Advancing Blockchain Interoperability: Protocols for Seamless Communication and Transactions Across Blockchain Networks

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation (kumulativ)

von
Michael Sober

aus
Bruck an der Mur, Österreich

2025

Date of Oral Examination	January 23, 2025
Chair of Examination Board	Prof. Dr.-Ing. Gerhard Bauch
First Reviewer	Prof. Dr.-Ing. Stefan Schulte
Second Reviewer	Prof. Dr.-Ing. Stefan Tai

DOI 10.15480/882.14500
ORCID 0000-0002-9612-9022

Acknowledgment

First and foremost, I would like to thank my friends and family who accompanied me on this journey. Their support and belief in me motivated me to get the best out of myself.

Further, I sincerely appreciate my colleagues and co-authors for their time and commitment to doing great research. Your expertise, feedback, and moral support were essential to the success of this thesis. It was a pleasure working with such great people.

I am deeply grateful to my supervisor, Stefan Schulte, for his insightful and valuable feedback. His guidance and expertise have been essential to my personal development as a researcher and motivated me to strive for excellence.

Finally, I would like to thank the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, as well as the Christian Doppler Research Association for financing my research.

Abstract

Missing interoperability is one of the major challenges limiting the application and innovation of blockchain technology. Many blockchain networks operate in isolation and cannot seamlessly interact with other blockchains. Consequently, users and applications are bound to a single blockchain ecosystem and cannot benefit from the recent advancements of the consistently evolving blockchain landscape. Unfortunately, existing blockchain interoperability solutions are resource-intensive or require tailored solutions for specific blockchains. Hence, there is a strong need to bridge the gaps between different blockchains by establishing lightweight and flexible solutions for cross-blockchain communication and creating application-level cross-blockchain protocols to foster a more connected and efficient blockchain landscape.

Therefore, this thesis presents our recent advances in blockchain interoperability, including novel approaches for cross-blockchain communication, token transfers, and smart contract calls. We propose two cross-blockchain oracles with different off-chain aggregation mechanisms based on threshold signatures and Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs). Additionally, we integrate blockchain technology directly into a Distributed Key Generation (DKG) protocol to improve transparency and add accountability and crypto-economic incentives to encourage honest behavior. Further, we present two cross-blockchain token transfer protocols operating on different blockchain layers and introduce a framework for asynchronous cross-blockchain smart contract calls providing the foundation for cross-blockchain Decentralized Applications (DApps).

Contents

List of Figures

List of Abbreviations

CCIP Cross-Chain Interoperability Protocol

DApp Decentralized Application

DeFi Decentralized Finance

DEX Decentralized Exchange

DKG Distributed Key Generation

EdDSA Edwards-curve Digital Signature Algorithm

EVM Ethereum Virtual Machine

IBC Inter-Blockchain Communication

IoT Internet of Things

NiPoPoW Non-interactive Proof of Proof of Work

PoS Proof of Stake

PoW Proof of Work

RPC Remote Procedure Call

VSS Verifiable Secret Sharing

XCM Cross-Consensus Message Format

ZKP Zero-Knowledge Proof

zk-SNARK Zero-Knowledge Succinct Non-interactive Argument of Knowledge

Part I

Introduction

1

Introduction

Over the past decade, blockchain technology has emerged as the key driver of Web 3.0 due to its decentralized nature [HJM+23]. A blockchain acts as a decentralized ledger that records transactions across a peer-to-peer network of nodes, ensuring integrity, decentralization, immutability, pseudonymity, and other key features. While blockchains still primarily form the backbone of many cryptocurrencies, e.g., Bitcoin [Nak08], recent advancements in blockchain technology have increased interest in other application areas such as healthcare [HKG+20], supply chain management [KFP19], the Internet of Things (IoT) [FF18], and others [SHM+22]. These advancements include, amongst others, the introduction of smart contracts in the second generation of blockchains, such as Ethereum [Woo14]. Smart contracts are deterministic programs stored on a blockchain and provide the means for creating Decentralized Applications (DApps). These applications do not depend on a centralized server but on a blockchain network.

