept of “Blockchain of Blockchains”(BoBs) refers to frameworks designed to facilitate the seamless creation and integration of application-specific blockchains capable of efficiently interacting with each other. This is achieved by providing comprehensive guidelines, robust SDKs, and reusable components addressing the fundamental layers of DLTs, including data, network, consensus, incentive, and contract layers.

BoBs implementations can be seen as building a network of relays and sidechains because usually, a main chain (sometimes referred to as a relay chain) acts as a (central) connector for multiple secondary (application-specific) chains. That approach results in high throughput but still gives users flexibility by offering interoperability capabilities between various attached chains. In addition to the preferred framework-specific interoperability scheme, BoBs also include components, often called bridges, that utilize various mechanisms to allow interactions with other types of blockchains.

Differences between specific BoB projects manifest in how generic or coupled their approach is. For example, how much customization and control are possible for each connected chain. Either there is a high degree of freedom where a chain can handle security and validation on its own or customization is limited but in turn, shared security layers are in place. This security-customization trade-off also has effects on the decentralization and robustness of the system.

Although BoBs theoretically pave the way for better blockchain interoperability, in the end, they just shift the problem from single isolated blockchains to competing BoB projects that do not efficiently integrate with each other. Two widely adopted examples, Polkadot and Cosmos, are described in more detail in the following sections.

Polkadot

Polkadot [BCC+20], initially proposed by Wood [Woo16], consists of one main chain, with multiple heterogeneous shards, called parachains (from "parallel chains") [Web21]. The main chain is referred to as the Relay Chain. It is in charge of validity and availability checks, message exchange between parachains as well as providing security guarantees to the whole network. In other words: Polkadot is a sharded state machine offering shared security that enables trust-less messages. Cross-chain communication in a system only built by bridging independent sidechains without shared security cannot be trustless because the receiving chain has to fully trust the sending one.

However, chains securing themselves have the advantage of greater sovereignty. A general bottleneck and security issues of one chain affecting all the others in the system can be avoided. Cosmos [KB19] is an example of a blockchain ecosystem, that does not require but optionally supports a group of shared security protocols called Interchain Security[8]..

Polkadot incorporates multiple components with different roles. Collators are responsible for assembling transactions from their parachain into a block but do not need to give security guarantees. They are running as a full node for both the parachain and the Relay Chain in order to receive all the necessary information. The candidate block is then transferred to one of the validators currently assigned to the parachain. For each block, the mapping of validators to parachains is determined randomly. Validators verify the block by checking the included state transitions against the state transition rules of the parachain. The state of the parachain is stored in a Merkle tree. Therefore, all the validator needs for a proof of validity is the block (list of state transitions), the values that are modified and the hashes of the unaltered parts of the Merkle tree. This also means that Polkadot only guarantees a valid state transition and not a valid state. If a parachain joins with a valid state and all changes are executed protected by Polkadot, the state is valid. The proof of validity is gossiped among all validators currently responsible for the parachain. When more than half of them agree that it is correct they construct a "candidate receipt", which is the data that will be part of the Relay Chain block.

Another component previously proposed for Polkadot but not implemented are the so-called Fishermen. Their role would have been to be full nodes of the parachain that watch the parachain block creation process to make sure that no invalid state transactions

[8]https://cosmos.github.io/interchain-security/introduction/overview, Last accessed: 2023-09-24

are packaged [Pol21a]. Clear information on why this is not necessary anymore or how this goal is achieved instead could not be found.

For arbitrary messages passed between parachains, Polkadot uses the Cross-Consensus Message Passing Format (XCM)[9]. A detailed description of XCM can be found in a blog post by Gavin Wood [Woo21b]. In the Cross-chain Message Passing protocol (XCMP), which is still under development[10], Collators serve as message routers because they are full nodes for both the parachain and the Relay Chain [Pol21c]. The actual messages do not pass through the Relay Chain. To ensure scalability, only proofs about them and operations for managing communication channels opened between parachains are stored. Until XCMP is finalized there is a stop-gap protocol, called Horizontal Relay-routed Message Passing (HRMP), available which implements the same functionality but is not as efficient because it stores all messages in the Relay Chain [Pol21b].

Cosmos

Cosmos [KB19] uses the so-called Cosmos Hub (a blockchain itself) to transfer messages between zones (shards), similar to Polkadot's Relay Chain connecting multiple parachains [BCC+20]. Although the Cosmos Hub is cited the most, multiple (nested) hubs are supported and there already are some in operation. In the following, *the Hub* refers to the Cosmos Hub, while *hub* is used for any hub in general. Different from Polkadot, Cosmos does not include the full state of its zones in the state of the Hub. Therefore, as already stated, it (currently) does not provide shared security guarantees and the separate zones need to have their own validators. This means that Cosmos as a whole system is only as secure as its least secure zone and users need to trust the individual zones to keep the state history.

The IBC (Inter-Blockchain Communication) protocol [Cos21; Goe20] used in Cosmos aims at being a base protocol to build and iteratively improve upon, similar to how HTTP builds on TCP. For now, it focuses on allowing cross-chain token transfers. Every DLT implementing the IBC interface and meeting several other specifications could directly interact with each other but it is assumed that most projects will connect to hubs. This resembles the Internet's network infrastructure which also consists of hubs instead of building direct connections to all the services one wants to communicate to [Cho19]. By using hubs, the efficiency is higher due to avoiding the quadratic growth of direct connections and also the ease-of-use is improved because new zones just need to join a hub to automatically be interoperable with all other participants [Dah21].

While you need to gain the right to connect a parachain to Polkadot by auction [Bas21], in Cosmos anyone can connect to the Hub. When transferring tokens over the Cosmos Hub, it keeps track of the total amount of tokens held by each zone. To transfer a token, the transaction has to be acknowledged by the sender, hub, and receiver. However,

[9] https://wiki.polkadot.network/docs/learn-crosschain, Last accessed: 2023-03-15

[10] https://wiki.polkadot.network/docs/learn-xcm-transport#xcmp-cross-chain-message-passing, Last accessed: 2023-09-24

transactions committed outside the Hub are not verified. For that reason, users are advised to only send tokens to zones they trust, and receiving zones must trust the security of the originating zone.

3.4 zk-SNARKs

Another relevant topic often neglected or left open for future work is the performance of and the necessary computational resources for zk-SNARK proof generation. Depending on the number of inputs, the proof generation can require more RAM than available at the average personal computing device. Therefore, memory often is the scalability bottleneck for the prover as it grows with the statement size [BNTT20]. Westerkamp and Eberhardt propose to split the proof into multiple ones in that case but even then it may not be feasible for, e.g., computationally-constrained IoT devices [GKO20; WE20]. Garoffolo, Kaidalov, and Oliynykov [GKO20] write about recursive SNARKs which merge multiple proofs into one but also mention that this requires a significant amount of computation and different approaches still need to be researched. A project using this in the field is Mina [BMR] (formerly Coda). It works by computing a SNARK proof validating the previous proof and the new blocks together. This enables state transition verification in constant time, independent from the number of previous blocks. Differing to common blockchains, a proof size of just 864 bytes and a verification time of 200 ms is making a full verification practical, even for computationally weak devices.

As part of Layer 2 solutions, zk-SNARKs are a promising way to scale DLTs. They can be utilized to build so-called "zk-rollups", that bundle many transactions into batches for off-chain execution. The result of the rollup is a summary of all the changes required by the transactions together and a proof stating their correctness. A smart contract on Layer 1 verifies the proof and executes the state changes [Eth23b]. Research to create a whole zero-knowledge EVM implementation that allows proving general-purpose computations is ongoing [But22].

Depending on the scenario, novel zk-SNARK proof systems can lead to considerable performance improvements. "Spartan" [Set19] is counted as one of the best performing schemes and an open-source library[11] written in Rust is available. Additional implementations of proof schemes from [Gro16] and [BCTV14] exist. Examples are the C library "libsnark"[12] and "bellman"[13] which is available for Rust. Initially, Zcash built on libsnark for implementation of the Pinnochio protocol [PGHR13] but currently, bellman, as a realization of [Gro16] is in use [Zcac].

zk-SNARKs can also help in building better relays. Approaches previously described in Section 3.3.1 rely on on-chain proof effort linear to the number of blocks or additional economic incentives. Westerkamp and Eberhardt propose a different, novel approach for

[11] https://github.com/microsoft/Spartan, Last accessed: 2023-03-15
[12] https://github.com/scipr-lab/libsnark, Last accessed: 2023-03-15
[13] https://github.com/zkcrypto/bellman, Last accessed: 2023-03-15

a relay. Instead of verifying block headers directly in the relay contract, their solution "zkRelay" [WE20], collects blocks into batches and executes the verification off-chain. By utilizing zk-SNARKs a cryptographically secure proof of this verification is created. That proof is then validated on-chain in constant time. Only the last block header of each batch is stored in the contract. To still allow for verification of transactions inside intermediary blocks a Merkle tree is built based on the hashes of all the block headers included in the batch. The root of this Merkle tree is stored in the relay and can be utilized to prove the inclusion of headers in the batch, similar to how proving the inclusion of transactions works. This specific Merkle tree is built with Pedersen hashes that, contrary to the often-used SHA-256, can be used efficiently for zk-SNARKs. The reason is that Pedersen hashes are based on elliptic curve cryptography, while SHA requires modulus operations that lead to a huge number of constraints in the proof system.

Garoffolo, Kaidalov, and Oliynykov [GKO20] propose a cross-chain protocol for native fund transfers between main- and sidechains in a parent-child relationship using zk-SNARKs. Their approach allows decoupled sidechains which may work differently in terms of their "structure", e.g., what type of consensus is used. They also describe a concrete sidechain using PoS communicating with a PoW mainchain, both on a blockchain basis. Contrary to blockchains, research on DLTs using DAGs is limited. Understanding, if a similar approach is feasible with a DAG-based DLT and what data is needed for the application of zk-SNARKs in this context, are still open questions.

3.5 DAG-based DLTs

Although many DAG-based DLTs have already been proposed, it is still a relatively new area of research, compared to blockchains. Thus, the number of papers giving a systematic overview on the topic is limited, making contributions to the field laborious, especially for newcomers. In a Systematization of Knowledge (SoK), Wang et al. [WYCX20] summarize their findings of both reviewing academic work and analyzing open-sourced systems including accompanying self-published white papers. Besides classifying existing approaches based on structure and consensus, they also provide an analysis of security and performance. Furthermore, the differences of other scalability techniques compared to a DAG-based approach are highlighted. To foster future work, they also list several open research topics for DAG-based systems. Most relevant to this work are the questions on how to enable sharding and cross-chain communication. Other noteworthy issues are the implementation of smart contracts and the effect of feeless systems (meaning the absence of incentive mechanisms). Popular examples[14] for DAG-based cryptocurrencies include IOTA [PMC+20; Pop18], Hedera Hashgraph [BL20], and Nano (formerly RaiBlocks) [LeM18].

Table 3.1 gives an overview of selected attributes of the three DAG-based projects, that are explained in more detail in the respective sections. Text in square brackets hints at changes or updates planned for future updates. Although all systems build upon a DAG

[14]Listed in order of market capitalization by 2021-04-21 from https://coinmarketcap.com/

as a data structure its actual usage varies. Other noteworthy differences can be found at Hedera: it currently only consists of permissioned nodes and runs a patented consensus algorithm. Furthermore, contrary to IOTA and Nano it is not feeless and even specifies the fees in US dollars. The different DAG structures are described as Divergent and Parallel. Divergent means that units form the DAG in a random way without a specific order while Parallel refers to keeping them in multiple chains.

Table 3.1: Comparison of selected DAG-based DLTs.

	IOTA	**Nano**	**Hedera Hashgraph**
Unit	Transaction	Transaction	Event
Tx model	UTXO	Account-Pair	Account
Openness	Permissionless	Permissionless	Permissioned [Permissionless]
Centralized	Partly [No]	No	No
Structure	Divergent	Parallel	Parallel
Patented	No	No	Yes
Consensus base	Tip selection	Voting	Voting
Spam Protection	PoW [Adaptive]	Prioritized PoW	Congestion pricing
Signature	Ed25519	Ed25519	Ed25519, RSA, ECDSA
Fees	Feeless	Feeless	Pricing schedule (USD)

3.5.1 IOTA

The initial theoretical foundation of IOTA and its DAG data structure called “the Tangle” was described by Popov [Pop18]. As quite common in the field of DLTs, the practical implementation soon deviated to different extent from the early concepts. Details on the current state of the protocol can be found in the developer documentation[15] which is based on multiple git repositories. At the time of writing, IOTA is at the halfway point to realize the visions outlined by the IOTA Foundation (IF) in the whitepaper “The Coordicide” (also referred to as IOTA 2.0) [PMC+20].

In the beginning, IOTA was built with the decision to use quantum-resistant cryptography for signatures and implement core systems based on the ternary number system[16]. Theoretically, this system is more efficient, due to the better *radix economy*[17] [Geo16] but in practice these gains are diminished because almost any computer system uses binary, making additional conversions and encoding steps necessary [IF21a].

[15] https://docs.iota.org/, Last accessed: 2023-03-15
[16] https://legacy.docs.iota.org/docs/getting-started/1.1/the-tangle/ternary, Last accessed: 2022-06-09
[17] https://iota.stackexchange.com/questions/8/why-does-iota-use-a-ternary-number-system, Last accessed: 2023-03-15

IOTA used the Winternitz One-Time Signature scheme (W-OTS). This scheme is quantum-resistant and therefore future-proof but in turn, comes with the huge drawback of being vulnerable when reusing addresses (keys) [BDE+11; IF21a]. Additionally, the size of the signature is relatively large with up to 3900 bytes (depending on the chosen security level)[18].

For these reasons, the latest update dubbed "Chrysalis" (also known as IOTA 1.5) [IF21a] remedies both of the stated issues by re-implementing algorithms to use binary encoding and replacing W-OTS with the Ed25519 signature scheme. By using atomic transactions instead of bundles consisting of multiple, mandatory, fixed-size parts, the transaction size was reduced from 1.7 kB to below 100 bytes [IF21d]. This not only results in less overhead for network communication and signature verification but also makes Merkle proofs, which may be used for a future sharding solution, significantly shorter [Moo19].

To improve scripting of transactions and to ease future additions, for example, digital assets, the previously used account-based model was replaced by UTXOs. All these changes together with optimizations including the tip selection algorithm lead to a confirmation time of less than 10 seconds [IF21d].

The Tangle, illustrated in Figure 3.2, consists of vertexes representing transactions and edges pointing to previous transactions to approve them [Pop18; ZY19]. Originally, each transaction had to reference two others but since IOTA 1.5 it is possible to choose between one and eight. More precisely, not transactions directly but *messages* are used [IF21a]. Currently, normal messages have four parents, as can be observed on the "Tangle Explorer"[19].

These messages with a size of up to 32 KiB can carry different types of payload such as transactions describing a value transfer. In the following, the terms *transaction* and *message* in context of IOTA may be used interchangeably if not stated otherwise. Ideally, new transactions approve other previously unapproved ones, called tips. It is also possible to reference already approved transactions because it could happen that discovered tips were approved by other transactions in the meantime. Before approving a transaction, it is verified to be positive and not conflicting with previous ones. After reaching enough direct or indirect approvals, a transaction is considered a fixed part of the Tangle.

Discovering and choosing tips is handled by a tip selection algorithm. For instance, the weighted random walk tip selection algorithm works as follows. It starts walking from the genesis transaction towards tips by choosing the path partly based on the cumulative weight (direct and indirect approvals plus one). This behavior can be compared to selecting the longest blockchain in Bitcoin. A parameter controlling how much impact the weight has for path selection is used to avoid always orphaning transactions that approve older ones instead of tips (e.g., due to network delay). Finding an optimal value and therefore balancing the randomness used for tip selection is important for the stability of

[18] https://github.com/iotaledger/tips/blob/ee07797acb/text/0000-ed25519-signature-scheme/0000-ed25519-signature-scheme.md, Last accessed: 2023-03-15
[19] https://explorer.iota.org/mainnet, Last accessed: 2023-03-15

the Tangle. In case transactions are not approved for some time, it is possible to promote them by issuing another empty transaction referencing the unapproved one together with a recent tip. Whenever a transaction receives more direct or indirect approvals, its cumulative weight increases making it more likely to stay in the Tangle. Changes applied with the Chrysalis update made it possible to use a different, more efficient algorithm called Uniform Random Tip Selection (URTS) [Rog21]. This was necessary because the cumulative weight computation needed for the original tip selection turned out to be too resource-intensive to reach high throughput.

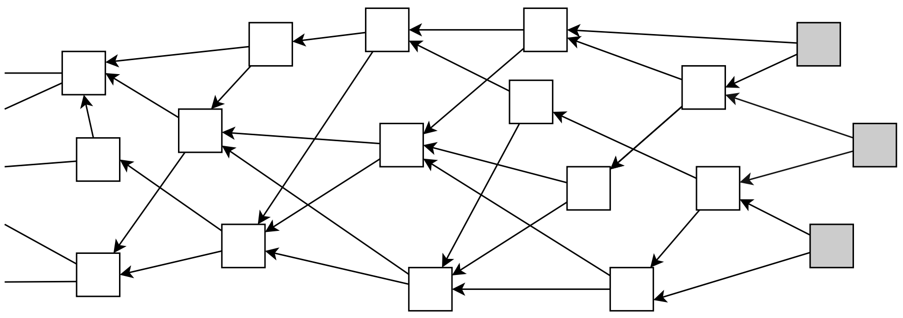


Figure 3.2: Schematic diagram of the IOTA Tangle. Grey squares represent tips where new messages will be attached. Based on [Pop18].

Because IOTA is feeless, it also is prone to spam attacks. For this reason, a small PoW is required to send a transaction. A value called *PoW score* defines how many hash computations per byte of a message are needed on average to find a valid nonce [Wel20]. It is important that a PoW algorithm (usually based on a hash function) is relatively hard to compute but very easy to validate. IOTA uses a custom hash function called "Curl-P-81" to compute the transaction hash. The number of trailing zeros in the trit representation of the hash value determines if the targeted difficulty is fulfilled. The PoW computation is the last remaining part of IOTA using ternary conversions. This decision was made to keep the disruption to the protocol and related projects as low as possible. However, with Chrysalis optimizations were made to avoid the validation bottleneck caused by the slow performance of Curl. By first using the faster BLAKE2b-256 function to hash the message and taking the shorter fixed-size digest as input to Curl, the validation speed was improved. Because network nodes need to keep track of UTXOs, a huge number of low-value transactions ("dust") could lead to performance and memory issues. Therefore a dust protection, limiting the allowance based on the amount of deposit the target address is holding, is in place. This way such spam attacks are expensive but micro-payments are still allowed by making a deposit.

Currently, IOTA does not use any type of sharding, but approaches for different parts of the project are being researched [Cam21]. The first step would be to implement a hierarchical sharding mechanism for data messages only. Not supporting transaction payloads allows for a far simpler concept. New data shards (Child Tangles) can be created by any node and configured to be permissioned/permissionless and public or private.

Nested shards are possible but in general, it is more efficient to keep the hierarchy shallow, as child tangles need to send message stamps to their parents and wait for confirmation there to reach finality. Proof of inclusion for specific messages in a data shard can be done using a chain of hashes, e.g., as a Merkle tree, without being part of the respective Child Tangle. Altogether, this allows to decouple data from the main tangle and therefore making it lighter. By shifting the data to shards it can be (indirectly) included in the main tangle without limitations. Although the concept of data sharding was discussed there is no recent activity on this topic[20].

A non-hierarchical concept called "Fluid Sharding" is in discussion[21] for further research in the more distant future (targeting IOTA 3.0). Its main feature is the fluidness or adjustable nature of requirements taking into account the available resources of the respective nodes joining the network. The idea is to configure a location together with a radius of perception for each node. Nodes may adjust the radius to fit their capabilities and then only communicate to neighbors within their reach. Among others, there are open questions regarding the security of the system, inter-shard as well as long-range communication [Cam21].