While the rapid progress and constant development in blockchain technology have led to novel solutions that offer additional functionalities and other useful properties, they also resulted in a fragmented ecosystem [SSFB19]. Each application area has specific requirements that a single blockchain often cannot fulfill. Therefore, research and industry tend to create new blockchains tailored to specific use cases and requirements. Unfortunately, interoperability is often not considered when designing or implementing new blockchains [BVGC21; WWC23]. While some blockchains provide interoperability [Woo16; KB20; Fou22], most are closed systems that can not interact with external systems, e.g., web applications or other blockchains. Therefore, users who use a particular blockchain face considerable obstacles in using multiple or migrating to other blockchains, often resulting in vendor lock-ins, where a user is stuck

in its initially selected blockchain. However, using different blockchains that offer new, unique capabilities, security guarantees, or applications is often desirable. Additionally, smart contracts deployed on a blockchain can only alter the state of its hosting blockchain and can not interact with smart contracts on other blockchains, limiting innovation, cooperation, and functionality since applications built on top of different blockchains are hard to integrate with each other.

Establishing interoperability between different blockchains allows for the creation of cross-blockchain DApps, facilitating the reuse of existing projects and collaboration [WWC23]. Additionally, interoperable blockchains enable users and developers to select from various blockchain platforms, choosing the one that best suits their specific needs. Multiple organizations using different blockchains can seamlessly integrate their systems, building upon interconnected blockchains to cooperate. Further, connecting different blockchains allows users to freely transfer their tokens to other blockchains without committing to a specific blockchain to use novel features and applications of multiple blockchains.

While research and development have already led to better interoperability between blockchains, there are problems that still need to be solved. Current blockchain interoperability solutions most prominently include notary schemes, blockchain relays, and blockchains of blockchains [BVGC21; WWC23]. A notary scheme or oracle is a trusted entity or group of entities that acts as a bridge between two blockchains. The oracle retrieves data from a source blockchain, attests to the data's correctness, and submits it to a target blockchain [SSSS21]. Blockchain relays are usually smart contracts that act as light clients to verify the rules of the source blockchain within the target blockchain [FSS+20; WD22]. Blockchains of blockchains, such as Polkadot [Woo16] and Cosmos [KB20], form a network where a main blockchain connects multiple blockchains, which can access the main chain to verify the connected blockchains. While these solutions provide the means to enable interoperability between heterogeneous and homogenous blockchains, they exhibit different advantages and disadvantages.

Blockchain relays and blockchains of blockchains often require tailored solutions for each blockchain since many blockchains use different consensus mechanisms, governance, tokenomics, security, and programmability. These restrict the flexibility of many blockchain interoperability solutions, making them hard to maintain. Further, these solutions are usually very heavyweight, i.e., the resource overhead associated with their usage is very high and constitutes a major obstacle to applying these solutions in real-world settings. Cross-blockchain oracles, on the other hand, are more flexible and lightweight but less decentralized. While cross-blockchain oracles already provide more flexibility and have less resource overhead, there is still potential to improve the cost-efficiency of aggregation mechanisms by using verifiable off-chain computations. Therefore, there is a strong need to create lightweight

cross-blockchain protocols, including cross-blockchain communication and higher-level protocols built on top, including protocols for cross-blockchain token transfers and smart contract calls.

For that, it is necessary to move computation and storage off-chain using various cryptography constructions. More specifically, applying threshold cryptosystems [Des94] and Zero-Knowledge Proofs (ZKPs) [GMR19] is a promising approach to creating more efficient cross-blockchain protocols. By moving computation and storage off-chain, the protocols consume fewer blockchain resources, leading to lower transaction fees [KSF+21]. More specifically, ZKPs allow for verifying arbitrary computations [GMW91; PHGR16]. Hence, it is possible to execute certain parts of cross-blockchain protocols off-chain, while a smart contract only needs to verify a small proof to verify the computations' correctness. These mechanisms are commonly utilized in Layer 2 blockchain scaling solutions. These solutions are constructed on top of Layer 1 blockchains, i.e., the core protocol, to reduce the network load, enable faster transaction processing, and lower costs. Additionally, threshold cryptosystems allow for distributing trust among a group of entities to create digital signatures, e.g., threshold signatures, while the verification complexity does not increase with the group size. This property is particularly useful when the verifier is a smart contract to decrease the costs of transaction execution.

In this thesis, we present our contributions to advancing blockchain interoperability. More specifically, we present cross-blockchain communication mechanisms using oracles. A cross-blockchain oracle enables two blockchains to exchange arbitrary data. A committee of oracle nodes attests to the integrity of the exchanged data by executing a voting mechanism and aggregating the results. We propose two different aggregation mechanisms for the oracles using threshold signatures and Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) to move most of the computations off-chain, which reduces the costs of executing the smart contracts. After that, we present two cross-blockchain token transfer protocols that rely on the aforementioned cross-blockchain communication mechanisms. First, we define the requirements of cross-blockchain token transfers and design a protocol using Layer 1. Second, we design another cross-blockchain token transfer protocol that can execute the same steps on Layer 2, which is more cost-efficient. We also present a framework for asynchronous cross-blockchain smart contract calls that allows two smart contracts deployed on different blockchains to call each other's functions and retrieve the results. Finally, we explore blockchain-based Distributed Key Generation (DKG) to improve the integration of a DKG protocol in our first oracle solution, which uses threshold signatures. For each contribution, we provide the respective research paper in this thesis.

Structure

The remainder of this thesis is structured as follows:

Part 1: Introduction. The remainder of Part I provides an introduction to blockchains and blockchain interoperability. Further, we give an overview of the research directions, underlining the importance of this field.

Part 2: Contributions. In Part II, we delve into the novel contributions of this work. Moreover, we provide a comprehensive overview of the state-of-the-art and highlight the distinctive aspects of our research findings.

Part 3: Blockchain Interoperability. In the final Part III, we present our original research papers. These papers cover cross-blockchain communication with oracles, token transfers, smart contract calls, and blockchain-based DKG.

2

Preliminaries

This section presents the general concept of blockchain technology, including blockchain basics, types, consensus mechanisms, and smart contracts. Additionally, we discuss blockchain interoperability.

2.1 Blockchain Technology

This section provides a general introduction to blockchain technology and lays the foundation for understanding the background for the included research papers. Initially, we explain blockchain basics and the categorization of blockchains. Afterward, we discuss consensus mechanisms and smart contracts.

2.1.1 Blockchain Basics

Blockchain technology is an emerging technology that allows for creating a decentralized online ecosystem, moving the Internet closer to Web 3.0. The first blockchain application was Bitcoin [Nak08], a cryptocurrency that does not rely on a trusted third party. A blockchain is a distributed ledger of transactions managed by a peer-to-peer network of nodes maintaining a shared state. Additionally, the term blockchain refers to the underlying data structure used to store the ledger. Each node in the network stores a copy of the ledger and runs a consensus mechanism to reach an agreement on the network's current state. The nodes store the ledger's state using a linked list of