IOTA does not yet have support for smart contracts in the mainnet. Nevertheless, Drąsutis presented the reasoning behind a potential design for an IOTA Smart Contracts (ISC) framework [Drą21]. In an effort to avoid the bottleneck of smart contract systems that proposal follows a horizontal scaling approach by extending the Tangle at Layer 1 with a multi-chain Layer 2 environment anchored to it. Each of these Layer 2 blockchains has its own set of validators and can independently run smart contracts. Although they can execute in parallel, smart contracts can call others not only within one chain but also across them by way of the Layer 1 ledger. For now, the ISC framework is only partly available in the IOTA 2.0 DevNet and the testnet of Shimmer[22].

Shimmer is a staging network for novel features, like updated consensus mechanisms, before they eventually make their way into the IOTA mainnet. The IF also envisioned a third network, called "Assembly". However, the project was discontinued in favor of focusing development resources and integrating the collected insights directly in the IOTA mainnet [IF23]. The plan for Assembly was to implement the full ISC framework but also add a permissionless pool of decentralized validators to reach shared security over the participating smart contract chains. A main chain would have handled all the functions for the validators such as keeping track of their trust scores, rotating the pool, paying out rewards, or slashing stakes. In other words, the idea was to build a governance layer on top of IOTA's Tangle enabling a permissionless ecosystem of multiple smart contract chains and validators [IF21b].

[20]Data sharding. https://govern.iota.org/t/data-sharding/1188, Last accessed: 2023-09-24
[21]Fluid sharding. https://govern.iota.org/t/fluid-sharding/1285, Last accessed: 2023-09-24
[22]https://blog.shimmer.network/shimmerevm-testnet-launch/, Last accessed: 2023-08-16

3.5.2 Hedera Hashgraph

Hedera Hashgraph [BHM20] in its current state is a permissioned but decentrally governed DLT built using the Hashgraph consensus algorithm invented by Baird [Bai16a; Bai16b; BL20]. Until it eventually reaches the state where everybody is allowed to run a node, only governing council members can do so. The council currently is constituted by 29 (up to 39) organizations[23] distributed around the world such as Google, University College London, and Deutsche Telekom..

It is noteworthy that the rights and patents to the Hashgraph consensus algorithm are held by the company Swirlds and its use is exclusively licensed to Hedera [BL20]. Swirlds is also the only permanent member of the Hedera council.

Additionally to legal protections, also technical control mechanisms are in place. Clients can request state proofs that contain the public keys of all nodes ("address book") needed to check the signatures of the state. Apart from this current key material, also a history of it is kept. Any new address book needs to be signed by nodes in the previous one with at least 2/3 of the total coins staked to them. The hash of the genesis address book is used as the unique identifier of the ledger. If some nodes try to fork, they cannot practically create an address book history reaching back to the genesis one because most nodes will not sign this new address book. Therefore, the fork is forced to have a different genesis address book resulting in a new unique identifier, avoiding any risk of clients being deceived into using the fork.

Baird, Harmon, and Madsen [BHM20] argue that such protections are beneficial for enterprise applications built on top of the system because they block network forks and their negative effects leading to better adoption by mainstream markets. Furthermore, the implementation is not fully open-sourced. The components making up the Hashgraph Platform, including the gossip and consensus protocol code, were released for open review in fall 2020[24], while the services and tools building on top of it are open-source since August 2020[25].

The hashgraph consensus mainly works by means of a gossip protocol. Nodes randomly choose other members of the network and tell them all the information they currently know. This process is repeated until all members are aware of any new data. Each synchronization between two parties creates an event (depicted as a circle in Figure 3.3) containing a timestamp, transactions, hashes of respective previous events and a signature. With these event hashes, a history of how events are interconnected and therefore how members communicated is recorded. This "gossip about gossip" results in a DAG, or as it is called by Hedera, a *hashgraph*, displayed in Figure 3.3. All nodes maintain a continuously updated copy of the DAG. To reach consensus, additional to knowing every event, it is necessary to agree on a linear ordering of them.

[23] https://hedera.com/council, Last accessed: 2023-09-24
[24] https://hedera.com/blog/hashgraph-platform-is-now-available-as-open-review, Last accessed: 2023-03-15
[25] https://hedera.com/blog/all-our-services-are-belong-to-you, Last accessed: 2023-03-15

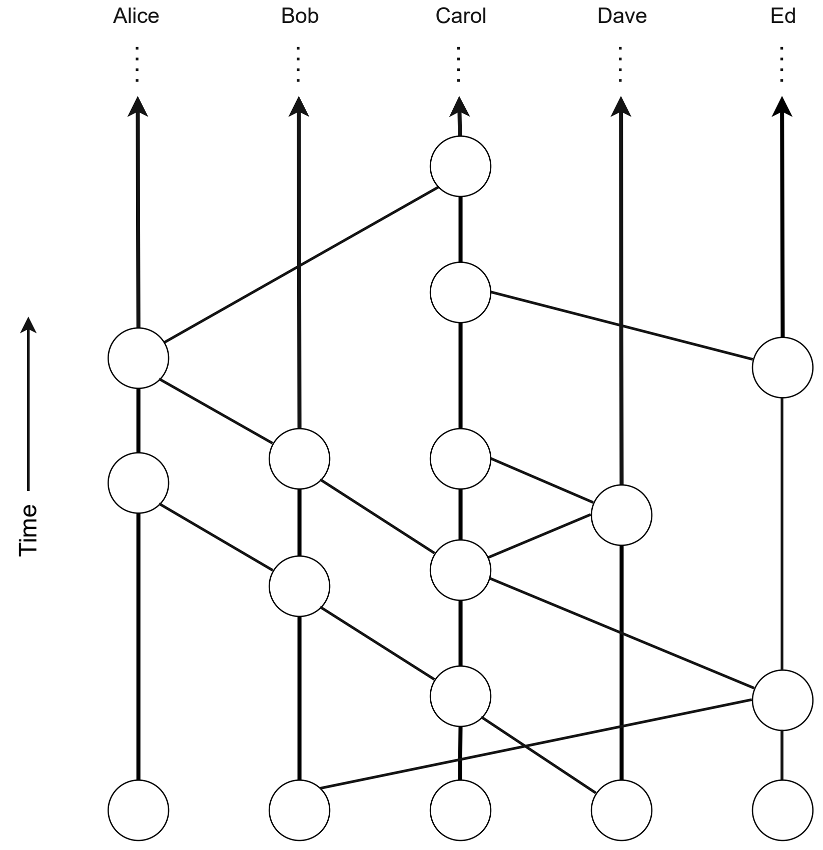


Figure 3.3: Hedera Hashgraph gossip history. Based on [Bai16b].

Hashgraph does not require separate voting messages to be sent, hence avoiding bandwidth consumption which often leads to scalability issues on bigger networks. Instead, a *virtual voting* algorithm is used. Every node can locally calculate the votes of other members based on the gossiped data according to the rules. The added benefit of this local approach is that a malicious node cannot directly influence the calculated voting result at honest nodes by, for instance, manipulating voting messages in any way.

The order of events is then decided by the consensus round they were received in, the (weighted) median of its timestamps, and the signature. The basic version of the hashgraph consensus assumes that each member is equal, but in Hedera, the algorithm is adjusted to include PoS. Nodes need to stake at least some coins to participate in consensus, while users of the network not running an own node can proxy-stake towards chosen ones. Since Hedera is still in a permissioned phase, only allowing council members to host nodes, the Hedera treasury proxy-stakes more than two-thirds of the total coin supply to nodes run by members. Proxy-staked coins are not locked and can be spent or reassigned to a different node freely. As of May 2021, normal users cannot proxy-stake yet[26]. It also is not needed currently because all nodes are permissioned.

To avoid too many people proxy-staking to a single node, the voting weight of a node, based on the sum of the stake it controls or that got proxied to it, is capped. An important difference between Hedera's proxy-staking model and DPoS is that users directly impact the consensus by increasing the voting-weight of the node they proxy-stake to, while in case of DPoS only the vote on delegate candidates, which in the end control the consensus, is influenced [Hed19].

Baird, Harmon, and Madsen state that the hashgraph is DoS resilient as it does not rely on nodes with special rights but all nodes participate in the consensus. Therefore, even if a small number of nodes are attacked, the network can still operate successfully. To create more disincentives, the implementation of "automated congestion pricing" is listed on the roadmap[27]. By adjusting the network fees in real-time, when it is used excessively, denial of service can be prevented further.

Another not yet available but described feature of Hedera is sharding. Currently, the network consists of a small number of nodes in a single shard. When more nodes join the network, it can be split into multiple shards to distribute transaction processing and allow parallel consensus. These shards trust each other and can communicate by means of push messages. Randomly chosen subsets of all nodes will be assigned to different shards that keep track of a subset of the ledger's state. A master shard assigns new nodes to shards randomly once a day and also ensures a secure distribution of stake by relocating nodes to other shards. Nodes of a shard only contribute to the consensus of transactions originating from within. Via state proofs and unique ledger identifiers (already described in context of the technical controls against forks) the message origin and state of shards can be validated. Each shard keeps a queue of outgoing messages and a sequence number for incoming messages for all the others. After a cross-shard transaction is part of the consensus state in shard A, a node sends a proof of it with a sequence number to a random contact in shard B. This procedure is repeated until a node of shard B responds with a proof that the shared state contains the given sequence number, meaning that it and all previous messages were received and processed. Using the sequence number, the order can be maintained and duplicates can be ignored.

To give incentives for running a node Hedera uses three types of fees: node, network, and service fees. Node fees are paid by the user to the node used to submit a transaction to the network. Network fees compensate for the resources needed to reach consensus. They are paid by users to the Hedera treasury. Service fees are paid to the treasury for instance when using the file storage or smart contract capabilities of Hedera. Income by fees paid to the treasury are regularly distributed to participating nodes. Users proxy-staking will also receive small amounts as a reward. The fees are calculated based on a pricing schedule set in USD by the governing council[28].

[26]https://help.hedera.com/hc/en-us/articles/360007176997-Will-Hedera-add-additional-services-to-the-platform, Last accessed: 2023-03-15
[27]https://hedera.com/roadmap, Last accessed: 2023-03-15
[28]https://hedera.com/fees, Last accessed: 2023-03-15

3.5.3 Nano

Nano [LeM18], a permissionless and feeless DLT, was initially proposed by LeMahieu in a whitepaper last updated in 2017. Nowadays, most of the whitepaper's content is outdated due to multiple protocol improvements over the years. The latest information can be found in a "living whitepaper" [Nana] which is maintained by the community in a git repository[29]. In the following, this is used as a general citation without reference to the specific sub-pages.

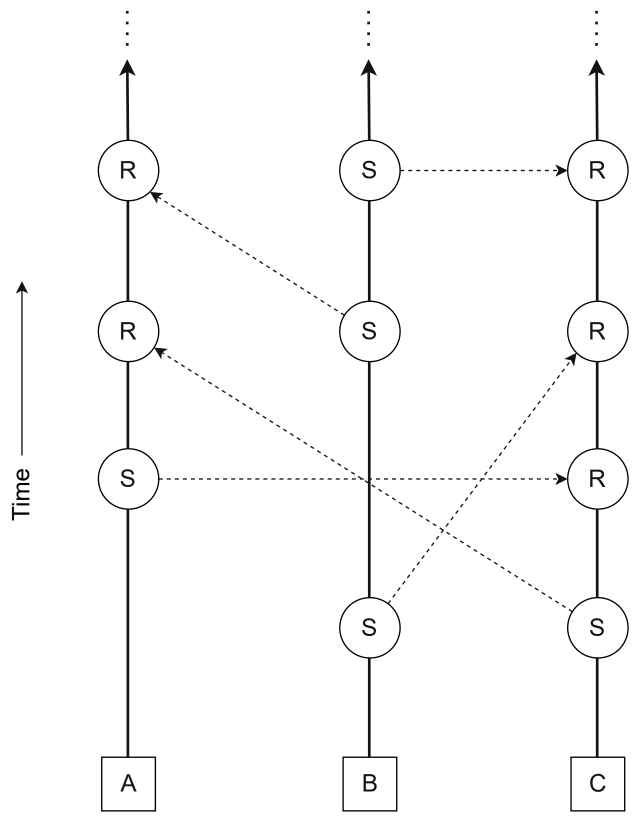


Figure 3.4: Nano's block-lattice structure. Each transfers consists of a send (S) and receive (R) block signed by their respective account (A, B, C). Based on [LeM18].

Instead of a classical blockchain, Nano uses a data structure called *block-lattice*, where each account has its own blockchain, illustrated by the squares and solid lines in Figure 3.4. Transfers between two accounts only occur with pairwise actions. To *send*, a block (circle with S) is attached to the sender's blockchain which conducts the chosen amount. Later, the receiver needs to create a block (circle with R) increasing his accounts balance while referring to both the latest block of its account and the sender's block to consume. If the receiver does not publish a matching block, the transaction will stay in a pending state indefinitely. There is no way for the sender to revoke its transaction. This asynchronous

[29]https://github.com/nanocurrency/nano-docs/, Last accessed: 2023-03-15

procedure builds up a DAG, like shown in Figure 3.4, connecting all accounts that (implicitly) interacted with each other. A block always contains the full state of an account at a specific point in time [Nana; WYCX20].

Contrary to traditional blockchain-based systems, a block always only is a single transaction. Therefore in the context of Nano, both terms are frequently used interchangeably. More precisely: *block* describes the encoding of the transaction while *transaction* refers to the action [Nana].

The consensus scheme used in Nano is called "Open Representative Voting" (ORV). Similar to a PoS approach, each Nano account can choose a representative to vote on its behalf. Representatives are permanently running nodes that vote on transaction validity. Their voting weight is defined by the sum of account balances delegated to them. Delegating voting weight to representatives does not lock any funds. Accounts can change their representatives at any time. Representatives share their votes with directly connected peers and also forward votes from Principal Representatives (PRs). PRs are nano nodes with more than 0.1% of the *online voting weight*. This value is specified as the median of active representative weights sampled regularly across a moving 2-week window. The differentiation to normal representatives was made to reduce the bandwidth cost of broadcasting votes of all nodes, where many of them do not have a meaningful impact on the consensus due to their low voting weight. Received votes are summed up, compared against the available online voting weight and if a block has more than 50% it is confirmed. Additionally, a specific minimal online voting value must be reached. On the one hand, this helps in protecting the network in case some nodes are cut off, but on the other hand, the network stalls until enough nodes (voting weight) are back online. To "cement" a block, nodes store a block confirmation height for each account [Nana; Nanb].

There are no direct economic incentives for Representatives and also no staking rewards for delegators. However Nano's fast and feeless transaction can be seen as natural incentives to participate in the network. Because the network can be used without fees businesses are attracted to it. Additionally running an own Representative node is not resource-intensive because no PoW is required and has the benefit of direct network interaction without having to trust a third party. Trusted Representatives with many delegators can also be used as a form of marketing as they are shown in lists on network overview applications such as NanoLooker[30] [Nan20].

In order to mitigate the possibility of spamming transactions, a small PoW needs to be done for each. The difficulty for a receive transaction is lower than for sending. Since the PoW computation only needs data of the account's previous block, it can be precomputed and therefore does not impact normal usage [LeM18]. Currently, blake2b is used but research for a replacement with a memory-hard hash function like Equihash [BK21] or the Nano PoW[31] especially designed for it, is ongoing. Alternative non-PoW approaches

[30]https://nanolooker.com/representatives, Last accessed: 2023-03-15
[31]https://github.com/nanocurrency/nano-pow, Last accessed: 2023-03-15

like using time as a currency for rate limiting and quality of service based on PoS are also discussed[32].

Nano does not yet use any form of sharding but in a discussion, its founder stated that it would be good to have a solution ready before it is needed[33].

3.5.4 Discussion

In reviewing the existing literature within the field of DLTs with respect to scalability and interoperability, several topics present opportunities for further exploration and innovation. This discussion section will first give a summary on the related work described in the chapter and then provide insights into the potential avenues for research.

Summary

A common understanding regarding DLT scalability is that of "the Scalability Trilemma", saying that scalability, decentralization, and security cannot be fully achieved at the same time. However, with complex solutions that combine several approaches, reaching all three properties to a high degree may be possible. Using multiple methods further helps reducing possible congestions and avoids single points of failures.

Some general ways to improve the throughput are to be found in the consensus mechanism. By tweaking parameters, for example, the block size, a system can be adapted to changing requirements and reach a better performance. Nevertheless, such changes may lead to issues at other (unforeseen) parts of the DLT and increase computational resources for running a node. Other approaches are combining several consensus mechanisms and using different kinds of blocks or data structures within the same ledger.

A more radical approach regarding the change in the data structure is followed by DAG-based DLTs that replace the underlying blockchain with a DAG. This allows to attach blocks at multiple points in the graph in parallel instead of just at the end of the blockchain, resulting in higher transaction throughput. However, that shift requires different approaches to consensus algorithms and trade-offs regarding decentralization or security. For this work, we looked into the details and compared three DAG-based DLTs in Section 3.5: IOTA, Hedera, and Nano. Key takeaways at the time of writing are: IOTA still has a centralized component to secure the network, Hedera uses a patented algorithm and currently is not permissionless, Nano has a low adoption and uses a consensus scheme similar to (D)PoS with only a few nodes together having more than 51% voting weight. Recently IOTA announced the change to a validator committee[34] similar to Hedera and therefore will be more decentralized but at the same time permissioned.

Sharding can be counted as a complex scalability solution in DLTs. As there is not a single correct definition of sharding, it could mean one of the following: That incoming

[32] https://forum.nano.org/t/time-as-a-currency-pos4qos-pos-based-anti-spam-via-timestamping/1332, Last accessed: 2023-03-15

[33] https://forum.nano.org/t/horizontal-scaling/1502/34, Last accessed: 2023-03-15

[34] https://blog.iota.org/replacing-coordinator-with-validator-committee/, Last accessed: 2023-09-13

transactions are distributed over multiple groups of nodes and therefore each transaction is only validated by a subset of all nodes. Alternatively, the term also is used to describe that the DLT network is built with multiple interacting sub-networks. Sharding can be of a hierarchical or non-hierarchical structure and further categorized into homogeneous or heterogeneous sharding. An example of a hierarchical, homogeneous sharding is Ethereum 2.0 described in Section 3.2.3. Note that after working on that section both the naming[35] and the goals regarding sharding changed for Ethereum[36]. Instead of Ethereum 2.0, it will just be Ethereum (execution and consensus layer) and sharding will be dropped in favor of Layer 2 solutions.

There exist several categories of DLT interoperability solutions. In Section 3.3, we described Public Connectors which include notary schemes and relays, as well as the BoB concept. Notable work regarding relays are the ETH Relay, which follows an optimistic approach with a dispute period and Verilay which was built for Ethereum's PoS upgrade. Note that Verilay did not make it into the related work chapter but is described later on in Section 4.1.2.

Like the name BoB suggests, such systems can be seen as a network of efficiently connected blockchains. Usually they include a component that acts as point of connection for the other chains. To facilitate adoption usually resources such as reusable components and SDKs for data, network, consensus, incentive and smart contract layers are available. By that the effort to create a custom application-specific blockchain connected to the network is greatly reduced. Section 3.3.2 also gives an overview to two representatives of that technology: Polkadot and Cosmos. Both follow slightly different approaches regarding the trade-off between security and customization of chains. In place of various incompatible DLT projects there now are competing ecosystems of multiple chains. It remains to see which project will take the lead in terms of adoption.

Chapter 2 already gave an introduction to the intriguing field of ZKPs and their impact on DLTs. Although ZKPs, especially zk-SNARKs with their succinct proofs, bring many possibilities, it is important to consider the computational resources needed or proof creation. With increasing statement size the memory consumption can quickly reach limits on common end devices. Notwithstanding, the relatively novel technology is already in use in various DLT solutions. For example in the blockchain protocol Mina, zk-SNARKs are utilized to reduce the blockchain into a constant sized proof, while Zcash uses them to preserve privacy. Additionally zk-SNARKs have been used to create interoperability solutions, such as a relay and a cross-chain transfer protocol. The "zkRelay", employs zk-SNARKs to do the validation of blocks off-chain. The on-chain relay code only needs to verify the resulting proof.

Like previously mentioned, Ethereum dropped sharding from its roadmap, instead focusing on Layer 2 solutions. One of these Layer 2 scalability solutions, zk-rollups, employs zk-SNARKs to execute transactions in batches off-chain and therefore reduce the burden

[35]https://blog.ethereum.org/2022/01/24/the-great-eth2-renaming, Last accessed: 2023-09-13
[36]https://ethereum.org/en/roadmap/#what-about-sharding, Last accessed: 2023-09-13

on the mainchain by a huge factor. This decision for Ethereum brings much weight behind the future research on zk-SNARKs, especially for a general-purpose zero-knowledge EVM implementation.

In general, zk-SNARKs and comparable families of ZKPs are a heavily researched topic, already resulting in highly improved efficiency and easier, more universal, applicability. Besides proposing new proof schemes, making them available and comparable in actual implementations is essential. Even with ready-made libraries, existing DLT cannot directly take the full advantage, because widely-used cryptographic primitives, like SHA hash functions perform poorly when executed within ZKPs. Some existing hash functions like Pedersen are more suitable but research already led to promising novel hash constructions specifically designed for the application in ZKPs.

Potential research

One notable research gap concerns the creation of a relay for DAG-based DLTs. Many papers have focused on the development and analysis of DAG-based ledgers and there also is work describing various relay approaches for blockchain-based systems. However, to our knowledge, these two areas of interest have not been investigated together yet. A relay would enable interoperability in a way that, e.g., smart contracts executing on another ledger can act on transactions that occurred at the DAG-based source.

Another path for practical research lies in the exploration of the ISC framework. Because it was just released recently in an early development stage and is not available on the IOTA mainnet yet, limited theoretical and practical insights are available. Investigating the capabilities and limitations of IOTA's smart contract framework as well as the proposed IOTA ecosystem consisting of the mainnet, Shimmer, and the attached Layer 2 chains, could shed light on the potential for decentralized applications in DAG-based ledgers. Nevertheless, smart contracts are not executed directly on the Tangle but on specific smart contract blockchains anchored to the Tangle that can inter-operate with each other over it. That "Tangle of Blockchains" system loosely resembles a BoB scheme such as Polkadot. In the long term, the IF plans to integrate a general-purpose VM enabling Layer 1 smart contracts [IF23].

CHAPTER 4

Design

This chapter presents the requirements and design approaches for a relay scheme with a DAG-based source ledger. The overall design follows the concept of the existing state of the art relay solutions built for blockchain-based DLTs. Due to the great difference in consensus algorithms and inner workings of DAG-based projects, specific design decisions for a use with IOTA are made. However, the validation functionality is independent of the target DLT. The only requirement is the support for arbitrary computations ("smart contracts"). After explaining the design considerations, we describe the different steps involved in the relay process, including their states and preconditions.

4.1 Requirements/Design Goals

In general, the goal of relays is to make events, such as transactions of a source ledger available on a target ledger in a verifiable way without depending on any intermediaries. A relay scheme usually is built with two components: a relay contract deployed on the destination ledger and an off-chain client allowing to easily *relay* data from the source ledger to the contract. For this thesis, no off-chain client will be built. Nevertheless, when needed for the usage of the contract, extensions to the interface of existing source ledger node software will be provided. Furthermore, several test cases for the relay's core features will be specified.

4.1.1 Design Criteria

In their work on a PoS relay [WD22], Westerkamp and Diez list several design criteria identified in previous cross-chain and relay literature:

- Forkless: No fork of the source ledger or active cooperation of a subset of its nodes is needed to operate the relay.

- Trustless: The initial state of the relay can be verified by its users. From there on trust is derived only by validating the consensus of the source ledger. Therefore, no trusted intermediaries are required.
- Autonomous: The relay is autonomous, in the sense that all that is required for it to be used, are fully functioning source and target ledgers.
- Robust: The relay's state is preserved and in case no update occurs for a considerable time, catching up with the current state of the source ledger is technically and economically feasible.
- Corresponding: Only data that is valid according to the source ledger is processed. Furthermore, the relay contract includes functionality for validating inclusion proofs of transactions, states, and events occurring there.
- Lightweight: Any execution of the relay contract or client should be efficient regarding computation and memory resources. Update transactions at least adhere to the limitations imposed by the target ledger.

4.1.2 Relay Schemes for PoW and PoS

Existing work on relays has largely concentrated on blockchain-based DLTs using a consensus algorithm with PoW and the *longest chain* rule. However, new projects, as well as established ones, such as Ethereum, show the trend of shifting away from the resource-consuming PoW to using variants of the more efficient PoS. First efforts to follow this development in the research field of relays have been made by Westerkamp and Diez in *Verilay: A Verifiable Proof of Stake Chain Relay* [WD22], which includes a proof of concept implementation[1] for Ethereum 2.0.

Validation of PoW block headers can be done straightforwardly by recalculating and comparing the hash value. Additionally tracking temporary forks and locking blocks until enough have been added on top results in a usable relay. A consensus built around PoS involves validator committees that sign the blocks they deem valid. In the case of Ethereum's Beacon Chain, a separate relay committee exists, which includes more validators and stays unchanged longer than the usual committee. By storing (or resubmitting) the public keys of the current and next relay committee's participants in the relay contract it is possible to validate if a block's aggregated signature reaches the minimal threshold needed. The relay committees, in turn, can be validated because every block references the current and the next one. Therefore, the relay contract needs at least one block of each sync committee period (256 epochs, roughly equal to 27 hours[2]) to validate the transition. Contrary to Bitcoin or Ethereum's PoW blocks, blocks of Ethereum's Beacon Chain include Merkle roots referencing the whole history. This avoids

[1] https://github.com/MaximilianDiez/PoSChainRelay, Last accessed: 2023-03-15

[2] https://github.com/ethereum/annotated-spec/blob/master/altair/beacon-chain.md#misc-1, Last accessed: 2023-03-15

the need to include all blocks (headers) in the relay, or in other words, it decouples blocks from each other in the context of the relay.

4.1.3 Relay Scheme for DAG-based DLTs on the Example of IOTA

The common requirements for a relay solution described with the specific example of IOTA, also depicted in Figure 4.1, are as follows: A source ledger (IOTA) is observed by off-chain clients that submit ledger data (milestones) to a smart contract (relay contract) at the destination ledger. After validating the submitted data (milestone) the contract stores the relevant data (Merkle root hash etc.) allowing it to answer inquiries from clients if a specific message (transaction) is included. For that, off-chain clients need to build a proof (Merkle proof) with data they can get from the source ledger and submit it to the relay contract for verification. If it passes the verification the relay contract could, for example, forward this information to another contract that acts on this information.

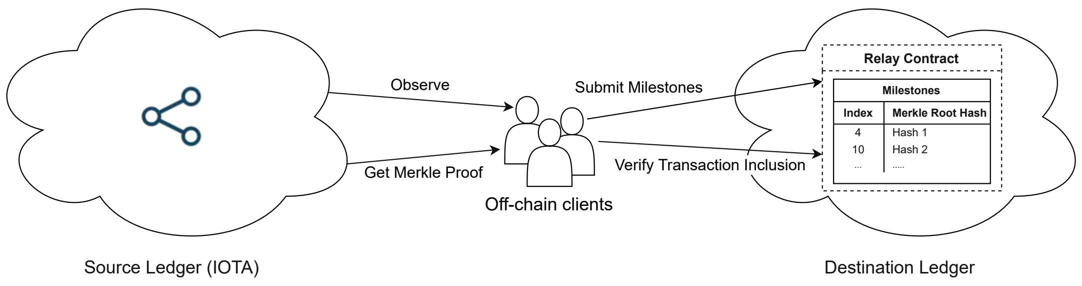


Figure 4.1: The IOTA Relay Scheme. Adapted from [FSS+20].

DAG-based DLTs not only differ in the way the consensus works but of course also in their data structures, with some not having a directly comparable concept of blocks like in a blockchain. As described in Section 3.5.1, IOTA builds a DAG out of messages. The Coordinator periodically issues a message with a signed payload called *milestone*. Like normal messages, milestones attach to multiple tips of the Tangle but also confirm the messages they directly or indirectly reference.

The set of confirmed messages can still include conflicts. Nodes have to apply a deterministic ordering and only apply the first of the conflicting messages, while the others are ignored. The order is determined by a post-order depth-first search. That process is called "White Flag Ordering" and is specified in a Tangle Improvement Proposal (TIP)[3]. The result of the White Flag Ordering is a list of all nonconflicting messages having a transaction payload. Put differently, it comprises all value transfers which are not double-spends. As illustrated in Figure 4.2, a Merkle root hash calculated based on that result is included in milestones. Therefore, nodes can validate the milestone by comparing the Merkle root hash against the hash obtained from performing the White Flag Ordering on their local state. Furthermore, inclusion proofs for transactions can also be validated

[3]https://github.com/iotaledger/tips/blob/main/tips/TIP-0002/tip-0002.md, Last accessed: 2023-03-15

using the Merkle root hash, but the same is not possible for other message types, as they are not included in that Merkle tree. The keys used to generate the milestone's signatures depend on the milestone index and are included in the configuration of the node software.

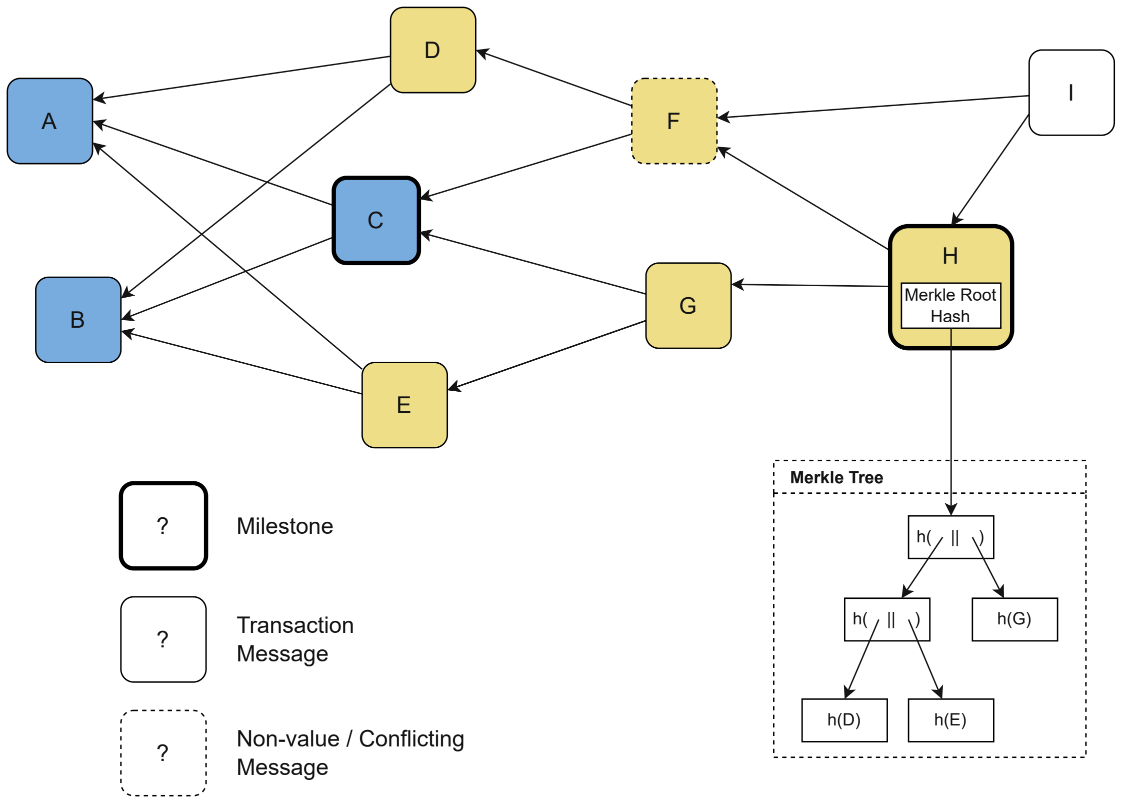


Figure 4.2: Example of IOTA Milestone with Merkle Tree based on White Flag Ordering. Adapted from [FSS+20; Mar21].

Albeit not that well thought out (or suitable for a relay) compared to the approach taken by Ethereum, there are similarities. Milestones can be seen as a kind of decoupled block because they include transactions via the Merkle root hash and do not solely define confirmed transactions by referencing them in the Tangle. Nevertheless, the signatures of a milestone are less trustworthy than the aggregated signatures of Ethereum's sync committee. The issue is that the Coordinator, its signing nodes, and the keys are under the control of the IOTA Foundation and there is no protocol to execute a trusted transition of the keys in use. Node operators need to update their configuration with newly published public keys. An expert of the IF confirmed via email that there is no on-chain protocol to avoid or secure that process.

Nodes in the IOTA network validate milestones by recalculating the Merkle root hash using their internal state of the Tangle and checking the result against the hash included in the milestone [Wel21]. There is no additional signature or other indication available that shows a node's consent regarding a milestone. Thus, in a relay contract that does not mirror the Tangle with all its messages, it is impossible to distinguish a milestone actually included in the network from a forged milestone that got signed with the real keys. This puts even more trust into the Coordinator but if the private keys to create valid milestone signatures are compromised, the whole IOTA network is at risk anyway.

In the following, the steps for relaying IOTA milestones and proofing the inclusion of transactions are outlined.

4.1.4 Initialization

A node trying to synchronize with the Tangle needs to be bootstrapped with a trusted snapshot file containing an initial state up to a specific point. These snapshots are provided regularly by the IF[4]. Full snapshots are published every 1-2 days with the time in between filled by delta snapshots at roughly two-hourly intervals. The snapshot file format is defined in a TIP[5]. It includes UTXOs and milestone data to allow the ledger state to be rebuilt. Additionally, nodes need the Coordinator's public keys in order to verify incoming milestones. Each key is only valid for a specific message index interval but there are always two keys with overlapping intervals in use to sign milestones. Both the keys and their applicable intervals are defined in the node's configuration file[6].

Because the relay only uses milestone data, just the public keys are needed for the initialization of the contract. Users need to verify that the keys used in the relay contract match the actual Coordinator keys. To avoid other unexpected behavior and increase trust, it is also recommended to verify the contract logic itself. For example, there should be no way to change the keys or intervals already added. Should the case occur that a change is needed, a new contract needs to be deployed. As Westerkamp and Diez [WD22] note, the average user lacks the skills to verify the relay contract on his own and therefore has to build his trust on the public perception and possibly audits performed by trusted third parties.

4.1.5 Relay Update

As soon as the relay contract is successfully deployed to the network, any user connected to the target ledger can update it. No trust in the relaying user is required because the relay contract validates the received milestones. Relays also need to make sure the data, e.g., blocks, they operate on is "finalized", i.e., non-reversible at least with a sufficient probability. For traditional blockchains operating on PoW, this is often done by waiting for a specific amount of new blocks confirming the previous ones, making the handling of

[4]IOTA snapshot files https://chrysalis-dbfiles.iota.org/?prefix=snapshots/hornet/, Last accessed: 2023-03-15

[5]Local Snapshot File Format https://github.com/iotaledger/tips/blob/main/tips/TIP-0009/tip-0009.md, Last accessed: 2023-03-15

[6]Hornet node configuration file https://github.com/gohornet/hornet/blob/mainnet/config.json, Last accessed: 2023-03-15

temporary chain forks in the relay mandatory. Contrary, PoS ledgers reach the finalized state by means of voting. Therefore, after a block reached the targeted majority, it is finalized and guaranteed to be part of the main chain. Because enough votes may not be instantly available but are part of successive blocks, the finalization could occur with a delay [WD22]. In place of votes for a block, in IOTA there are milestones confirming transactions. For the use cases of a relay, it is enough to only store the milestone index and the Merkle root hash instead of the whole milestone.

The data from the source ledger can either be obtained directly by running an own node or by utilizing APIs of public nodes. Usually, calls to fetch data via HTTP are available. In some cases, node software also provides means to subscribe to specific events one wants to receive messages for, commonly called publish/subscribe (pub/sub) pattern. IOTA nodes provide both, a "Node Event API" using the lightweight MQTT protocol[7] and a REST API[8].

4.2 Milestone Validation

Before validating the milestone data itself, the message encapsulating it can also be checked. TIP-6 defines several criteria for a syntactical validation of messages in general, while TIP-8 contains details about the milestone payload:

- Maximum size: messages must not exceed 32 KiB. This limit should be checked at the contract before any further processing in order to avoid malicious attempts, such as bloating the contract's data.

- Parents: at least 1 but not more than 8 parent message ids in lexicographical order need to be included. The milestone payload of the message has to include the same parent data. The reasoning behind this is to prevent the attachment of the same milestone payload at different places in the Tangle. This makes the milestone independent from the message itself, allowing simpler processing. It also ensures the actual position of the milestone message is consistent with milestone data like the Merkle root hash which is calculated based on the location and the Tangle topology.

- Nonce: valid solution to the PoW defined in TIP-12.

- No excessive bytes after parsing all defined message fields.

- Payload type: each payload starts with an integer identifying its type. For milestone payloads, this is the number 1. This is important for deserializing the data from raw bytes, e.g. when received as a parameter in the contract.

[7]https://wiki.iota.org/iota.rs/examples/mqtt, Last accessed: 2023-03-15
[8]https://wiki.iota.org/hornet/references/api_reference, Last accessed: 2023-03-15

- Keys: The public keys for the signature of a milestone are also included in its payload, even though they are not used for verification in the node software. Instead, one or more keys stored in the configuration file for the appropriate index are used. The keys must be unique, sorted lexicographic, and match the given count. Additionally, there is a lower bound for the amount of keys/signatures specified in the configuration.
- Signatures: Must match the amount and order of the keys.

The most important thing is however the verification of the signature. It is created by signing the BLAKE2b-256 hash of the milestone payload with an Ed25519 signature scheme (RFC 8032). In practice, IOTA currently uses two independent signatures per milestone. The Coordinator only assembles the milestone while the actual signing is done by separate signature providers. By that, there is no central point holding all the key material which makes it more difficult for potential attackers to acquire all keys needed to create malicious milestones that pass verification.

4.3 Message Inclusion Proof

Just block or milestone data of the source ledger stored in a smart contract on the target ledger alone is not enough to fulfil a relay's function. A way to prove that a transaction has happened, meaning it is included in a valid milestone, is necessary. Milestones include the root hash of a Merkle tree built out of transactions like previously described in Section 4.1. Therefore, it is possible to provide inclusion proofs for any transaction that is part of it. The relay contract can verify the proof and if correct, for example, forward details of the proven transaction to another smart contract that acts on this information.

For building a Merkle proof, two things are needed: First, the data to prove inclusion for, and second, the sibling hashes along the path from the data to the Merkle root (also called Merkle audit path). That is enough to recalculate the Merkle root to compare against the Merkle root hash of the milestone which previously got stored in the contract. If they match, the proof is valid and the transaction, therefore, is part of the milestone.

IOTA uses binary Merkle trees with BLAKE2b-256 as hash function like specified in TIP-4 [Wel21]. Although there is no reference in the TIP it seems to be heavily based on [RFC6962]. The input for creating one is a list of messageIds as binary data of messages determined with the White Flag Ordering for the respective milestone. In Figure 4.2, the construction of the Merkle tree hashes is displayed in a simplified way not differing between leaf and node hashing. This would allow replacing two leaves with one having the hash result of them as value. Therefore, different trees with the same root hash could be created. To mitigate this security issue and provide "second preimage resistance", leaves and nodes get a different prefix when hashing.

A more formal definition of the Merkle Tree Hash (MTH) is given in the recursive Algorithm 4.1. The result of an empty input list is the hash of an empty byte string (line 3). Line 6 gives the hash for a leaf, i.e., the hash for the only remaining element after recursively splitting the input list. Note the prefix of *0x00* for the hash input, contrary to line 9 which prepends *0x01* for the node hash in order to achieve the previously mentioned second preimage resistance. The value k (line 8) defines how the input is split into subtrees and by that also the structure of the tree based on the length of the input list.