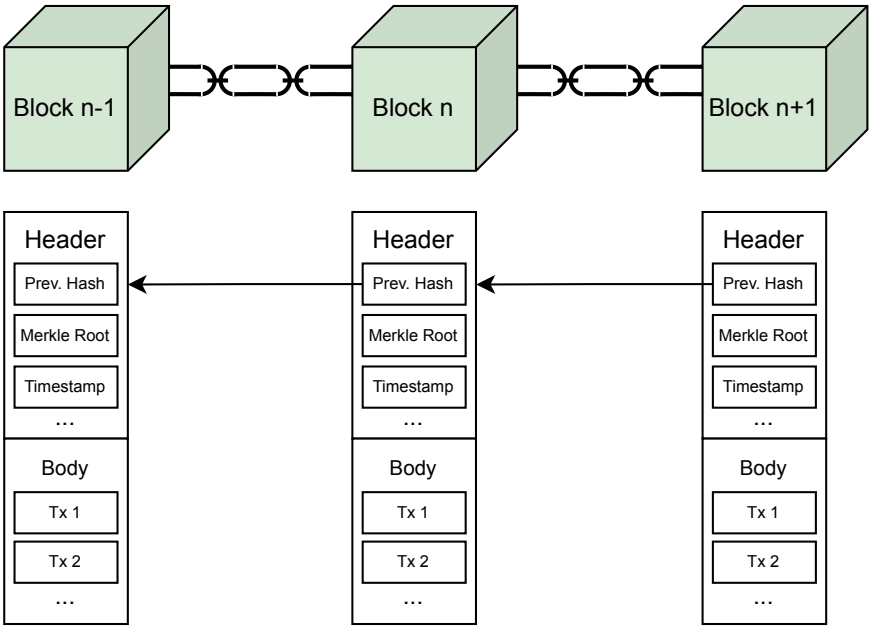

Figure 2.1: Basic blockchain structure.

blocks connected by hash pointers, which connect each block to its predecessor until the genesis block, i.e., the first block of the blockchain.

A block consists of a block header and body (see Fig. 2.1). The block header usually includes crucial information for the network's operation and security, such as the previous block hash, Merkle root of transactions, timestamp, and consensus-specific parameters. The body of a block contains the transactions selected by the block proposer during block creation. The transactions describe specific actions to advance the state of the blockchain and generally consist of a sender, receiver, amount of cryptocurrency, and additional data depending on the underlying blockchain. Each transaction included in a block is stored in a Merkle tree where the corresponding Merkle root is part of the block header. The application of Merkle trees for storing transactions enables the efficient inclusion verification of transactions in a block.

Blockchains exhibit many valuable properties, including immutability, pseudonymity, transparency, decentralization, and security [GY22]. The hash pointers ensure that altering the data stored in a blockchain is infeasible as long as the cryptographic hash functions remain secure and sufficient network participants follow the consensus protocol. Changing the data in a block requires altering all subsequent blocks since a slight change in data already results in a completely different hash value.

Further, participants in the network can generate new identities simply by creating new public-private key pairs under which the participants operate, not directly tied to real-world identities. Privacy-preserving blockchains even

allow for complete anonymity by breaking the link between senders and receivers of transactions, preserving the users' privacy [BCR+19].

Further, every participant in the network can view the whole state of the blockchain, allowing everyone to audit transactions and increasing trust in the system. Most blockchains are also decentralized, whereby the degree of decentralization varies depending on the blockchain. This decentralization removes the necessity of a trusted third party, removing the potential risks of a single point of failure and increasing the resilience against specific attacks. Finally, blockchains apply multiple cryptographic primitives, such as cryptographic hash functions and digital signatures, which ensure the system's security by preventing tampering with and ensuring the stored data's integrity.

2.1.2 Blockchain Types

Blockchains constantly evolve due to the demands of various stakeholders in different application domains [BKN+21]. Currently, blockchains can be roughly categorized into the following categories: public, private, consortium, and hybrid blockchains. These categories mainly differentiate blockchains based on who is allowed to participate in the system.

Public blockchains such as Bitcoin and Ethereum allow anyone to join the network and participate in executing the protocol. They usually exhibit a high degree of decentralization and enable everyone to view the current state of the blockchain. On the other hand, private blockchains are managed by a central authority and often used by organizations where trust, privacy, and confidentiality are important [ABB+18]. Private blockchains are not decentralized but provide better performance. Consortium blockchains are similar to private blockchains but are used by multiple mutually distrustful organizations that want to distribute trust. Only a group of pre-selected participants are authorized to create new blocks and view the state of the blockchain. Finally, hybrid blockchains combine public, private, or consortium blockchains, where organizations can set up a private or consortium blockchain alongside a public blockchain to profit from the advantages of both networks.

While this rough categorization of blockchains is a starting point, they also differ in other aspects, such as consensus mechanism, governance, programmability, speed, and cost-effectiveness [ASZ22]. Each of these properties holds varying degrees of relevance across industries and application areas. In this work, our primary focus is on blockchain interoperability solutions for public blockchains. This choice is driven by the complexity of preserving properties like decentralization and security in such environments.

2.1.3 Consensus

An integral part of blockchains is the applied consensus mechanism. The participants must reach a consistent shared state, i.e., maintain the canonical

blockchain in the peer-to-peer network [WHH+19] under the condition of Byzantine failures [CL02]. The two major consensus mechanisms applied nowadays are Proof of Work (PoW) and Proof of Stake (PoS), which are used by Bitcoin [Nak08] and Ethereum [Woo14], respectively.

In PoW, the participants must spend considerable computational power to append new blocks to the blockchain [Nak08]. A participant who creates a new block and wants to append it to the blockchain must solve a cryptographic puzzle. The process of solving this puzzle is also called mining. Mining requires participants, also called miners, to repeatedly compute a block's hash value until it falls below a particular target value defined by the network's difficulty parameter. For that, miners change the so-called nonce of a block to find a hash value that falls below the target. Since cryptographic hash functions exhibit specific properties [RS04] such as preimage resistance, second-preimage resistance, collision resistance, and puzzle-friendliness, the only way to solve this puzzle is by iteratively computing the hash values, which is computationally very expensive and often requires specialized hardware consuming a high amount of electricity [De 20]. However, miners receive a monetary reward in exchange for mining new blocks. Hence, the chance of appending a specific block to the blockchain depends on the miner's available computation power. This also means that if a participant can control more than 51% of the network's computational power, it has control over the whole network [GKW+16]. It allows the miner to double-spend, censor transactions, and reorganize the blockchain. However, executing such attacks is challenging and expensive, making it hard to execute on well-established blockchain networks such as Bitcoin due to the high network hash rate.

PoS follows a different approach, which does not rely on the computation power but on the amount of cryptocurrency a participant locks up as collateral [Sal21]. The amount of staked cryptocurrency is proportional to the probability of being selected as a block proposer and provides the right to act as a validator. The block proposer verifies transactions, creates new blocks, and adds them to the blockchain. In exchange, the block proposer receives the transaction fees as a reward for creating the block. This process is called minting or forging new blocks rather than mining since it involves considerably less computational power and electricity. Further, the protocol can hold misbehaving validators accountable and slash the locked collateral.

The consensus mechanisms used by different blockchain networks play a crucial role in blockchain interoperability since the applied interoperability mechanisms depend on the underlying blockchains. Blockchains communicating with each other must be able to verify each other's state to determine the integrity of the transferred data. For that, blockchain interoperability solutions need to be able to verify the consensus rules of the connected blockchains to determine if specific data is included and confirmed with overwhelming probability in the respective blockchains. Hence, it is an essential factor to consider when designing cross-blockchain protocols.

2.1.4 Smart Contracts

The second generation of blockchains allows for value transfers between different participants and the execution of smart contracts. Nick Szabo [Sza97] introduced the concept of smart contracts to create a computerized transaction protocol to execute the terms of a contract automatically. In blockchain technology, smart contracts are programs stored on the blockchain. The introduction of smart contracts enables the creation of DApps for finance [CB20], healthcare [HKG+20], the IoT [RMC+18], and others [WOY+19]. With the help of protocols for cross-blockchain smart contract calls [SSS+23], it is also possible to create DApps that utilize smart contracts communicating through the boundaries of multiple blockchains [WWC23].

Participants create transactions to call specific smart contract functions by providing the necessary call data specifying which function to call and the parameters. These transactions result in the execution of a smart contract by every network node, which needs to reach a consensus on the execution's result. Therefore, smart contracts must be deterministic to ensure the consistency of the ledger's state among all nodes. Further, different blockchains can provide various levels of programmability. For example, Bitcoin only provides a stack-based scripting system [Wik24], while Ethereum offers quasi-Turing-complete smart contracts [Woo14].