Algorithm 4.1: Merkle Tree Hash

```
    Input: An ordered list of $n$ input strings: $D_n = \{d_1, d_2, ..., d_n\}$
    Output: The Merkle Tree Hash of $D_n$
 1  Function MTH($D_n$)
 2      if $n = 0$ then
 3          return BLAKE2()
 4      end
 5      if $n = 1$ then
 6          return BLAKE2(0x00 || $d_1$)
 7      end
 8      Let $k$ be the largest power of two less than $n$, i.e., $k < n \leq 2k$
 9      return BLAKE2(0x01 || MTH($\{d_1, ..., d_k\}$) || MTH($\{d_{k+1}, ..., d_n\}$))
10  end
```

CHAPTER 5

Implementation

This chapter serves as a detailed exploration of the implementation aspects for the solution design already described in Chapter 4. First, it provides insights into the technology stack, including programming languages and execution environments as well as the various frameworks and tools utilized. Next, the Relay Contract's functionality and testing approach are discussed. The chapter concludes with a description of the extended node interfaces and multiple zk-SNARK libraries.

5.1 Technology Stack

5.1.1 Programming Languages and Execution Environments

The most popular programming language for implementing smart contracts most probably is *Solidity*. It was originally designed by Gavin Wood in 2014[1] and since then was heavily used for smart contract development in Ethereum and in general for any DLT using a compatible execution engine, also denoted as "virtual machine" (VM). Through the years, a vast development ecosystem formed around Solidity and the EVM. For this reason, many projects, such as the Binance Smart Chain[2], chose to build compatible systems, so that they can leverage the existing toolchains and make it easy for smart contracts to be deployed there as well. Before deployment to the DLT network, smart contracts written in Solidity or alternative languages are compiled into EVM bytecode that can be executed by a node through the EVM.

Although Solidity was specifically built for smart contracts and has gained much adoption, there still is the urge to use other established general-purpose programming languages. In recent years, WebAssembly (Wasm) emerged, originally, as a binary format and runtime for Web applications. Its standardized feature set is supported nearly completely by

[1]https://polkadot.network/gavin-wood/, Last accessed: 2023-03-15
[2]https://github.com/bnb-chain/bsc, Last accessed: 2023-03-15

all major browser engines. Wasm bytecode is intended to be used as a compilation target for modern programming languages. Studies show that Wasm can reach a better performance than Javascript when executed in a browser. Wasm is not only intended to be run in browsers but in general is built for memory-safe, sandboxed environments. Therefore, it quickly gathered interest from the DLT community as an alternative to the EVM [ZWW+21]. The main advantage of Wasm is support from a wide array of programming languages, for instance, Rust, Go, and C#[3]. Additionally, the general assumption is that it can perform better than code executed by the EVM.
Zheng et al. [ZWW+21] conducted a first comparative analysis between several Wasm and EVM engines using diverse benchmarks. They found that the performance greatly differs depending on the concrete implementation and benchmark used. However, the following general observations could be made: Wasm uses 32- or 64-bit words while the EVM runs with 256. Therefore, benchmarks utilizing the full 256 bit require an overhead when executed on Wasm. The Ethereum flavored *Ewasm* is a restricted subset of Wasm with interfaces to interact with the Ethereum environment and includes gas metering. Compared to a standalone Wasm engine, these additions make a noteworthy impact on execution time. In summary, standalone Wasm VMs perform better than EVMs but DLT-specific Wasm engines are slower in most cases.

IOTA's Wasm VM is built on top of *wasmtime* [IF22a]. The benchmarks in [ZWW+21] show that wasmtime is the second-fastest engine in many cases. Interestingly, the performance for BLAKE2b is weak, with an execution time higher by a factor of 50 compared to the best engine. This is especially noteworthy as IOTA primarily uses that hash function. Nevertheless, this may have already been improved as wasmtime v0.15.0 used in the benchmarks is from April 2020. Running the benchmark again with the latest version (v0.35.1 released in March 2022) unfortunately is not possible because the link to the source code is missing from the document.

To write a smart contract, using a programming language allowing to target Wasm is not enough. Additionally, some kind of library or SDK of the target DLT is required. Often programming models similar to Solidity are used.

There are several options for implementing the IOTA Relay design described in Chapter 4. In theory, IOTA smart contracts also include support for the EVM and one could therefore benefit from Solidity and its tooling. However, it currently is only an experimental feature with several limitations. Furthermore, the goal for creating the relay is to re-use existing IOTA libraries to reduce possible implementation errors as well as to increase maintainability, performance, and robustness. Because the IOTA Foundation uses Rust as its main language for current and new IOTA libraries, the choice seems natural for the relay implementation. Additionally, it also was the first language to be used for the smart contract library due to its stable Wasm support. Later implementations of this library for Go and Typescript were made available as well [IF22a].

[3]Status of Wasm support of multiple programming languages https://github.com/appcypher/awesome-wasm-langs, Last accessed: 2023-03-15

In the discussion of whether to support Solidity or Rust for smart contracts one reason for Rust is to get developers to really think about the implementation and avoid copy-pasting of existing Solidity code [Mac22]. Moreover, Rust is a very popular general-purpose programming language opening up the field of smart contract development to a wide audience. Then again an inexperienced Rust developer could also produce insecure and inefficient code. Ways to mitigate these issues are thorough reviews of new contracts by the more experienced community and providing easy-to-use APIs with good documentation.

During work on this thesis, we sometimes had difficulties due to having no previous experience with Rust or Go despite the fact that only a subset of language features are needed for IOTA smart contracts. Using Solidity, on the one hand, may have been easier in general because we have worked with it in the past already but on the other hand, it would increase the complexity through the need to re-implement procedures otherwise readily available in libraries.

5.1.2 IOTA Frameworks & Tools

This subsection introduces several libraries and components developed by the IOTA Foundation or the community to allow for easier implementation of related projects and smart contracts.

Hornet

IOTA has two officially supported node implementations. Hornet is written in Go and is currently the recommended node software because it includes features not available in the alternative. It is maintained by the community and supported by the IOTA Ecosystem Development Fund.

Bee

Bee is developed by the IF directly and follows a modular approach by splitting its components into separate extendable libraries to allow re-usage for client software and other projects. At the same time, it will act as the reference implementation for several algorithms and data structures used in IOTA. Due to its performance and memory safety guarantees, the IF chose Rust as their main programming language for most new projects, including Bee. Other languages, such as Python and Java, are supported through bindings to the Rust libraries. However native implementations with reduced functionality for C, Go, and TypeScript, are also available. The reasoning is that C is needed for embedded devices and the others were in use before the Rust-based library was finished [IF21c].

Solo

Smart contract development without a local development environment or framework easing the task of deployment and testing would be laborious. With Solo, developers can validate their smart contracts with tests written in Go, without the need to deploy them on a full testnet. Solo takes care of starting a virtual chain and deploying the contract. Furthermore, it allows assertions on the execution of a smart contract call. For interacting with the contract in tests, the same interfaces generated by the schema tool for the implementation are used.

WasmLib

Every smart contract needs a secure way to execute in isolation and work with its state in a deterministic way. To support a wide range of programming languages, IOTA chose to use Wasm code as an intermediate representation which then can be executed within an appropriate VM. A VM isolates the contract execution from the underlying host system and ensures software portability. Wasm code running inside the VM has its own memory space and by design is isolated from the outside except for the access to external functionality needed for its execution. The ISCP sandbox API can be used by any type of VM and provides generic functionality to store, access, and modify the state of the smart contract. The use of the sandbox API is enabled by embedding the so-called “WasmLib” into each compiled smart contract. This is needed to make the communication between the different memory spaces of the smart contract and host possible [IF22a].

Schema Tool

There are many use-cases for smart contracts but usually, they involve handling assets of some kind. For that reason, it is especially important that they are implemented in a robust and secure way. Past experience shows that this is not an easy task and even slight mistakes can have grave consequences like the loss of all assets held by the contract. Recurring tasks while developing smart contracts, among others, include checking for access rights and verifying the correctness of parameters. To reduce possible mistakes and help with the implementation of smart contracts, the IF created the schema tool which generates interfaces for functions, parameters, results, and state, based on a schema definition file. One advantage is that the generated code uses strict type-checking at compile time. Furthermore, it can output code in different programming languages making it possible to program the contract in Rust but still easily test it with the Go-based Solo framework.

Defining the contract for the Schema Tool is done in a YAML (default), as seen in Listing 5.1, or JSON file. Besides fields for the name and description of the contract, the schema is divided into several sections.

events. To allow clients to keep up with the state of the contract without polling, the ISCP provides an event interface. However, this just supports arbitrary strings, making it inconsistent and cumbersome to work with. That is why the schema tool can generate client and contract code handling the encoding and decoding of events defined in a structured way inside the schema file. Every event is logged and therefore indirectly part of the state of the contract. If the data is not needed in the contract itself using events is an alternative to storing data directly through the contract. This approach can be used to reduce transaction costs because events are cheaper than actual data access.

structs. This section is used to define structured data types with multiple fields that can have different types and are accessible by their name. An example is, storing different (meta-)data of a milestone in a grouped and structured way. Again, the schema tool helps by generating code for accessing such a struct and also in the background transforming it to a single byte array for storing.

typedefs. Allows defining a type alias. This is mainly used to build nested arrays or similar which is not directly allowed otherwise.

state. The state of IOTA smart contracts is a simple key/value store using raw bytes. The fields defined in this schema section specify the variables that will be stored in the state. Type-safe handling is enabled by the schema tool-generated code. IOTA provides some predefined types through the WasmLib that can be used directly for the schema definition. On the one hand, there are basic types like (unsigned) integers, String, Bytes, and on the other hand smart contract-specific types, such as Address and Hash.

funcs. Each entry in this section defines a function with an access identifier, parameters, and results. Funcs are allowed to modify the smart contract state. Code generated for variables defined in the state section includes two interfaces, one for mutable access and another immutable one for the use with views. The method stubs created for the contract implementation do not directly include these interfaces as parameters. Instead, the access works via an extra context object specific to each function. Additionally, there is a common context object providing utility functions for the smart contract, for instance, calling another contract, creating an event, or logging.

views. Contrary to funcs, views only retrieve information from the state but are not able to change it. Therefore, no tokens can be transferred with views and calling non-view funcs is not possible. An example is the retrieval of saved milestone information.

5.2 Relay Components

Based on the considerations and the tech stack explained in Section 5.1 we can now discuss our own relay schema and its components. Afterward, the relay functionality is described in Section 5.2.2 throughout Section 5.2.4 and some thoughts on testing approaches are given in Section 5.2.5.

An overview of the extensive interplay of different systems and tools is given in Figure 5.1. The relay contract, as the central resource, has the most connections to other parts of the implementation. Several libraries are used for its implementation, the schema tool for its specification and Wasm tooling for compilation and execution. The programming languages and protocols in use are given in square brackets, showing the multi-language interfaces generated by the schema tool and the test components written in Go. On a user- or client-side the main components are the extended node (Section 5.3) for fetching proof data and the smart contract-capable node to call the relay.

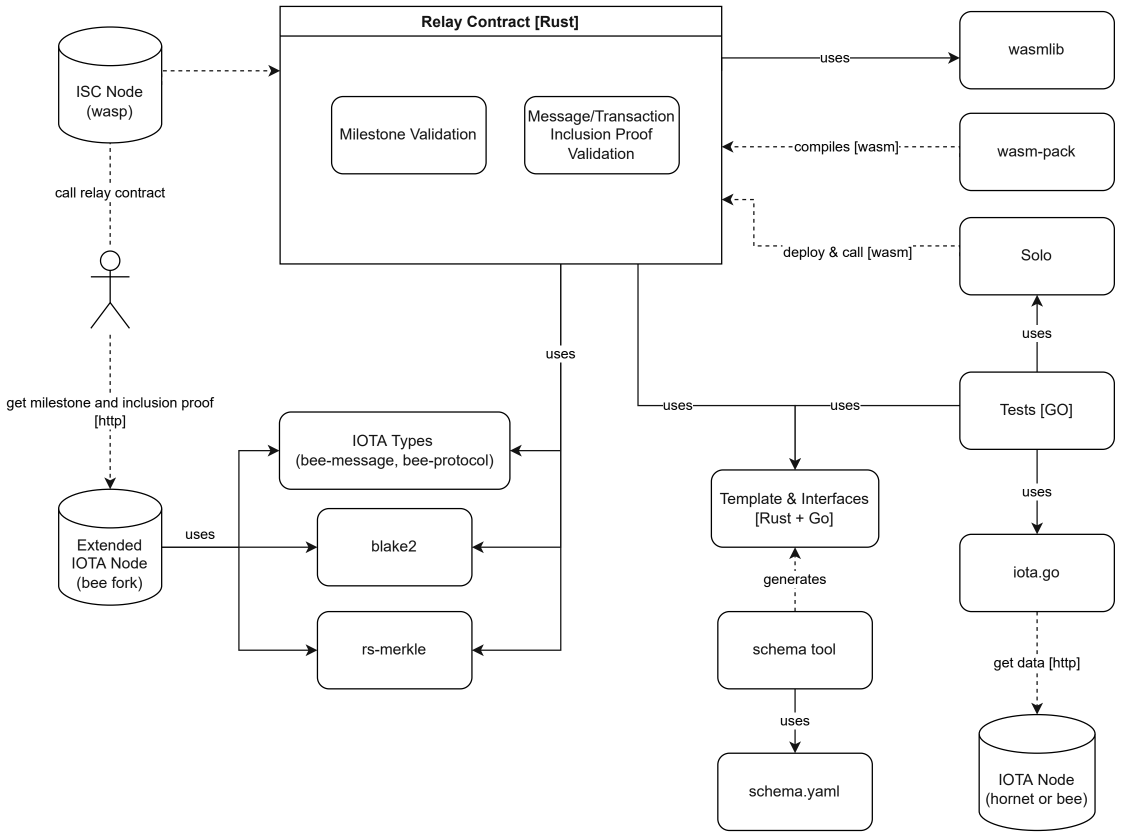


Figure 5.1: The IOTA Relay Implementation Components.

5.2.1 Contract Schema

The contract schema, shown in Listing 5.1, defines the interfaces for the implementation of the relay contract. Besides specifying the function names, parameters, and results also the contract state storage is declared.

There are several built-in data types available, such as *Bytes*, *Uint32*, *Bool* and *Hash*, which eases the handling of hashes used for IOTA. The Hash type is used as part of the *MilestoneData* struct to store the Merkle Tree root hash (line 6). In the contract state, the map (line 9) resolves milestone indices to their *MilestoneData*.

The function *addMilestone* (line 12) accepts the milestone payload serialized as Bytes.

For transaction verification, the parameters (line 17) include the byte representation of the Merkle proof, the index of the message to prove, as well as the message data itself, and the milestone index. The read-only function *getMilestone* (line 27) returns the data for a given milestone index.

Listing 5.1: Shortened schema.yaml defining the relay contract.

```yaml
 1 name: IOTA Relay
 2 description: Validate milestones and verify transaction inclusion
 3 
 4 structs:
 5   MilestoneData:
 6     merkleRoot: Hash
 7 
 8 state:
 9   milestones: map[Uint32]MilestoneData
10 
11 funcs:
12   addMilestone:
13     params:
14       payload: Bytes
15 
16   verifyTransaction:
17     params:
18       proofBytes: Bytes
19       indexToProve: Uint32
20       leavesLength: Uint32
21       msgData: Bytes
22       milestoneIndex: Uint32
23     results:
24       valid: Bool
25 
26 views:
27   getMilestone:
28     params:
29       index: Uint32
30     results:
31       milestone: MilestoneData
```

5.2.2 Relay Update

The first thing to do for the relay when receiving an update is to unpack the message binary data. In order to reduce the data transferred and therefore possible transaction costs, just the milestone payload itself is used. Milestone payloads in IOTA are identified by “1” as a 32-bit integer in the beginning. However, when directly using the unpack method of *MilestonePayload* in bee’s codebase, this identifier is not read but set statically as it is assumed to have been read previously to identify the type of the payload in the first place. When using an IOTA client library to get milestone data, it will usually include that identifier. This is also the case for iota.go which is used to fetch data for the tests done with Solo. Hence, the first 4 bytes need to be removed, ignored, or handled by a more general method that reads the identifier before deserializing further. After validating the signature, the milestone index together with at least its Merkle root hash is stored in a map. To allow for easy adaption of the contract in case more milestone details are needed, a struct is used as the value type of the map. Candidates for storing alongside the hash value in the struct or as a separate field are milestone timestamp, address of the submitter, and the latest milestone index.

5.2.3 Milestone Validation

Validating a milestone signature requires matching public keys. The keys need to be stored at the contract and it should not be possible to change them without deploying another version of the whole contract. Thus trust in the relay can be established by checking the stored keys and verifying that no method exists that allows a malicious party to change them. A downside of this approach however is that from time to time a new contract is necessary because the public keys are only valid until a specific milestone index. Furthermore, the minimum number of valid signatures a milestone needs are defined in the contract. With that number and the given public keys, a key manager instance is created which is used to return fitting keys to validate a milestone with a specific index.

5.2.4 Transaction Inclusion Proof

The requirements for the implementation of transaction inclusion proofs and the Merkle tree can be derived from Section 4.3:

- Merkle proof/audit path: It must be possible to create an audit path (inclusion proof) for a specific message included in the Merkle tree. For transferring this proof to the relay contract and verifying it there, functionality to serialize/deserialize to/from bytes is needed.
- Hash function: The hash function utilized while building the Merkle tree needs to be BLAKE2b-256. Existing libraries therefore either need to already include it or allow for defining an own function to be used.

- TIP/RFC compatibility: The structure of the Merkle tree as well as the prefixes for hashing leaf and node have to match the specification of the TIP/RFC. Therefore, the correct prefixes for hashing leaves and nodes have to be included or a hook to define them needs to be available.

Possible options to fulfill them are reusing IOTA implementation from the Bee or Hornet node software, utilizing other existing Rust code preferably in form of a crate (library), or building it from the ground up. The implementation that is part of Bee unfortunately does not include any means of creating or verifying proofs, nor has any serialization capabilities. Several Rust crates provide Merkle trees with differing features and goals. Because the relay contract tests are written in Go, it would be an advantage to have a compatible library, ideally from the same authors, for it as well.

- rs-merkle[4]: This Rust crate, according to its author is “the most advanced Merkle tree library for Rust”. It provides good documentation as well as all needed features and is actively maintained.
- merkletree-rs[5]: In addition to this Rust crate, there also is a compatible Go implementation[6] from the same author.
- merkletree[7]: This implementation is actively maintained by the Filecoin project. There are recent updates to the repository but during the initial evaluation of libraries, the latest available crate was from 2020.
- tendermint: tendermint is a BFT-based consensus implementation used for Cosmos and written in Go that includes suitable Merkle tree code[8]. There also is a project working on providing a Rust port[9].

After an initial selection with an emphasis on the availability as stand-alone Rust crate, project activity, documentation, and actual usage, two libraries, *rs-merkle* and *merkletree*, underwent practical experimentation to adapt them to the given requirements.

Both libraries allow relatively easy integration of arbitrary hash functions and support proof generation. For BLAKE2b-256, the crate *blake2*[10] was used. The advantage of *merkletree* is that it already implements the hashing with prefixes compliant to the TIP. However, the number of input leaves must be a power of two. Therefore, it is not flexible enough for the variable number of messages included in IOTA milestones.

[4]https://github.com/antouhou/rs-merkle, Last accessed: 2023-03-15
[5]https://github.com/arnaucube/merkletree-rs, Last accessed: 2023-03-15
[6]https://github.com/arnaucube/go-merkletree-old, Last accessed: 2023-03-15
[7]https://github.com/filecoin-project/merkletree, Last accessed: 2023-03-15
[8]https://github.com/tendermint/tendermint/tree/master/crypto/merkle, Last accessed: 2023-03-15
[9]https://github.com/informalsystems/tendermint-rs/blob/master/tendermint/src/merkle.rs, Last accessed: 2023-03-15
[10]https://github.com/RustCrypto/hashes/tree/master/blake2, Last accessed: 2023-03-15

In contrast, *rs-merkle* does not use prefixes but allows any input size. Adding the needed prefixes before hashing a leaf or node requires fewer and less extensive changes. Furthermore, this crate also comes with inbuilt serialization capabilities. For that reason, it was chosen for the implementation of the relay contract.

5.2.5 Testing

As already explained in Section 5.1.2, tests for IOTA smart contracts are written in Go with the Solo framework, which employs the generic Go package "testing". It automatically deploys the previously built Wasm file of the contract. To test the validation of milestones as well as the inclusion of messages, test data is needed.

One way is to use existing data from the IOTA test or mainnet. With the iota.go client library, recent milestones and messages can be fetched and serialized into bytes to prepare them for sending to the contract. However, that approach to dynamically load data from an IOTA node is not fully suitable for testing as it breaks with some best practices. It introduces a dependency on external resources and makes them nondeterministic if not always the same data is used.

Of course, it is possible to always request specific messages but IOTA nodes do not store them indefinitely. Therefore, it is advisable to only do so during the early development of the proof of concept code, when a quick and flexible way to run contract functions is needed, like in the context of this work. Later on, for the stable test cases and evaluation, selected data should be saved to disk or hard-coded into the tests, to guarantee consistent results. Alternatives are querying a permanode that stores all data, using an own node that is properly configured and holds all necessary data, or to directly make use of the database and snapshot files provided by the IF[11].

5.3 Node Interfaces

This section explains additions made to the API of the IOTA node software. They expose the data needed to use the relay functions.

Milestone Included Messages

To check at the relay contract if a given transaction is in included in a milestone a way to first build the Merkle proof for that procedure is required. This Merkle proof path can only be built with the knowledge of the transaction data or intermediary hashes along with it. Although IOTA nodes implement an API enabling queries for messages and their payloads, there is no functionality to fetch all messages that are included in a milestone. Therefore, we added another endpoint for this use case. It is based on a modified version of the DFS search/traversal function implementation for the White Flag

[11]https://chrysalis-dbfiles.iota.org/, Last accessed: 2023-03-15

Ordering found in *bee-ledger*[12]. The main differences are that the messages in question are already applied and only messages referenced by the given milestone index should be included. Additionally, messages having the conflict flag set or that do not include a transaction payload have to be ignored.

Transaction Inclusion Proof

Instead of just fetching all messages included in a milestone, the node software was also extended by an endpoint, that given a message id, provides the whole message data, the id of the milestone including it and the respective proof. The proof is sent as base64 encoded bytes while the rest of the payload is in JSON format.

5.4 zk-SNARK Libraries

Initially, a possible idea for implementing the IOTA relay was to utilize zk-SNARKs similar to zkRelay. Therefore at a relative early stage of this work a research regarding possible zk-SNARK programming libraries was conducted. Although not used in the actual implementation the following overview is kept in order to serve as a starting point for future work.

5.4.1 ZoKrates

ZoKrates [ET18; ZoK] is not only a library implementing a proof scheme but a full-featured toolbox helping to build DApps with zk-SNARKs. It uses its own domain-specific language, including a standard library of various optimized cryptographic components, for defining proof statements and a compiler to generate proofs for specific inputs. Furthermore, it can create Solidity code capable of proof verification. This code is ready to be deployed as Ethereum smart contract.

To represent programs "ZIR", ZoKrates' own intermediate representation format, is used. It is closely related to R1CS but can also include directives for witness derivability, meaning that it allows calculating all variables from inputs or intermediate results. Because R1CS constraints often involve large numbers, ZIR also uses an isomorphism to shorten representations and therefore improve readability.

ZoKrates does not implement proving schemes itself but employs multiple backends (bellman, libsnark, ark) supporting different ones. Besides two schemes by Groth (one with Maller) [GM17; Gro16] and "Pinnochio" [PGHR13], recently the universal and updatable "Marlin" [CHM+20] was added.

[12] https://github.com/iotaledger/bee/blob/e836fde6de/bee-ledger/src/workers/consensus/white_flag.rs #L207, Last accessed: 2023-03-15

libsnark

libsnark is one of the backends used for ZoKrates. It is written in C++ and includes implementations using R1CS as well as other languages that are required for specific proof systems. Zcash initially used the "Pinnochio" scheme implemented in a fork of libsnark [Zcac].

bellman

bellman, a library implemented in Rust, is another backend used in ZoKrates. The only proving system included is [Gro16].

Since the update called *Sapling* in 2018, Zcash employs bellman as zk-SNARK library [Zcac]. Forks of this project are used in various projects. For instance "bellperson" of filecoin[13] added assembly-level optimizations and GPU parallel acceleration for multiple essential algorithms. The bellman community edition by matter-labs enables a multicore feature by default and supports the BN256 curve for compatibility with Ethereum. Future plans include the additions of SONIC [MBKM19] and another scheme by Groth and Maller [GM17].

Ark

Ark (or arkworks)[14] is a collection of multiple separate Rust libraries (called crates) for developing with zk-SNARKs. It includes implementations of cryptographic primitives, R1CS constraints, elliptic curves and many more. ZoKrates supports the schemes [GM17] and Marlin [CHM+20] via the ark backend, although [Gro16] would also be available. The ZoKrates documentation [ZoK] does not explicitly state that ark is used for Marlin but the source code[15] references it. As noted in the Readme files all three proof scheme implementations are academic prototypes, have not received careful code reviews and therefore are not production ready. Additionally, the recursive verification feature[16] is also based on ark libraries.

5.4.2 gnark

gnark [Con21] is another open-source library for zk-SNARKs, as of now supporting the [Gro16] and experimentally[17] the PlonK [GWC19] scheme.

The main incentives for gnark were the often given statements that ZKPs are complicated, hard to use and perform slow. In contrast to bellman it is written in Go and according to

[13]https://github.com/filecoin-project/bellperson, Last accessed: 2023-03-15
[14]https://github.com/arkworks-rs, Last accessed: 2023-03-15
[15]ZoKrates code with ark_marlin reference https://github.com/Zokrates/ZoKrates/blob/develop/zokrates_ark/src/marlin.rs, Last accessed: 2023-03-15
[16]Pull request for recursive verification https://github.com/Zokrates/ZoKrates/pull/918, Last accessed: 2023-03-15
[17]https://github.com/ConsenSys/gnark/blob/master/CHANGELOG.md, Last accessed: 2023-03-15

benchmarks also significantly faster, reaching a speedup from 2.6x to 3.5x depending on the number of constraints as well as concrete implementation (underlying elliptic curve) used [Bot20]. These speedups are partly possible because of a specially built library for efficient field arithmetics.

A notable difference to ZoKrates is that Gnark does not use a purpose-built language but simply uses Go. Botrel [Bot20] explains that decision with the benefits during the whole development lifecycle, especially debug, document, test, and benchmark using Go and its robust toolchain. Besides that, full and stable IDE integration across platforms comes for free.

With *gnarkd*, there also is a way to use gnark as a service. This isolation may be desirable for reasons of software architecture, resource allocation, or security. Communication with the daemon is implemented with gRPC.

5.4.3 libspartan

libspartan [Set21] is a Rust library based on the equally named paper by Setty [Set19] from Microsoft Research. More concretely, that work describes Spartan, as a whole family of zk-SNARK schemes. The use of different underlying “commitment schemes” results in multiple concrete zk-SNARKs with distinctive properties regarding the trade-off between verifier costs and proof size.

Spartan’s outstanding feature is that it is a transparent zk-SNARK. This means a trusted setup, like an MPC ceremony, is not required because there is no “toxic waste” involved which needs to be kept secret. Furthermore, it offers sub-linear verification and linear proving costs for any R1CS statements. Compared to other modern zk-SNARKs with trusted setup, Spartan achieves a two-time speedup for proving arbitrary R1CS circuits with up to 2^{20} constraints. Data parallel workloads are even 16 times faster.

CHAPTER 6

Evaluation

This chapter evaluates the solution presented in Chapter 5. It starts with a review of the implementation against the design criteria specified in Chapter 4. Afterward, the collection of the datasets is described, followed by an analysis of the gathered data. Next, different evaluation scenarios based on the resulting data classification are discussed. Subsequently, the benchmark structure is outlined and the results are examined. The chapter concludes with a security analysis focusing on selected aspects relevant to relay solutions.

6.1 Fulfillment of Requirements

In the following, we discuss if the IOTA relay achieves the design goals established in Section 4.1 and additionally cover limitations and possible improvements.

- Forkless: The IOTA relay builds on milestone validations specified in the IOTA protocol such as verification of the signature that is part of the milestone payload. For this reason, no explicit support from the nodes is needed. Instead of relying on the centrally issued milestone signature, the trust in the relay could be increased if some nodes would cooperate in a sort of committee to add an aggregated signature to each milestone. Of course, that would contradict this design goal.

- Trustless: After the relay contract is deployed trust can be established by inspecting the included public keys. These keys are used for the on-chain verification of newly submitted milestones. Consequently, no trust in specific entities updating the relay state is required, because the updates are validated by the contract.

- Autonomous: As there is no specific single entity required to operate the relay contract it can be used by anyone with access to the source and target ledger.

- Robust: Due to the approach of using the milestone signature for validation, there is no dependency on previous milestone data, meaning milestones can be seen as a kind of decoupled “blocks”. Therefore a prolonged time without updates is not an issue. The relay can be updated with the latest (or another arbitrary) milestone without any further preconditions.

  Adding missing milestones in between at a later point in time is only necessary if a proof of inclusion for a transaction targets a milestone index not yet available at the relay contract.

  However, there are two limitations: The first is that the public keys included in the relay contract for milestone validation are only applicable until a specific milestone index. Milestones with an index greater than it cannot be verified and need an updated relay contract instance including public keys with matching index ranges.

  Another problem is the short interval at which new milestones (and messages in general) are issued, leading to a significant amount of disk capacity needed over time. Normal IOTA nodes like Hornet and Bee per default do not store the full ledger history. Instead, they create snapshots of the ledger state and prune old message data after reaching a configured database size or a milestone threshold[1].

  This implies that these nodes cannot provide milestones or data needed for inclusion proofs indefinitely. For such cases “permanodes” are needed that store the whole transaction history in a distributed database. IOTA’s permanode solution is called “Chronicle” and has recently been rewritten as an extension for the Hornet node[2].

- Corresponding: Milestones are issued by the trusted Coordinator and validated by nodes of the IOTA network. Relayed milestones that pass the signature verification and fit the data format requirements are considered valid. Merkle paths can be used to prove the inclusion of transactions within milestones. Due to limits in the milestone specification, a proof for non-value transferring messages is not possible.

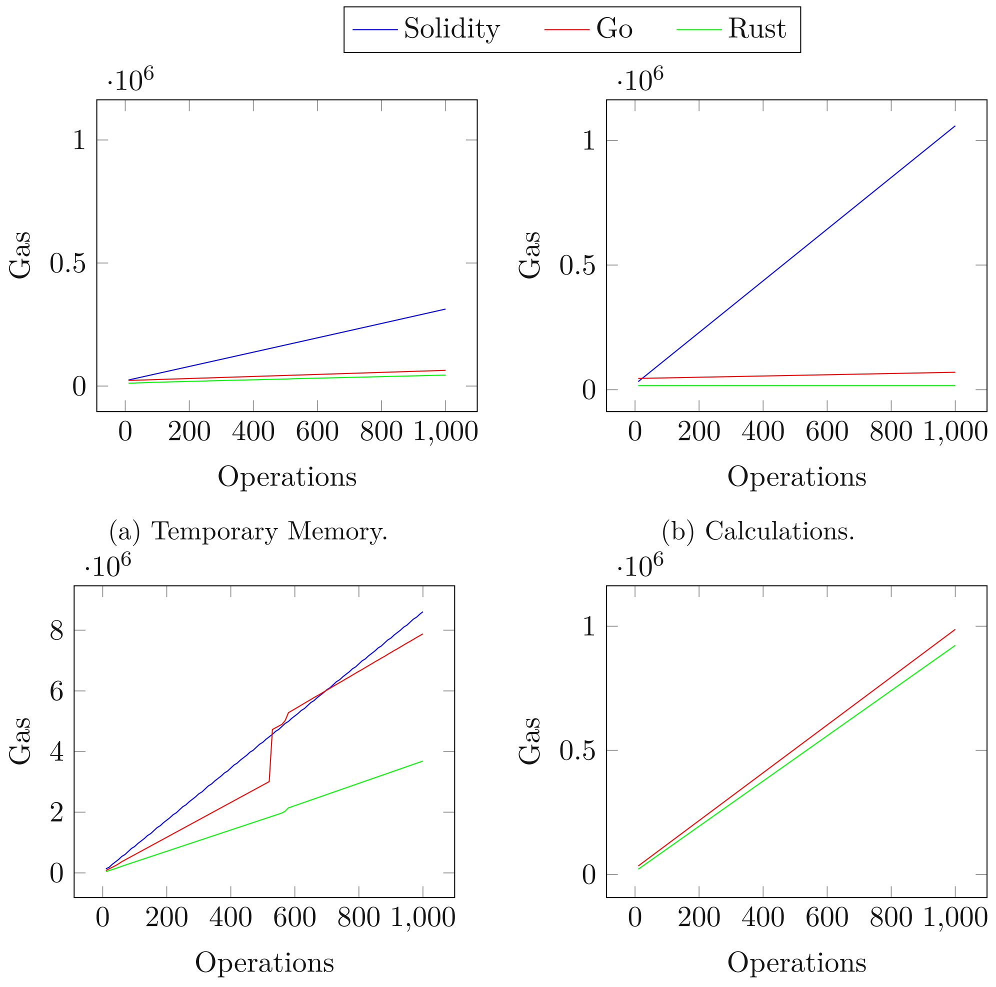
- Lightweight: The IOTA relay is lightweight because there is no requirement to constantly update it to keep it operable. Also, the creation of inclusion proofs by using the extended node software is not resource heavy. It only involves a depth first search similar to what is needed for milestone processing and the hashing to build the Merkle tree.

  Furthermore, the chosen programming language, Rust, shows the best gas utilization in benchmarks targeting the execution of calculations, temporary memory usage, and state storage writes when compared to Solidity, Typescript, and Go smart contract implementations (Figure 6.1). For more clarity, the data for Typescript was not plotted. It performed worse than Go and Rust, and in the case of storage, it even is significantly inferior to Solidity. The interested reader is referred to the original plots and scripts created by the IF [IF22b].

[1] https://wiki.iota.org/hornet/how_tos/managing_a_node#snapshot-pruning, Last accessed: 2022-12-09
[2] IOTA permanode https://github.com/iotaledger/inx-chronicle, Last accessed: 2023-03-15

In general, the most expensive operations in smart contracts are storage operations. Note that contrary to the other subfigures the y-axis for it is scaled differently. The sudden increase of storage operation gas usage for Go (and less notable Rust) in Figure 6.1c probably is due to internal storage restructuring necessary when hitting a specific size threshold.

In the context of this work, the existing gas usage tests were adapted in terms of output data format and extended by another test measuring the hashing performance of the blake2b implementation available in IOTA's WasmLib. As there was no blake2b function provided for Solidity, Figure 6.1d only shows the results for Rust and Go, which both perform similarly.



(a) Temporary Memory. (b) Calculations. (c) State Storage. (d) Hashing.

Figure 6.1: IOTA smart contract gas benchmarks for different programming languages. Adapted from [IF22b] and extended by Figure 6.1d.

6.2 Quantitative Analysis

6.2.1 Dataset

To acquire a realistic IOTA dataset for the evaluation of the relay contract, the following approach was used. First, a Docker image for the Bee node software, including our HTTP API extensions to fetch milestone-included messages and proof data as explained in Section 5.3, was built. A container using the image was then started via Podman with a configuration to connect to the IOTA mainnet and automatically pair to existing nodes. After collecting some milestones and their transaction data the node was stopped and the auto-pairing was disabled. That allowed us to start the node later and use the HTTP API without the node syncing more Tangle data.

This process was done twice on different dates and times of day to catch possible variations in the data:

1. October 3 2022 from 17:38:14 until 18:54:24 GMT,
   Milestone index 4512232 - 4512689 (458 milestones)
2. November 6 2022 from 13:19:23 until 14:55:53 GMT,
   Milestone index 4804402 - 4804981 (580 milestones)

Before using the gathered data for the relay contract, an analysis based on the listed metadata was conducted:

- Number of messages included in milestones:
  For building and evaluating a relay solution, it is important to ascertain an average milestone. A key property is the number of transaction messages it includes. Every included message is used to calculate the Merkle root hash stored in the milestone. This allows for building a proof of inclusion for any of them.

  To verify such a proof the relay needs the matching milestone data. However, many of the messages in IOTA are non-value messages which are not included in milestones, resulting in a remarkable number of "empty" milestones (Figure 6.2). Depending on the dataset, between 29% and 51% of the milestones include no messages at all, and hence are of no use for the relay.

  In order to give a better perception of the data interesting for further relay evaluation, Figure 6.3 shows the number of milestones with at least one included message in a cumulative histogram. The highest share of non-empty milestones just includes one message, causing the usage of the Merkle tree hash to be an unnecessary overhead. Approximated only 1.3% (Figure 6.3a) to 2.7% (Figure 6.3b) of the milestones include 4 or more messages. As a result, the data savings of the Merkle tree are marginal.

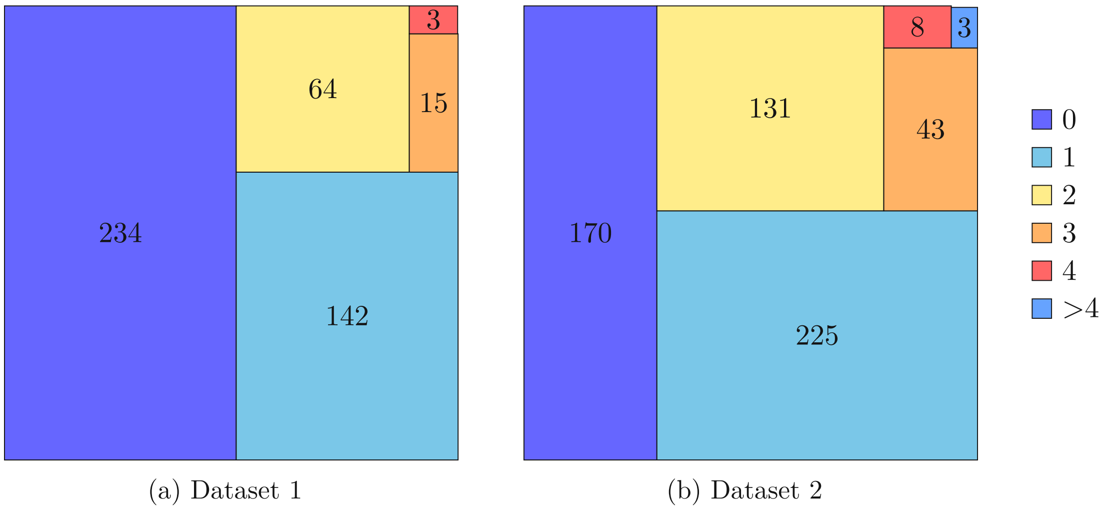


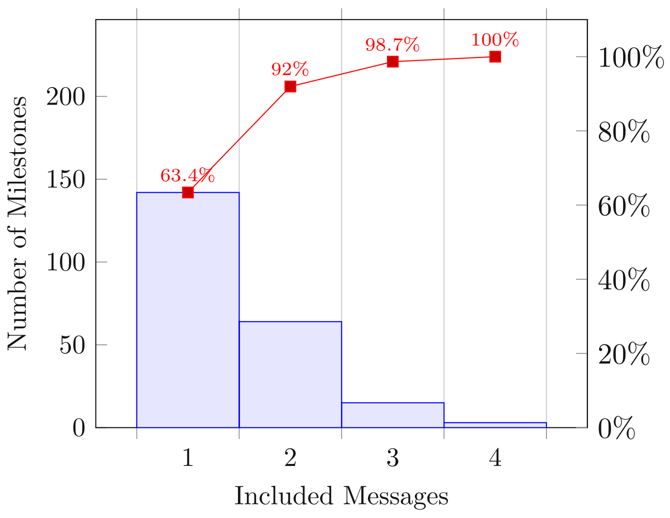
Figure 6.2: Overview of the number of milestones with a specific amount of included messages (0-4 and more) for both datasets.

- Timestamps:
  The many empty milestones partly are due to the interval in which milestones are created. When looking at the timestamps one can see that currently every 10 seconds a fresh milestone is published in the mainnet. This allows for fast finalization times but leads to the previously explained drawbacks.