For the execution of smart contracts, Ethereum uses the Ethereum Virtual Machine (EVM), a stack machine that processes and executes transactions, including the execution of smart contracts [Woo14]. Developers can write Ethereum smart contracts using Solidity or Vyper, programming languages specifically designed for smart contracts. The source code is compiled to EVM bytecode, i.e., the native code for execution by the EVM. The EVM bytecode can also be expressed as opcodes, predefined instructions supported by the EVM.

Executing smart contracts and transactions requires resources from the participants who execute them. Hence, many blockchains require a fee for executing transactions to compensate the participants and protect the network from misuse. The incentive mechanisms and fee structures of blockchains depend on the underlying blockchains. Ethereum applies the concept of Gas, which assigns each opcode a specific amount of Gas to measure the computational power necessary to execute the operation [Woo14]. The transaction's sender must pay a fee, considering the computational power required to execute the transaction.

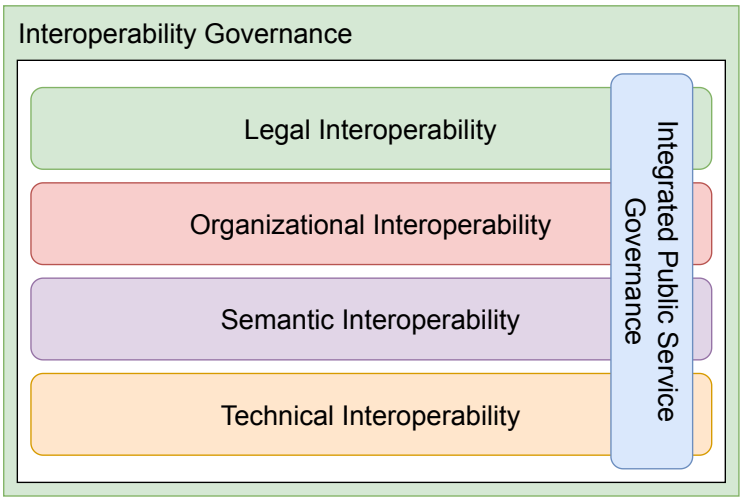

Figure 2.2: Interoperability layers according to the European Interoperability Framework [Com17].

2.2 Blockchain Interoperability

Interoperability refers to a system, device, or application's ability to interact seamlessly without further involvement from the user's side. This ability has several advantages: increased efficiency, scalability, cost savings, flexibility, and adaptability. Interoperability encompasses multiple levels, each of which considers different aspects [Com17]. These four layers include technical, semantic, legal, and organizational interoperability (see Fig. 2.2). Technical interoperability describes interoperability on the hardware and software levels, enabling systems to interoperate through a precise specification of standard interfaces and data formats. Semantic interoperability extends this notion such that the system must also support the same data structures and schemas. Further, the system must also understand the meaning of the exchanged data. Legal interoperability considers the usage of the systems across different jurisdictions, e.g., adhering to different regulations regarding the usage and storage of data. Finally, organizational interoperability ensures that organizations can effectively use a system in different contexts. Our work mainly focuses on the technical and semantic levels of blockchain interoperability.

Unfortunately, there is no clear definition of blockchain interoperability [BVGC21; WWC23]. Hence, we define blockchain interoperability as the ability of heterogeneous or homogenous blockchain networks to interact and exchange data through a cross-blockchain communication protocol. This protocol facilitates direct interactions and verifies the other blockchain's state, ensuring integrity, consistency, and security. While existing blockchains, e.g., Cosmos [KB20] and Polkadot [Woo16], already provide interoperability with other blockchains, this is relatively rare when looking at the whole blockchain landscape. Hence, there is a strong need for blockchain interoperability solu-

tions to bridge the gaps between different blockchains to enable an Internet of blockchains [SSFB19]. There is a lack of cross-blockchain communication protocols that allow for arbitrary data exchange and the protocols built on top. These protocols include, amongst others, cross-blockchain token transfer protocols and protocols for executing cross-blockchain smart contract calls. While the first allows for the transfer of tokens between distinct blockchains, the latter enables smart contracts deployed on different blockchains to call each other's functions directly. Hence, this work follows up on creating protocols to advance blockchain interoperability while paying attention to their applicability in the real world.