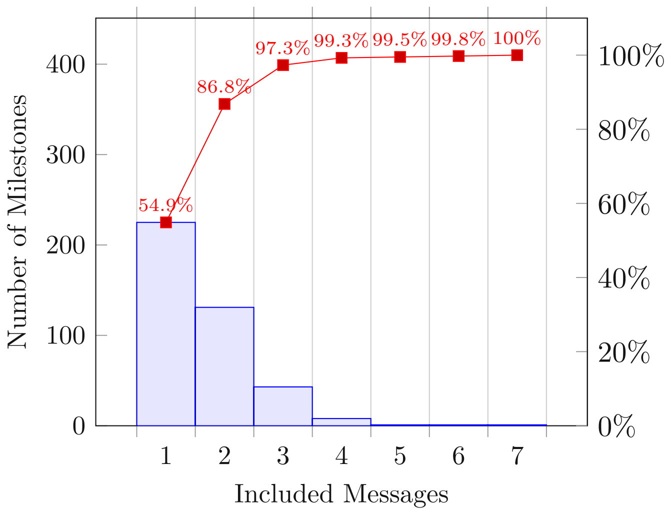
- Milestone payload byte size:
  Strictly speaking, a milestone always is a message with a milestone payload. Yet in this work, we use the term milestone for the milestone payload. Consequently, also the relay contract expects milestone payloads without the wrapping message. The raw data size of a milestone directly impacts execution costs when adding a milestone to the relay contract.

  There are not many dynamic components that allow increasing the size of a milestone. One part that is variable is the number of parents, which can be between one and eight. Usually, in the IOTA mainnet the maximal count is used, leading to milestones of 447 bytes. However, occasionally special milestones with an additional “receipt” payload[3] still occur. These allow the migration of funds from a previous IOTA version's signature scheme and have a size of 598 bytes.

[3]https://github.com/iotaledger/tips/blob/main/tips/TIP-0017/tip-0017.md#receipts, Last accessed: 2023-03-15



(a) Dataset 1



(b) Dataset 2

Figure 6.3: Cumulative histogram for the number of included messages in milestones (excluding 0) for both datasets.

- Message byte size:
  Contrary to a milestone, a transaction message has several possibilities to increase in size but the overall size limitation of 32 KiB for IOTA messages still applies.
  - Indexation payload. An additional payload that can be embedded into transaction messages. It adds an index as well as arbitrary data. Messages can be queried using this index via the node's API.
  - Inputs/Outputs. A different number of payment sources (inputs, UTXOs)

and targets (outputs) results in slightly varying transaction data sizes. Each input requires a matching unlock block with a signature.

The relay needs to parse and hash the whole message in order to verify the proof of inclusion of the message for a milestone. More data leads to more computational effort and therefore higher costs. Both datasets show a clear split into two groups, the messages with and without indexation payloads, depicted in Figure 6.4. In general, apart from the extent and the differing proportion of small messages, the datasets are quite similar.

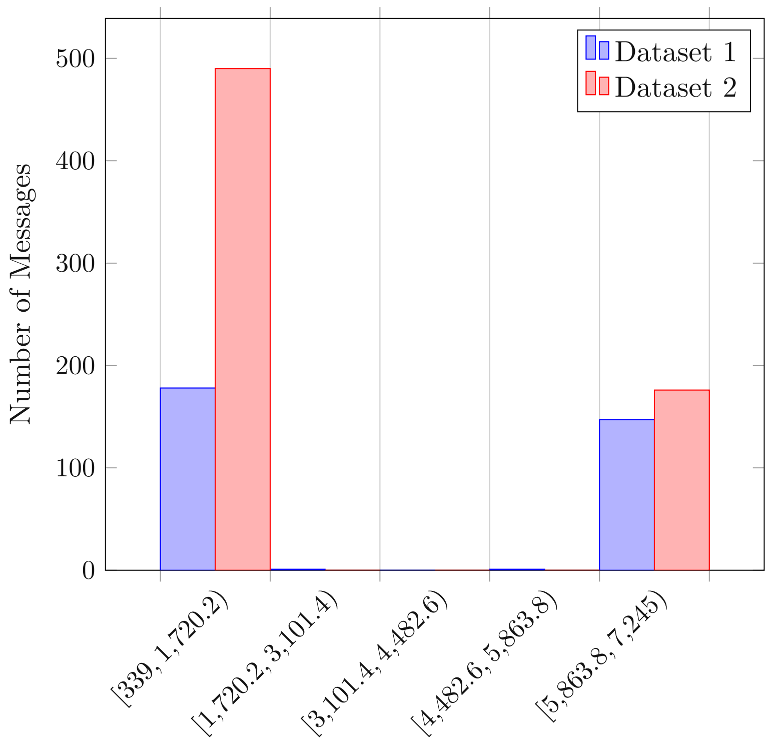


Figure 6.4: Histogram for the byte size of included messages.

6.2.2 Evaluation Scenarios

Based on the conducted data set analysis, multiple distinct evaluation scenarios can be derived. By that, a realistic assessment of the relay's costs in different use cases is possible. Furthermore, if it is not necessary to use concrete parts of the message after proving its inclusion, it suffices to do the proof just with the message id (hash) of the message instead of the full message data. That approach leads to a major reduction in execution (gas) costs. Note that the evaluation focuses on verifying transactions because adding milestones incurs nearly constant costs of around 11,600,000 gas due to the fixed size of 447 bytes. The following enumeration describes the evaluation scenarios.

(A) Full message data: The complete message bytes are sent to the relay contract where they are deserialized into a message object. The message is then hashed twice, first to get the message id and a second time for using as a leaf in the Merkle proof to prove the inclusion of the message in a given milestone. If the proof is valid the contract returns that result but it could also use any data included in the message, such as transaction details, with confidence.

   (1) Small transaction: A normal transaction without any indexation payload using one input and one output.

   (2) Medium transaction: Multiple inputs or outputs and a small indexation payload.

   (3) Large transaction: A large number of inputs or outputs with an indexation payload of medium size.

   (4) Huge indexation payload: Indexation payload among the maximum observed size in the datasets.

(B) Message ID only: Instead of the full message data just its message ID (the hash of the data) is used. Although, in general, there are no obvious use cases for this scenario it is included for comparative reasons. It vastly reduces data that needs to be transferred and deserialized at the contract and reduces the hashing by one round.

(C) Number of included messages: Although the preceding dataset analysis showed that milestones include only a few messages most of the time, it still is important to assess the course of the gas consumption with respect to an increase of included messages and therefore the height of the Merkle tree.

An overview of the scenario group (A) is given by the scatter plot in Figure 6.5 showing the correlation between bytes and gas usage of the message verification for all messages included in the datasets. As already foreshadowed by the histograms (Figure 6.4) there are two clusters, which are mainly caused by the difference in indexation payload. The one on the left mainly consists of messages without or with small- to medium-sized indexation payloads while the right cluster is the result of a third-party project using huge indexation payloads compared to other messages. In addition, two outliers for the second dataset that deviate from the linear relationship can be observed. However, the distinction between the transaction scenarios (A1) to (A3) is not clearly visible in the plot. Furthermore, it does not show how the number of messages included in milestones affects the gas usage when verifying transactions.

To give a more unobstructed view, Table 6.1 lists the message byte sizes and the gas costs for doing the inclusion proof for representatives of the described evaluation scenarios. It clearly shows that “simple” value transactions like scenarios (A1) to (A3) are located at the bottom left cluster in the scatter plot. Note that in this case the representatives were chosen such that they each are the only included message of a milestone.

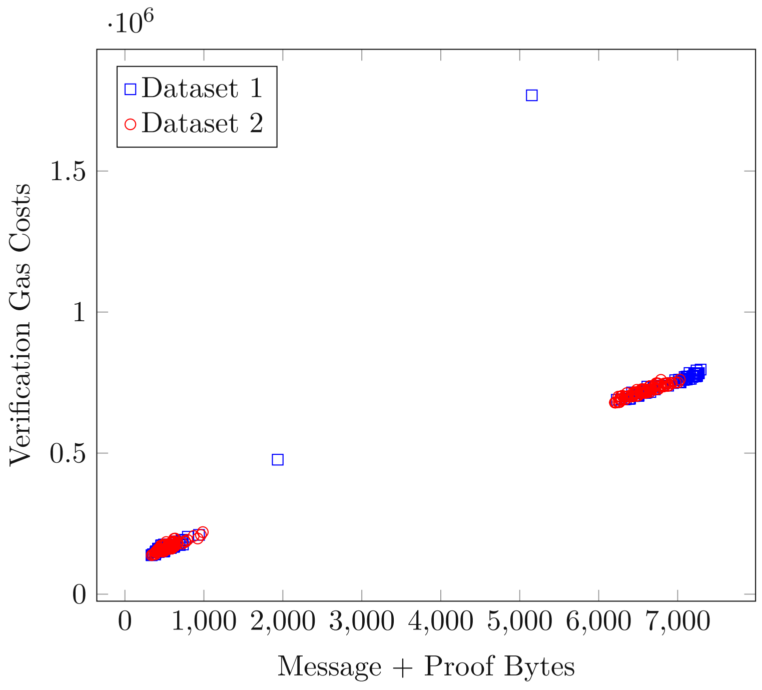


Figure 6.5: Scatter plot of the relation between byte size and verification gas costs.

As can be seen in the fourth column, the gas costs per byte are lowest for the biggest message. The reason for that is the diminishing cost overhead of the common data passing and the fact that in its current implementation, the relay contract does not store any data when verifying transactions. If the contract would for example save indexation payloads the cost factor for scenario (A4) compared to the others would increase remarkably as storage is more expensive than computation time (like needed for hashing).

Table 6.1: Overview of evaluation scenarios A and B.

Transaction Scenario		Message Bytes	Verification costs	Gas/Byte
A1	Small	339	138,232	407.76
A2	Medium	527	167,549	317.93
A3	Large	696	182,959	262.87
A4	Huge indexation	7245	773,541	106.77
B	MessageID only	32	72,581	2268.16

The effect of the number of messages in a milestone on the verification gas costs can be seen in Figure 6.6. More included messages mean increased proof bytes due to additional leaves in the Merkle tree, which in turn results in additional hash calculations for the inclusion proof. Contrary to Figure 6.5, we use only the message bytes for the x-axis in this case. By that the difference in gas costs due to the position of the message in the

Merkle tree is better visible: messages of identical size that are included in milestones with the same number of messages can have different gas costs because of the varying number of tree leaves needed to construct the root hash. This is one of the reasons for points aligning one above the other in Figure 6.6.

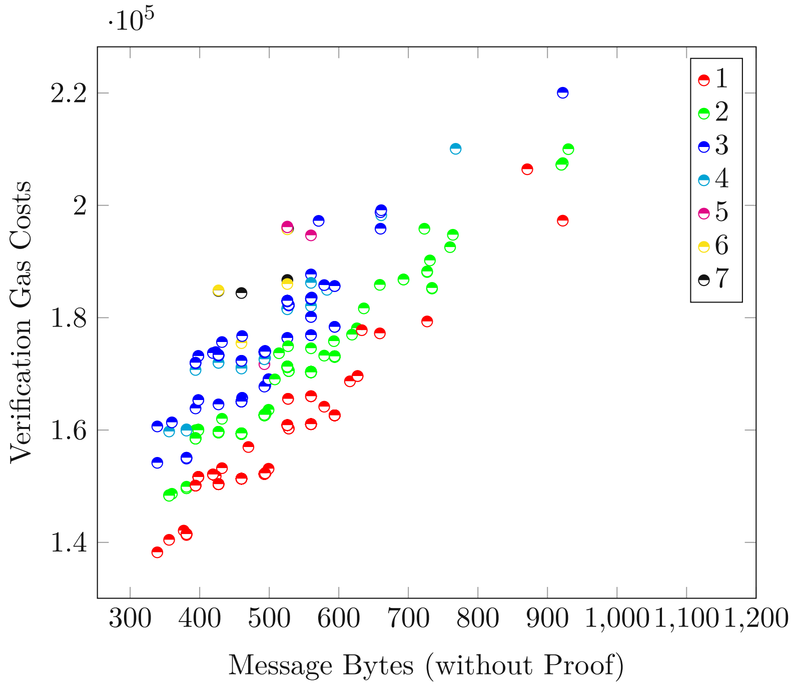


Figure 6.6: Scatter plot for Dataset 1[x<1200] similar to Figure 6.5. The colors indicate the number of included messages in the milestone.

6.3 Security Analysis

Apart from the occasional actual bug in smart contract code leading to exploits, it is even more important to consider a relay under multiple scenarios, to understand its level of security. A popular way of security analysis in DLT literature is the BAR (Byzantine, Altruistic, Rational) model. In it system participants are categorized by their behavior. Byzantine actors could deviate from the expected relay scheme for various reasons, for example, because they strive to reach an arbitrary goal or simply are faulty. Contrary, altruistic users obey the scheme independently of any gain different actions could result in. The last class, rational clients, always aim to maximize their own profit, no matter if it means following the procedure or deviating from it.

There is a multitude of ways to attack DLTs directly or concrete smart contract implementations. Some are applicable to whole classes of systems, others are so specialized they only work for selected ones. In the following two selected security threats with respect to relays are discussed [FSS+20].

6.3.1 Relay Poisoning

For a relay to be of any use, blocks (data) of the source ledger need to be submitted to it continuously. Other smart contracts can utilize the relay for proof of inclusion for transactions, e.g., to create tokens based on them. For relay poisoning an attacker tries to get (invalid) data into the relay that can be used to its advantage, for example, by using transactions that never took place like that on the source ledger for getting some tokens. Usually, there are two possible ways such an attack can be achieved.

The first option is to just send invalid blocks to the relay, the other is to send data passing validations but including faked transactions. Submitting invalid data to the relay only works if it optimistically accepts data and only checks it in case of other users disputing it. In the case of PoW that would greatly reduce the effort for an attacker. However, the attacker somehow needs to stop all other participants from starting disputes over the invalid blocks until the lock time has passed. One way to reach that goal is an "incentive attack", where the attacker offers rewards higher than the ones for successful disputes. Rational clients want to maximize profit and therefore choose this alternative compensation. Even if the attacker is well-funded enough and is able to reach all the other participants the attack most likely will not be successful, as just one altruistic user that triggers a validation is enough to stop it. This consideration of having at least one altruistic participant is often used as a practical assumption when reasoning about security in distributed permissionless systems, as it is difficult to guarantee correct results with just rational behavior.

The attacker's second option is to build correct blocks that contain faked transactions. This means a validation of the block headers in case of a dispute would pass. However, a correct PoW is needed, vastly increasing the necessary amount of computing resources. To succeed with the attack, all that is required is to stay ahead of legitimate source ledger blocks submitted to the relay. Therefore also in this case alternative rewards raise the chances of a successful attack by giving an incentive to not submit valid blocks.

In IOTA there are no blocks, just messages in the Tangle linking to others. However, in regular intervals, milestones with special signatures, that confirm some of the new messages, are issued. The relay only works with milestones that similar to usual blocks also contain a Merkle root hash for all the messages they confirm. When a milestone is submitted to the relay its signatures are verified via the pre-shared public keys that are valid for a specific milestone interval. As long as this is done for every milestone, relay poisoning is not possible. Contrary to building a fake block with the right PoW hash, issuing a milestone requires the correct private key to create a signature passing the validation.

6.3.2 Changes to the Source Ledger

A change of the source ledger also has effects on its relays, for example, if there are changes in what is considered a valid block or message. Without adapting the relay, it could either reject now valid blocks or still accept ones that are invalid according to the new rules. Furthermore, relays can also be more indirectly affected, e.g., in the case of a fork occurring at the source ledger. Such an event does not necessarily require a change in the relay but depending on the concrete implementation it could lead to competition on what is the source ledger for the relay going forward.

For the IOTA relay described in this work, the most crucial changes are the keypairs used for signing the milestones. As already mentioned in Section 4.1.3 there is no on-chain protocol for updating the keys in use for the current range of milestones, nor for future ones where the public keys are not known yet. Therefore additional contract instances need to be deployed when a new set of keys is published. The same reason also makes issues with forks unlikely as the keys would be different in that case.

6.4 Discussion

This chapter showed that the proposed relay solution fulfills the requirements derived from related literature. To argue how lightweight the relay implementation using Rust as a programming language is, gas utilization benchmarks targeting different relevant aspects were executed for Rust, Go, and Solidity, with Rust giving the best results. In order to measure the relay's performance, various evaluation scenarios based on real-world data were conducted.

The datasets for evaluating the relay contract were obtained by connecting containers of the Bee node software to the IOTA mainnet and storing incoming transactions and milestones. Extensions made to the HTTP API enabled the fetching of the required data, including proofs of inclusion for transactions. Two datasets were collected on different dates in October and November 2022, with a size of 458 and 580 milestones. To build a foundation for the analysis, the gathered data was first examined in detail to surface the possible implications for further steps.

Surprisingly, a significant number of milestones were found to be empty as they did not include any transactions, ranging from 29% to 51% depending on the dataset. Furthermore, the majority of the remaining milestones only include up to four transactions. This mostly is due to the fact that a new milestone is issued every ten seconds. Milestones usually have the same size of 447 bytes, while message sizes vary widely between 339 and 7245 bytes.

Based on these insights collected from the datasets, several scenarios were defined and evaluated in Section 6.2.2. Both the number of messages inside milestones and the size of messages have an impact on the efficiency and cost of the relay. Although two datasets were collected at different times both show similar properties and therefore also results. The initial assumption was that the data would exhibit greater diversity and consist of more milestones containing a higher number of transactions. Apart from two outliers and the baseline established by scenario B, the results show a quasi-linear relation between message bytes and verification costs. Data handling and especially storing are the most gas-expensive operations in smart contracts. Therefore these topics always are primed for additional tests and in-depth optimizations but are not the scope of this work.

Assessing the meaningfulness of the results is challenging due to the implementation's reliance on evolving software components that are still in an experimental state. For example, gas usage may fluctuate as a result of ongoing adjustments to the costs associated with certain operations. In addition, there is no baseline to compare relay gas costs to, owing to the early stage of the IOTA smart contract framework. Moreover, contrary to, e.g., Ethereum, in the planned Assembly network (based on the IOTA smart contract framework) [IF21b], chain operators can assign fees on their own, possibly resulting in multiple instances of the relay with monetary costs.

The evaluation is concluded by arguing about the relay's security, regarding relay poisoning and changes to the source ledger, under the BAR model. Similar to the IOTA network itself the relay relies on the security of the centralized Coordinator and its signatures.

A promising way to optimize both the efficiency and the security of the relay is to utilize zk-SNARKs. By that, a more thorough approach could be taken to increase trust in the relay without raising gas consumption, due to the off-chain creation of the proof and the near-constant verification costs for it. Furthermore the high costs of validating milestones, compared to transaction verification, could potentially be reduced with the help of zk-SNARKs. In addition, the tests should be repeated with recent IOTA smart contract framework versions as well as after revisiting and improving the relay code. Updates to the Rust programming language and for the compiler to Wasm can also potentially have a significant impact on performance.

CHAPTER 7

Conclusion

In this concluding chapter, we first provide a comprehensive summary of the research conducted, highlighting key findings and insights gathered throughout this work. We also explore potential directions for future work by discussing areas where further investigation and development seem promising for the results of this work as well as the broader field of research.

7.1 Summary

Research in the area of DLT in the last few years mostly has focused on two broad topics, scalability and interoperability. The first has to be improved in order to handle increased traffic and avoid congestion of the system during high load. The latter stems from the fact that over time more and more competing projects have evolved, resulting in fragmentation of the field due to the various incompatible technologies in use. For both, several solution approaches surfaced. Regarding scalability, these are the optimization of components (algorithms, data structures, networking) as well as their parameters, and ways of splitting and distributing work, such as sharding and introducing sidechains or other means of moving computations off the main chain. Shards and multiple chains need efficient ways of communication. Therefore, scalability has overlapping research goals with interoperability.

To establish deeper knowledge about scalability and interoperability approaches, the thesis started with extensive literature research including whitepapers and Web pages of DLT projects, as these resources give a more up-to-date view compared to the often outdated information found in traditional research papers.

In Chapter 2, we described the evolution of DLTs and their challenges as well as possible solutions in a general way to give a background for the remaining thesis.

DLTs can be classified into three generations. The first generation is defined by Bitcoin, the first cryptocurrency, built in a decentralized way using a blockchain and PoW. However, it was not designed in a way that fostered adaptation and did not support building custom programs on top of it. With the second generation and Ethereum that changed, because smart contracts were introduced as a way to create own distributed applications that execute on the network. The new possibilities and the fact that all the participating nodes have to execute the smart contracts, lead to an increased load on the system. This is why third-generation DLTs focus on efficient solutions with scalability in mind. Besides scalability, projects of the third generation also introduced solutions for interoperability between multiple DLT instances and made it easier to build application-specific blockchains that integrate with each other.

Because blockchains are a linear data structure, they pose a bottleneck. To resolve this issue several approaches utilizing DAGs were proposed that offer a high transaction throughput. However, DAG-based DLTs come with their own drawbacks and challenges in terms of decentralization and security.

Other promising techniques to scale DLTs are sharding and ZKPs. Sharding aims to grow the overall capacity by scaling horizontally and distributing work while keeping the overhead of handling additional nodes as low as possible. ZKPs in their various manifestations can be used as a tool for many use cases. They allow the construction of proofs for statements over a secret that can be verified without giving the secret away. This characteristic makes them a valuable construct for privacy-related goals. A family of ZKPs called zk-SNARKs has the property of succinct proofs, meaning that the proof is small and fast to verify compared with the statements it proves. Therefore it is primed for usage in DLTs, where it is beneficial to do as much work as possible off-chain and have a low memory and computation footprint on-chain.

After establishing an understanding of the general topics, Chapter 3 goes into more detail by exploring related work for the previously mentioned areas specific to the field of DLTs. We discuss the scalability trilemma and present different approaches for scaling done in various projects while also pointing to the resulting drawbacks. Furthermore, an overview of the different consensus strategies is given. The chapter continues by describing off-chain as well as cross-chain solutions. Afterward, sharding is revisited in the context of DLTs and a concrete example is given with Ethereum 2.0. In the following, we focus on two categories of interoperability: Public connectors, such as relays, and the concept of BoBs. For both, examples based on the literature or actual projects are discussed. Similar to sharding, we take another look at ZKPs, concretely zk-SNARKs, and their usage in the field of DLTs. The chapter concludes by describing and comparing three DAG-based DLTs, namely IOTA, Hedera, and Nano.

Chapter 4 and Chapter 5 propose the design and implementation for an IOTA relay built with the ISC framework. We derived the requirements and design goals from existing literature on relays for PoW- and PoS-based ledgers. Using that as a guideline, we reasoned about the specifics of IOTA and the realization of a relay for it.

For the implementation, we chose the Rust programming language, based on the fact that it was stated as the main language for IOTA going forward, it has good support of WebAssembly and performed notably better than Go in benchmarks, displayed in Figure 6.1. In addition, Wagner et al. [WMM+23] also identified Rust as a better choice than Go in their exploration of Wasm performance across various programming languages. This decision turned out to be challenging and in hindsight, we would choose Go instead because, in the ISC framework, it needs to be used for testing and allows for easier debugging. Having to handle two languages and their various dependencies was very time-intensive and caused a lot of issues. That was especially the case due to the plan of reusing existing libraries to write the actual relay code that handles data of the current IOTA mainnet while the ISC frameworks are based on the devnet. Therefore, different incompatible versions of multiple libraries were needed in the project causing version conflicts that partly had to be solved by forking.

While evaluating the different processes and specifications of IOTA, we found several options for improvements. Milestones only include transactions by means of a Merkle tree hash but not conflicts or data-only messages. Furthermore, we had to extend the API of the node software on our own to be able to fetch the actual transactions included in a milestone to be able to build a proof of inclusion. The Coordinator signs each milestone using two different keys. The IOTA nodes and also the relay verify these signatures. However, the keys are only used for a defined range of milestones. Afterward, they need to be manually updated because there is no on-chain protocol for that.

To enable an efficient and secure implementation of, e.g., relays, DLTs, and their consensus have to be designed with interoperability in mind. Security-relevant processes such as the rotation of keys have to be automated and should happen on-chain.

The evaluation of the proposed solution is done in Chapter 6. For a summary, we refer to Section 6.4.

7.2 Future Work

In this section, we outline various possible improvements and extensions to the proposed relay solution as well as some more general topics for future work in the field of DLT interoperability and scalability.

Similar to ETH Relay, an optimistic or on-demand approach could be applied to the proposed relay solution, which in turn would require a supporting incentive scheme. To increase trust in the relay in addition to the Coordinator milestone signatures, measures like for example a confirmation by means of a threshold signature coming from some of the IOTA ledger nodes or a dispute timeframe before milestones can be used, would be beneficial.

One highly promising strategy to enhance both security and efficiency involves leveraging zk-SNARKs to move computations off-chain and only verify the resulting proof at the relay contract.

A first step would be to port the existing solution for adding milestones and verifying transactions to use zk-SNARKs, which could be extended to allow batch proofs of multiple milestones and transactions at once. In the case of a relay implementation for the EVM, using zk-SNARKs instead of verifying the ed25519 milestone signatures on-chain has less gas costs. The reason is that the EVM does not have a pre-compiled function to verify ed25519 signatures. Batching the verification of multiple signatures into one proof increases the cost savings even more. The discussion and implementation of such a solution by Goel, Ghangas, and Jain [GGJ22] can serve as a starting point for future research.

However, ed25519 internally uses SHA-512[1], which cannot be used efficiently inside zk-SNARKs, diminishing the possible performance gains. In zkRelay (Section 3.4), Westerkamp and Eberhardt [WE20] combine batch proofs for blocks with a Merkle tree using a zk-friendly hash function. By that construct, they validate all blocks of the batch off-chain but store only data about the last block on-chain to reduce storage cost, while still allowing to prove the inclusion of intermediate blocks, via the Merkle tree, later on if needed.

zk-SNARKs also open up whole new possibilities that would not be feasible to do on-chain. Advanced uses are building proofs for the whole state of the ledger and aggregating multiple proofs into one (also referred to as "recursive composition"). One recent approach is "zkTree: A Zero-Knowledge Recursion Tree with ZKP Membership Proofs" by Deng and Du [DD23]. Utilizing their proposed framework for a new relay implementation instead of the less sophisticated Merkle tree usage (described in the previous paragraph) looks promising. Fitting to our use case of validating the signatures of IOTA milestones, the authors give an example of employing zkTree for validating ed25519 signatures in a single proof within the EVM. As IOTA will move from the Coordinator, currently using two signatures per milestone, to a decentralized committee of ten members, each apparently signing milestones independently[2], optimizing signature validation for the relay is even more crucial than before.

When designing new systems around zk-SNARKs, it is important to do it in a "zk-friendly" way, by choosing suitable cryptographic primitives. For example, the often-used hash families SHA or BLAKE cannot be used efficiently, which has already been researched and led to novel hash function constructions, such as "Poseidon" [GKR+21].

Further potential questions to answer are under which criteria (for example, data size or number of messages included in the batch) zk-SNARKs offer better efficiency for a relay compared to direct on-chain computations. When following such an approach, the impact on clients has to be considered, as creating zk-SNARK proofs can be quite resource-intensive, especially regarding RAM consumption. A comparison between multiple zk-SNARK libraries implementing the same underlying scheme as well as a benchmark

[1]RFC8032: Edwards-Curve Digital Signature Algorithm (EdDSA). https://datatracker.ietf.org/doc/html/rfc8032, Last accessed: 2023-09-27

[2]IOTA testnet milestones have ten signatures as seen at https://explorer.iota.org/testnet, Last accessed: 2023-09-27

across different schemes would be of interest for both on- and off-chain computations. Our overview of libraries in Section 5.4 showed that the most common available scheme is [Gro16]. Nevertheless, more recent schemes, for example, Spartan, can offer better performance and useful properties like transparency. Although Rust seems to be a popular choice for implementing zk-SNARKs, with *gnark* a Go library is available that could be used for further tests regarding the impact of using different programming languages in an IOTA smart contract.

Part of the reason for constructing the relay using the ISC framework was its support of Wasm, enabling the utilization of existing IOTA libraries for the implementation. In the meantime, the development of the ISC and its dependencies, as well as other IOTA libraries and the node software progressed notably. Therefore an update and reevaluation of the implemented relay could show improved efficiency and would serve as a more relevant baseline.

To underscore the potential for reuse, subsequent efforts should involve abstracting the code from the actual ISC framework, to assess the possibilities of deployment to other Wasm-compatible DLTs. This is even more important as a significant shift occurred regarding the preferred method to execute smart contracts. Initially, the focus was on Wasm, while the EVM support was labeled as experimental. Now the EVM is set to be released for Shimmer soon and the Wasm VM should only be used for experimental purposes, giving more reason to deploy the contract to another Wasm-capable DLT. The decision to prefer the EVM comes despite performance evaluations indicating Wasm's superior performance. The rationale behind this change most probably is the EVM's widespread adoption and its ability to facilitate the deployment of existing smart contracts originally designed for platforms like Ethereum or any other DLT that supports the EVM. Therefore, a direction for future work is to take a close look at performance and feature differences between the EVM and Wasm in the ISC framework. In terms of relay implementation with zk-SNARKS, the EVM has the benefit that the ZoKrates framework directly outputs verification smart contracts.

Research on a novel consensus algorithm to get rid of the Coordinator for IOTA 2.0 is still ongoing at the IF. While working on this thesis, one proposed solution has already been implemented in the devnet for evaluation but at a later point, the approach changed. Future work could look into the forthcoming, ideally zk- and interoperability-friendly, finalized specifications in order to build a relay for it.

Bibliography

[AVD20] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. "Preventing Denial of Service Attacks in IoT Networks through Verifiable Delay Functions". In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference* (2020), pages 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322260.

[Bai16a] Leemon Baird. *Hashgraph Consensus: Detailed Examples*. 2016. URL: https://www.swirlds.com/downloads/SWIRLDS-TR-2016-02.pdf (visited on 2021-05-13).

[Bai16b] Leemon Baird. *The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance*. 2016. URL: https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf (visited on 2021-05-13).

[Bar] Joshua Baron. *Securing Information for Encrypted Verification and Evaluation*. Defense Advanced Research Projects Agency. URL: https://www.darpa.mil/program/securing-information-for-encrypted-verification-and-evaluation (visited on 2021-08-09).

[Bas21] Gilbert Bassey. *What Are Parachain Auctions? — All You Need to Know*. Coinmonks. 2021. URL: https://medium.com/coinmonks/what-are-parachain-auctions-all-you-need-to-know-df54f96c552c (visited on 2021-10-06).

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, Transparent, and Post-Quantum Secure Computational Integrity*. 046. 2018. URL: http://eprint.iacr.org/2018/046 (visited on 2021-09-08).

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. "The Hunting of the SNARK". In: *Journal of Cryptology* 30.4 (2017), pages 989–1066. DOI: 10.1007/s00145-016-9241-9.

[BCC+20] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kılınc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. *Overview of Polkadot and Its Design Considerations*. 2020. URL: https://github.com/w3f/research/blob/master/docs/papers/OverviewPaper-V1.pdf (visited on 2021-09-23).

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. “From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS ’12. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pages 326–349. DOI: 10.1145/2090236.2090263.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *2014 IEEE Symposium on Security and Privacy*. 2014, pages 459–474. DOI: 10.1109/SP.2014.36.

[BCT20] Aritra Banerjee, Michael Clear, and Hitesh Tewari. *Demystifying the Role of Zk-SNARKs in Zcash*. 2020. arXiv: 2008.00881 [cs].

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. “Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture”. In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pages 781–796. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.

[BDE+11] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. “On the Security of the Winternitz One-Time Signature Scheme”. In: *Progress in Cryptology – AFRICACRYPT 2011*. Lecture Notes in Computer Science. Springer, 2011, pages 363–378. DOI: 10.1007/978-3-642-21969-6_23.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. “Noninteractive Zero-Knowledge”. In: *SIAM journal on computing* 20.6 (1991), pages 1084–1118. DOI: 10.1137/0220068.

[BHK+20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. *Combining GHOST and Casper*. 2020. arXiv: 2003.03052 [cs.CR].

[BHM20] Leemon Baird, Mance Harmon, and Paul Madsen. *Hedera: A Public Hashgraph Network & Governing Council*. 2020. URL: https://hedera.com/hh_whitepaper_v2.1-20200815.pdf (visited on 2021-05-13).

[BK21] Alex Biryukov and Dmitry Khovratovich. *Equihash-Nano Research Report*. 2021. URL: http://content.nano.org/ABDK-Nano-Equihash-Report.pdf (visited on 2021-05-17).

[BKLM19] Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J. Marmolejo-Cossío. *Fairness and Efficiency in DAG-Based Cryptocurrencies*. 2019. arXiv: 1910.02059 [cs].

[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)*. 2020, pages 1–7. DOI: 10.1109/COINS49042.2020.9191430.

[BMR] Joseph Bonneau, Izaak Meckler, and Vanishree Rao. *Mina: Decentralized Cryptocurrency at Scale*. URL: https://minaprotocol.com/static/pdf/technicalWhitepaper.pdf (visited on 2021-01-04).

[BNTT20] Daniel Benarroch, Aurelien Nicolas, Justin Thaler, and Eran Tromer. *A Benchmarking Framework for (Zero-Knowledge) Proof Systems*. 2020. URL: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-benchmarking.pdf (visited on 2021-07-17).

[Bot20] Gautam Botrel. *Introducing gnark: a fast zero-knowledge proof library*. HackMD. 2020. URL: https://hackmd.io/@zkteam/gnark (visited on 2021-07-25).

[Bra] Brainbot Labs. *What Is the Raiden Network?* URL: https://raiden.network/101.html (visited on 2021-10-10).

[BS84] László Babai and Endre Szemeredi. "On The Complexity Of Matrix Group Problems I". In: *25th Annual Symposium on Foundations of Computer Science, 1984*. 1984, pages 229–240. DOI: 10.1109/SFCS.1984.715919.

[But20] Vitalik Buterin. *An explanation of the sharding + DAS proposal*. HackMD. 2020. URL: https://hackmd.io/@vbuterin/sharding_proposal (visited on 2021-12-06).

[But21a] Vitalik Buterin. *An Approximate Introduction to How Zk-SNARKs Are Possible*. 2021. URL: https://vitalik.ca/general/2021/01/26/snarks.html (visited on 2021-06-13).

[But21b] Vitalik Buterin. *Ethereum Whitepaper*. ethereum.org. 2021. URL: https://ethereum.org/en/whitepaper/ (visited on 2021-10-12).

[But21c] Vitalik Buterin. *The Limits to Blockchain Scalability*. 2021. URL: https://vitalik.ca/general/2021/05/23/scaling.html (visited on 2021-06-13).

[But21d] Vitalik Buterin. *Why Sharding Is Great: Demystifying the Technical Properties*. 2021. URL: https://vitalik.ca/general/2021/04/07/sharding.html (visited on 2021-06-13).

[But22] Vitalik Buterin. *The Different Types of ZK-EVMs*. 2022. URL: https://vitalik.ca/general/2022/08/04/zkevm.html (visited on 2023-09-14).

[BVGC21] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. *A Survey on Blockchain Interoperability: Past, Present, and Future Trends*. 2021. arXiv: 2005.14282 [cs].

[Cam21] Darcy Camargo. *Scalability and Sharding - IOTA Research Symposium 2021*. IOTA Foundation. 2021. URL: https://www.youtube.com/watch?v=hdWEdIuBF5k (visited on 2021-08-30).

[CG12] Jan Camenisch and Thomas Groß. "Efficient Attributes for Anonymous Credentials". In: *ACM Transactions on Information and System Security* 15.1 (2012), 4:1–4:30. DOI: 10.1145/2133375.2133379.

[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*. Zagreb, Croatia: Springer-Verlag, 2020, pages 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[Cho19] Jason Choi. *Demystifying Cosmos: Atomic Swaps, Ethereum, Polkadot and the Path to Blockchain Interoperability*. Based on an interview with Sunny Aggarwal from Tendermint, a core contributor to Cosmos. The Spartan Group. 2019. URL: https://medium.com/the-spartan-group/demystifying-cosmos-atomic-swaps-ethereum-polkadot-and-the-path-to-blockchain-interoperability-d1a2d75c20d6 (visited on 2021-10-02).

[CHX18] Jeff Coleman, Liam Horne, and Li Xuanji. *Counterfactual: Generalized State Channels*. 2018. URL: https://l4.ventures/papers/statechannels.pdf (visited on 2022-01-02).

[CKN+22] Mauro Conti, Gulshan Kumar, Pranav Nerurkar, Rahul Saha, and Luigi Vigneri. “A Survey on Security Challenges and Solutions in the IOTA”. In: *Journal of Network and Computer Applications* 203 (2022). DOI: 10.1016/j.jnca.2022.103383.

[Con21] ConsenSys. *Gnark Documentation*. 2021. URL: https://docs.gnark.consensys.net/en/0.4.0/ (visited on 2021-07-25).

[Cor20] Matt Corallo. *Bitcoin Improvement Proposal 152 - Compact Block Relay*. 2020. URL: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki (visited on 2021-10-27).

[Cos21] Cosmos. *IBC Technical Specifications*. 2021. URL: https://github.com/cosmos/ibc (visited on 2021-10-28).

[CZZ20] Shan-Te Chao, Yang Zhao, and Jun Zhao. *Reviewing Blockchain Scalability Challenge with a Discussion of Off-Chain Approaches*. 2020. DOI: 10.13140/RG.2.2.19055.87209.

[Dah21] Treyce Dahlem. *A Journey Through the Cosmos (ATOM)*. The TIE Research. 2021. URL: https://research.thetie.io/a-journey-through-the-cosmos-atom/ (visited on 2021-10-06).

[DAR21] DARPA. *Researchers Demonstrate Potential for Zero-Knowledge Proofs in Vulnerability Disclosure*. Defense Advanced Research Projects Agency. 2021. URL: https://www.darpa.mil/news-events/2021-04-22 (visited on 2021-08-09).

[DD23] Sai Deng and Bo Du. “zkTree: A Zero-Knowledge Recursion Tree with ZKP Membership Proofs”. Recording: https://www.youtube.com/watch?v=2s_c6KD5Yyc. Zero Knowledge Summit 9 (April 4, 2023). Lisbon, 2023. URL: https://eprint.iacr.org/2023/208 (visited on 2023-09-27).

[DDL+19] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. “Towards Scaling Blockchain Systems via Sharding”. In: *Proceedings of the 2019 International Conference on Management of Data*. SIGMOD ’19. New York, NY, USA: Association for Computing Machinery, 2019, pages 123–140. DOI: 10.1145/3299869.3319889.

[DeS21] Marta De Stradis. *ZKP in Action - How Insurance Industry Can Benefit from Secure Data Collaboration*. QEDIT. 2021. URL: https://qed-it.com/2017/07/04/zkp-in-action/ (visited on 2021-08-02).

[Do23] Thuat Do. *SoK on Blockchain Evolution and a Taxonomy for Public Blockchain Generations*. 2023. DOI: 10.2139/ssrn.4377849. preprint.

[DPS+20] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. *SOK: Cryptocurrency Networking Context, State-of-the-Art, Challenges*. 2020. URL: https://www.univie.ac.at/ct/stefan/ares20.pdf (visited on 2021-02-18).

[Drą21] Evaldas Drąsutis. *IOTA Smart Contracts*. IOTA Foundation, 2021. URL: https://files.iota.org/papers/ISC_WP_Nov_10_2021.pdf (visited on 2021-11-11).

[Edg] Ben Edgington. *Eth2 Annotated Spec*. Ethereum 2.0 Phase 0 – The Beacon Chain. URL: https://benjaminion.xyz/eth2-annotated-spec/phase0/beacon-chain/#introduction (visited on 2022-01-05).

[EGSV16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol”. In: *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. NSDI’16. USA: USENIX Association, 2016, pages 45–59. URL: https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-eyal.pdf (visited on 2021-10-09).

[ET18] Jacob Eberhardt and Stefan Tai. “ZoKrates - Scalable Privacy-Preserving Off-Chain Computations”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pages 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199.