3

Research Directions

In this section, we discuss the research directions based on the previously mentioned shortcomings regarding blockchain interoperability in today's blockchain landscape. Initially, we discuss cross-blockchain communication and follow with a discussion of the higher-level protocols building on top of it, including cross-blockchain token transfers and smart contract calls. Finally, we discuss blockchain-based DKG.

3.1 Cross-Blockchain Communication

The foundation of blockchain interoperability lies in the capability to exchange arbitrary data across different blockchains and coordinate the execution of transactions using a cross-blockchain communication protocol [BVGC21; ZAZ+21]. These protocols enable the development of higher-level cross-blockchain protocols for cross-blockchain token transfers [SSF+23; SLS24] and smart contract calls [SSS+23]. For that, the involved blockchains rely on the ability to verify each other's state to ensure integrity and consistency. However, since most blockchains operate as closed systems with access limited to their own states, verifying the state of another blockchain requires a trusted third party [ZAZ+21]. The trusted third party allows the verification of the state of the communicating blockchains and attests to the correctness of the transferred data or execution of transactions. While cross-blockchain communication requires a trusted third party, it can be decentralized to align with the decentralized nature of blockchains.

The importance of cross-blockchain communication for enabling blockchain interoperability led to multiple approaches, including blockchain relays, no-

tary schemes, blockchains of blockchains, and others [BVGC21]. A major challenge lies in creating a decentralized, secure, and cost-efficient solution applicable to real-world scenarios. Unfortunately, many solutions are very heavyweight and are tailored to specific blockchains since every blockchain exhibits different consensus mechanisms, tokenomics, governance, and other properties. An essential factor for cross-blockchain communication is the applied consensus mechanism since cross-blockchain communication protocols must be able to verify the state of the other blockchains, including checking that specific data is included and confirmed with overwhelming probability. Hence, it is very costly if cross-blockchain communication protocols need to conduct these checks on-chain, e.g., a blockchain relay that uses a smart contract acting as a light client. While there has already been some effort to create more lightweight and cost-efficient solutions [FSS+20; WE20; XZC+22], there is still potential for further improvements in cost-efficiency and flexibility.

This thesis focuses on oracle-based solutions to cross-blockchain communication since these are less dependent on the underlying blockchains. A cross-blockchain oracle can act as a bridge between blockchains to transfer and attest to the correctness of data. These oracles can even be decentralized to distribute trust among multiple participants and preserve a certain degree of decentralization concerning the underlying blockchains. Compared to other solutions, an oracle only needs to conduct the above-mentioned checks off-chain and create an aggregated result, resulting in a flexible and lightweight approach to cross-blockchain communication.

3.2 Cross-Blockchain Token Transfers

Cross-blockchain token transfers are a particular aspect of blockchain interoperability, allowing users to transfer tokens to another blockchain [SSF+23]. Currently, users are, for the most part, only able to use their tokens on one particular blockchain, i.e., users can only exchange value on the blockchain hosting the token contract and not use any Decentralized Finance (DeFi) applications based on other blockchains. While atomic swaps [Her18] allow the decentralized exchange of tokens between blockchains, this approach does not constitute actual cross-blockchain token transfers since atomic swaps only exchange tokens in an atomic and trustless manner while the respective tokens remain on their blockchains. Further, atomic swaps always require a counterparty willing to exchange tokens. Another way to exchange tokens is through centralized exchanges. However, these contradict the main design philosophy of blockchains, which is complete decentralization. Hence, there is a strong need for cross-blockchain token transfer protocols to enhance liquidity, improve efficiency, and provide users access to diverse ecosystems. While there are already some attempts by research and industry to realize cross-blockchain token transfers [ZHL+19; PBHM20], there is still potential for improvement.