[Etha] EthHub. *Plasma*. URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/ (visited on 2021-10-10).

[Ethb] Ethhub. *ZK-Rollups*. URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/ (visited on 2021-10-10).

[Eth20] Ethereum Foundation. *Design Rationale*. Ethereum Wiki. 2020. URL: https://eth.wiki/fundamentals/design-rationale (visited on 2021-08-16).

[Eth21] Ethereum Community. *The Eth2 upgrades*. ethereum.org. 2021. URL: https://ethereum.org/en/eth2 (visited on 2021-12-02).

[Eth23a] Ethereum Community. *The Merge*. ethereum.org. 2023. URL: https://ethereum.org/en/roadmap/merge/ (visited on 2023-09-25).

[Eth23b] Ethereum Foundation. *Zero-Knowledge Rollups*. ethereum.org. 2023. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups/ (visited on 2023-09-14).

[Fan21] Tobias Fan. *Ethereum 2.0 For Dummies — Part 2: How Does Staking Actually Work?* Coinmonks. 2021. URL: https://medium.com/coinmonks/ethereum-2-0-for-dummies-part-2-how-does-staking-actually-work-96bb714e4ad4 (visited on 2021-10-10).

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. "Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 308–317 vol.1. DOI: 10.1109/FSCS.1990.89549.

[FSS+20] Philipp Frauenthaler, Marten Sigwart, Christof Spanring, Michael Sober, and Stefan Schulte. "ETH Relay: A Cost-efficient Relay for Ethereum-based Blockchains". In: *2020 IEEE International Conference on Blockchain (Blockchain)*. 2020, pages 204–213. DOI: 10.1109/Blockchain50366.2020.00032.

[Geo16] Harris V. Georgiou. *On the Optimality of Ternary Arithmetic for Compactness and Hardware Design*. 2016. arXiv: 1611.03715 [cs].

[GGJ22] Garvit Goel, Rahul Ghangas, and Jinank Jain. *Verify Ed25519 Signatures Cheaply on Eth Using ZK-Snarks - Zk-s[Nt]Arks*. Ethereum Research. 2022. URL: https://ethresear.ch/t/verify-ed25519-signatures-cheaply-on-eth-using-zk-snarks/13139 (visited on 2023-09-27).

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Advances in Cryptology – EUROCRYPT 2013*. Lecture Notes in Computer Science. Springer, 2013, pages 626–645. DOI: 10.1007/978-3-642-38348-9_37.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. *Updatable and Universal Common Reference Strings with Applications to Zk-SNARKs*. 280. 2018. URL: http://eprint.iacr.org/2018/280 (visited on 2021-06-27).

[GKO20] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. *Zendoo: A Zk-SNARK Verifiable Cross-Chain Transfer Protocol Enabling Decoupled and Decentralized Sidechains*. 2020. arXiv: 2002.01847 [cs].

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pages 519–535. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi (visited on 2023-09-27).

[GM17] Jens Groth and Mary Maller. *Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs*. 540. 2017. URL: https://eprint.iacr.org/2017/540 (visited on 2021-01-03).

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pages 186–208. DOI: 10.1137/0218012.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design". In: *27th Annual Symposium on Foundations of Computer Science (Sfcs 1986)*. 1986, pages 174–187. DOI: 10.1109/SFCS.1986.47.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems". In: *Journal of the ACM* 38.3 (1991), pages 690–728. DOI: 10.1145/116825.116852.

[Goe20] Christopher Goes. *The Interblockchain Communication Protocol: An Overview*. 2020. URL: https://github.com/cosmos/ibc/raw/old/papers/2020-05/build/paper.pdf (visited on 2021-10-28).

[Gol01] Oded Goldreich. *Foundations of Cryptography: Volume 1: Basic Tools*. Volume 1. Cambridge: Cambridge University Press, 2001. DOI: 10.1017/CBO9780511546891.

[Gol04] Oded Goldreich. "Zero-Knowledge Twenty Years after Its Invention". In: (2004). URL: http://www.wisdom.weizmann.ac.il/~oded/PSX/zk-tut02v4.pdf (visited on 2021-06-07).

[Gro16] Jens Groth. *On the Size of Pairing-Based Non-Interactive Arguments*. 260. 2016. URL: https://eprint.iacr.org/2016/260 (visited on 2021-01-03).

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-Bases for Oecumenical Noninteractive Arguments of Knowledge*. 953. 2019. URL: http://eprint.iacr.org/2019/953 (visited on 2021-06-26).

[Hed19] Hedera. *How Is Hedera's Proxy Staking Different from DPoS?* Hedera Help. 2019. URL: https://help.hedera.com/hc/en-us/articles/360000665318-How-is-Hedera-s-proxy-staking-different-from-DPoS- (visited on 2021-05-30).

[HGT+22] Jonathan Heiss, Elias Grünewald, Stefan Tai, Nikolas Haimerl, and Stefan Schulte. “Advancing Blockchain-Based Federated Learning through Verifiable off-Chain Computations”. In: *2022 IEEE International Conference on Blockchain (Blockchain)*. 2022, pages 194–201. DOI: 10.1109/Blockchain55522.2022.00034.

[ICF20] Interchain Foundation. *Application-Specific Blockchains*. Cosmos Network. 2020. URL: https://docs.cosmos.network/master/intro/why-app-specific.html (visited on 2021-11-11).

[IF21a] IOTA Foundation. *Chrysalis Documentation*. 2021. URL: https://chrysalis.docs.iota.org/ (visited on 2021-05-29).

[IF21b] IOTA Foundation. *IOTA x Shimmer x Assembly*. IOTA Foundation Blog. 2021. URL: http://blog.iota.org/iota-shimmer-assembly/ (visited on 2022-01-18).

[IF21c] IOTA Foundation. *The New IOTA Client Libraries: Harder, Better, Faster, Stronger*. IOTA Foundation Blog. 2021. URL: https://blog.iota.org/the-new-iota-client-libraries-harder-better-faster-stronger/ (visited on 2022-01-20).

[IF21d] IOTA Foundation. *Towards Full Decentralization with IOTA 2.0*. IOTA Foundation Blog. 2021. URL: http://blog.iota.org/path-towards-full-decentralization-with-iota-2-0/ (visited on 2021-05-29).

[IF22a] IOTA Foundation. *Introduction to the Wasm VM for IOTA Smart Contracts*. IOTA Wiki. 2022. URL: https://wiki.iota.org/smart-contracts/guide/wasm_vm/intro (visited on 2022-03-20).

[IF22b] IOTA Foundation. *Wasm Gas Usage Tests*. GitHub Wasp repository. 2022. URL: https://github.com/iotaledger/wasp/tree/03e00b5ef7/contracts/wasm/gascalibration (visited on 2023-03-16).

[IF23] IOTA Foundation. *IOTA’s Stardust Upgrade and the Evolution of $IOTA Tokenomics*. IOTA Foundation Blog. 2023. URL: http://blog.iota.org/stardust-upgrade-iota-tokenomics/ (visited on 2023-09-25).

[IOT22] IOTA Foundation. *The Tangle | IOTA Wiki*. 2022. URL: https://wiki.iota.org/learn/about-iota/tangle/ (visited on 2023-03-19).

[Jos20] JosephC. *The Beacon Chain Ethereum 2.0 Explainer You Need to Read First*. ethos.dev. 2020. URL: https://ethos.dev/beacon-chain/ (visited on 2021-10-09).

[KB19] Jae Kwon and Ethan Buchman. *Cosmos Whitepaper - A Network of Distributed Ledgers*. Cosmos Network. 2019. URL: https://v1.cosmos.network/resources/whitepaper (visited on 2021-09-23).

[KJG+18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pages 583–598. DOI: 10.1109/SP.2018.000-5.

[KP19] Tommy Koens and Erik Poll. “Assessing Interoperability Solutions for Distributed Ledgers”. In: *Pervasive and Mobile Computing* 59 (2019). DOI: 10.1016/j.pmcj.2019.101079.

[KS22] Yaşanur Kayıkcı and Nachiappan Subramanian. “Blockchain Interoperability Issues in Supply Chain: Exploration of Mass Adoption Procedures”. In: *Big Data and Blockchain for Service Operations Management*. Studies in Big Data. Springer International Publishing, 2022, pages 309–328. DOI: 10.1007/978-3-030-87304-2_13.

[KSF+21] Benjamin Körbel, Marten Sigwart, Philip Frauenthaler, Michael Sober, and Stefan Schulte. *Blockchain-Based Result Verification for Computation Offloading*. 2021. DOI: 10.48550/arXiv.2110.11090. preprint.

[LeM18] Colin LeMahieu. *Nano: A Feeless Distributed Cryptocurrency Network*. 2018. URL: https://content.nano.org/whitepaper/Nano_Whitepaper_en.pdf (visited on 2021-04-11).

[LLW21] Eric Lombrozo, Johnson Lau, and Pieter Wuille. *BIP141 - Segregated Witness (Consensus Layer)*. 2021. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (visited on 2021-10-27).

[Mac22] Richard MacManus. *Solana Uses Rust to Pull in Developers and Avoid Copypasta*. The New Stack. 2022. URL: https://thenewstack.io/solana-rust-developers/ (visited on 2022-03-22).

[Mal19] Mary Maller. “Practical Zero-Knowledge Arguments from Structured Reference Strings”. PhD thesis. UCL (University College London), 2019. URL: https://discovery.ucl.ac.uk/id/eprint/10075582/ (visited on 2021-06-27).

[Mar19] Gautier Marin. *Why Application-Specific Blockchains Make Sense*. 2019. URL: https://blog.cosmos.network/why-application-specific-blockchains-make-sense-32f2073bfb37 (visited on 2023-05-27).

[Mar21] Thibault Martinez. *White Flag Ordering - Tangle Improvement Proposal (TIP) 2*. 2021. URL: https://github.com/iotaledger/tips/blob/5386420a65/tips/TIP-0002/tip-0002.md (visited on 2022-03-02).

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. New York, NY, USA: Association for Computing Machinery, 2019, pages 2111–2128. DOI: 10.1145/3319535.3339817.

[Mon] Monero. *Moneropedia: Bulletproofs*. getmonero.org, The Monero Project. URL: https://www.getmonero.org//resources/moneropedia/bulletproofs.html (visited on 2021-07-15).

[Moo19] Hans Moog. *Atomic Transfers / Transactions Instead of Bundles*. IOTA.cafe. 2019. URL: https://iota.cafe/t/atomic-transfers-transactions-instead-of-bundles/318 (visited on 2021-05-29).

[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". In: *IEEE Access* 10 (2022), pages 105807–105842. DOI: 10.1109/ACCESS.2022.3211422.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 2021-10-11).

[Nana] Nano Foundation. *Living Whitepaper*. URL: https://docs.nano.org/what-is-nano/living-whitepaper/ (visited on 2021-05-12).

[Nanb] Nano Foundation. *Nano RPC Protocol Documentation*. URL: https://docs.nano.org/commands/rpc-protocol/#confirmation_quorum (visited on 2021-05-12).

[Nan20] Nano Foundation. *The Incentives to Run a Node*. Medium. 2020. URL: https://blog.nano.org/the-incentives-to-run-a-node-ccc3510c2562 (visited on 2021-11-20).

[NYI+20] Ken Naganuma, Masayuki Yoshino, Atsuo Inoue, Yukinori Matsuoka, Mineaki Okazaki, and Noboru Kunihiro. "Post-Quantum Zk-SNARK for Arithmetic Circuits Using QAPs". In: *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. 2020, pages 32–39. DOI: 10.1109/AsiaJCIS50894.2020.00017.

[PB17] Joseph Poon and Vitalik Buterin. *Plasma: Scalable Autonomous Smart Contracts*. 2017. URL: https://plasma.io/plasma.pdf (visited on 2021-10-10).

[PB21] Serguei Popov and William J. Buchanan. "FPC-BI: Fast Probabilistic Consensus within Byzantine Infrastructures". In: *Journal of Parallel and Distributed Computing* 147 (2021), pages 77–86. DOI: 10.1016/j.jpdc.2020.09.002.

[PD16] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. 2016. URL: https://lightning.network/lightning-network-paper.pdf (visited on 2021-10-10).

[PGHR13] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. *Pinocchio: Nearly Practical Verifiable Computation*. 279. 2013. URL: https://eprint.iacr.org/2013/279 (visited on 2021-01-03).

[PMC+20] Serguei Popov, Hans Moog, Darcy Camargo, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, Olivia Saa, William Sanders, Luigi Vigneri, Wolfgang Welz, and Vidal Attias. *The Coordicide*. IOTA Foundation. 2020. URL: https://files.iota.org/papers/20200120_Coordicide_WP.pdf (visited on 2020-12-01).

[POK19] Seongjoon Park, Seounghwan Oh, and Hwangnam Kim. “Performance Analysis of DAG-Based Cryptocurrency”. In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2019, pages 1–6. DOI: 10.1109/ICCW.2019.8756973.

[Pol21a] Polkadot. *Availability and Validity · Polkadot Wiki*. 2021. URL: https://wiki.polkadot.network/docs/learn-availability (visited on 2021-10-06).

[Pol21b] Polkadot. *Cross-Consensus Message Format (XCM) · Polkadot Wiki*. 2021. URL: https://wiki.polkadot.network/docs/learn-crosschain (visited on 2021-10-06).

[Pol21c] Polkadot. *Polkadot and Cosmos · Polkadot Wiki*. 2021. URL: https://wiki.polkadot.network/docs/learn-comparisons-cosmos (visited on 2021-10-06).

[Pop18] Serguei Popov. *The Tangle*. 2018. URL: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf (visited on 2020-12-01).

[QQQ+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. “How to Explain Zero-Knowledge Protocols to Your Children”. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Lecture Notes in Computer Science. Springer, 1990, pages 628–631. DOI: 10.1007/0-387-34805-0_60.

[Qua20] QuarkChain. *How Heterogeneous Sharding Empowers Enterprise*. QuarkChain Official. 2020. URL: https://medium.com/quarkchain-official/how-heterogeneous-sharding-empowers-enterprise-e1ca05131009 (visited on 2021-09-08).

[Rei16] Christian Reitwiessner. *zkSNARKs in a Nutshell*. Etherum Foundation Blog. 2016. URL: https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/ (visited on 2021-06-07).

[RFC6962] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962. Internet Engineering Task Force, 2013. 27 pages. DOI: 10.17487/RFC6962.

[Rog21] Gal Rogozinski. *Uniform Random Tip Selection*. IOTA Protocol RFCs. 2021. URL: https://github.com/iotaledger/tips/blob/7d19cf3894/text/0008-uniform-random-tip-selection/0008-uniform-random-tip-selection.md (visited on 2021-08-25).

[Set19] Srinath Setty. *Spartan: Efficient and General-Purpose zkSNARKs without Trusted Setup*. 550. 2019. URL: https://eprint.iacr.org/2019/550 (visited on 2021-01-02).

[Set21] Srinath Setty. *Spartan: High-Speed zkSNARKs without Trusted Setup*. Github. 2021. URL: https://github.com/microsoft/Spartan (visited on 2021-07-25).

[SKSB19] Martin Schanzenbach, Thomas Kilian, Julian Schütte, and Christian Banse. "ZKlaims: Privacy-Preserving Attribute-Based Credentials Using Non-Interactive Zero-Knowledge Techniques". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications* (2019), pages 325–332. DOI: 10.5220/0007772903250332.

[Ten] Tendermint Inc. *What Is Cosmos?* Cosmos Network. URL: https://v1.cosmos.network/intro (visited on 2023-05-21).

[Tod14] Peter Todd. *Tree Chains*. 2014. URL: https://github.com/petertodd/tree-chains-paper (visited on 2023-03-19).

[TSB19] Jason Teutsch, Michael Straka, and Dan Boneh. *Retrofitting a Two-Way Peg between Blockchains*. 2019. DOI: 10.48550/arXiv.1908.03999.

[TSH22] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. "Blockchain Scaling Using Rollups: A Comprehensive Survey". In: *IEEE Access* 10 (2022), pages 93039–93054. DOI: 10.1109/ACCESS.2022.3200051.

[WD22] Martin Westerkamp and Maximilian Diez. *Verilay: A Verifiable Proof of Stake Chain Relay*. 2022. arXiv: 2201.08697 [cs].

[WE20] Martin Westerkamp and Jacob Eberhardt. "zkRelay: Facilitating Sidechains Using zkSNARK-based Chain-Relays". In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. 2020, pages 378–386. DOI: 10.1109/EuroSPW51379.2020.00058.

[Web21] Web3 Foundation. *Polkadot Wiki*. 2021. URL: https://wiki.polkadot.network/docs/getting-started (visited on 2021-09-22).

[Wel20] Wolfgang Welz. *Message PoW*. IOTA Protocol RFCs. 2020. URL: https://github.com/Wollac/protocol-rfcs/blob/e00bfce106/text/0024-message-pow/0024-message-pow.md (visited on 2021-08-26).

[Wel21] Wolfgang Welz. *Milestone Merkle Validation - Tangle Improvement Proposal (TIP) 4*. 2021. URL: https://github.com/iotaledger/tips/blob/c2485e8177/tips/TIP-0004/tip-0004.md (visited on 2022-04-02).

[WMM+23] Linus Wagner, Maximilian Mayer, Andrea Marino, Alireza Soldani Nezhad, Hugo Zwaan, and Ivano Malavolta. "On the Energy Consumption and Performance of WebAssembly Binaries across Programming Languages and Runtimes in IoT". In: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. EASE '23. New York, NY, USA: Association for Computing Machinery, 2023, pages 72–82. DOI: 10.1145/3593434.3593454.

[Woo16] Gavin Wood. *Polkadot: Vision For A Heterogeneous Multi-Chain Framework*. 2016. URL: https://github.com/polkadot-io/polkadotpaper/raw/master/PolkaDotPaper.pdf (visited on 2021-09-21).

[Woo21a] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Version Istanbul 6ef6062. 2021. URL: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 2021-10-12).

[Woo21b] Gavin Wood. *XCM: The Cross-Consensus Message Format*. Polkadot Network. 2021. URL: https://medium.com/polkadot-network/xcm-the-cross-consensus-message-format-3b77b1373392 (visited on 2021-10-06).

[WSNH19] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. *SoK: Sharding on Blockchain*. 1178. 2019. URL: http://eprint.iacr.org/2019/1178 (visited on 2021-02-21).

[WYCX20] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. *SoK: Diving into DAG-Based Blockchain Systems*. 2020. arXiv: 2012.06128 [cs].

[YWY+20] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. "Survey: Sharding in Blockchains". In: *IEEE Access* 8 (2020), pages 14155–14181. DOI: 10.1109/ACCESS.2020.2965147.

[Zcaa] Zcash. *How It Works*. URL: https://z.cash/technology/ (visited on 2021-06-28).

[Zcab] Zcash. *Parameter Generation*. URL: https://z.cash/ko_KR/technology/paramgen/ (visited on 2021-07-12).

[Zcac] Zcash. *What Are Zk-SNARKs?* URL: https://z.cash/technology/zksnarks/ (visited on 2021-06-26).

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. "vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pages 863–880. DOI: 10.1109/SP.2017.43.

[ZHZB20] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. "Solutions to Scalability of Blockchain: A Survey". In: *IEEE Access* 8 (2020), pages 16440–16455. DOI: 10.1109/ACCESS.2020.2967218.

[ZKP19] ZKProof. *ZKProof Community Reference. Version 0.2*. 2019. URL: https://docs.zkproof.org/reference.pdf (visited on 2021-06-27).

[ZoK] ZoKrates. *ZoKrates Documentation*. URL: https://zokrates.github.io/ (visited on 2021-07-25).

[ZWW+21] Shuyu Zheng, Haoyu Wang, Lei Wu, Gang Huang, and Xuanzhe Liu. "VM Matters: A Comparison of WASM VMs and EVMs in the Performance of Blockchain Smart Contracts". 2021. arXiv: 2012.01032 [cs].

[ZY19] Liangrong Zhao and Jiangshan Yu. "Evaluating DAG-Based Blockchains for IoT". In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pages 507–513. DOI: 10.1109/TrustCom/BigDataSE.2019.00074.