Ideally, cross-blockchain token transfers enable users to freely choose which blockchain they want to hold and use their assets. Users should not be tied to particular blockchains and should be able to hold different denominations of their tokens on multiple blockchains at the same time. If a new blockchain technology emerges and offers novel features, users should be able to transfer their tokens to this new blockchain, taking advantage of the novel capabilities. Several aspects play a critical role when transferring tokens between blockchains, such as consistency, security, decentralization, and cost-efficiency. Within this work, we explore different approaches to transferring tokens between blockchains while also preserving decentralization. Hereby, we focus on creating cross-blockchain token transfer protocols based on our previously mentioned cross-blockchain oracles. Additionally, we investigate the application of Layer 2 scaling solutions, such as rollups, to enhance the cost-efficiency of these transfers.

3.3 Cross-Blockchain Smart Contract Calls

Most blockchains are closed worlds with no capability to interact with external systems, including other blockchains. This restriction prevents more complex interactions between smart contracts deployed on different blockchains, e.g., cross-blockchain smart contract calls, since a smart contract can only modify its state on the hosting blockchain. Further, a smart contract can only be called by other smart contracts on the same hosting blockchain. The lack of possibilities to coordinate and ensure the consistency of smart contract calls across blockchains is a major problem. A cross-blockchain smart contract call protocol enables smart contracts hosted on different blockchains to call each other's functions and retrieve the execution result [SSS+23]. These protocols enable the creation of cross-blockchain DApps, which can utilize different smart contracts across multiple blockchains. Further, it allows developers to use the best blockchain for specific requirements and fosters code reuse. Cross-blockchain smart contract calls require further attention by research to improve on the shortcomings of already existing protocols. Many protocols rely on several trusted intermediaries to allow synchronous cross-blockchain smart contract calls [NSSB21; RR21]. Unfortunately, these protocols do not provide the necessary abstractions and only allow conducting synchronous cross-blockchain smart contract calls.

Therefore, we rely on blockchain relays for communication between blockchains to realize a framework for cross-blockchain smart contract calls. For that, we explore concepts from computer networks, such as Remote Procedure Calls (RPCs) for the application within blockchains. The smart contracts can call each other through stub contracts which abstract cross-blockchain communication and a standard message format for exchanging call and reply messages. Such a framework allows developers to create

cross-blockchain DApps without focusing on the underlying mechanisms to execute cross-blockchain smart contract calls.

3.4 Distributed Key Generation with Smart Contracts

DKG protocols allow multiple participants to generate a shared secret key, which can be used for threshold cryptosystems [Ped91]. Threshold cryptosystems play a very important role in blockchain technology as these systems divide the ability to, e.g., create digital signatures or encrypt and decrypt messages among a group of participants. A predetermined threshold number of participants need to collaborate to execute cryptographic operations. Distributing trust among multiple participants reduces the risks of relying on a central authority. Hence, threshold cryptosystems exhibit great potential in the application within blockchain networks, e.g., voting in consensus mechanisms where a threshold number of participants need to collaborate to verify and sign a new block [KMS20].

Naturally, these properties are also beneficial in cross-blockchain communication protocols using oracles where multiple participants must aggregate the retrieved data and attest to its correctness. A DKG protocol allows a committee of oracle nodes to generate a shared private key to create threshold signatures attesting to the data's integrity and submit it to the blockchain [SSSS21]. A smart contract on the blockchain can then verify the threshold signature to check that a threshold number of participants agree on the transferred data. However, a major problem is the execution of the DKG as the resulting public key must be available to the smart contract verifying the signature. Since the participants execute the DKG protocol off-chain, this requires additional measures to make the public key available to the smart contract. Hence, there is a need for a DKG protocol based on smart contracts where the smart contract has direct access to the resulting public key on-chain while simultaneously augmenting the DKG with crypto-economic incentives to ensure honest behavior. There are few attempts to create DKG protocols based on blockchains [SJSW19; SRMH21], introducing a considerable overhead through smart contracts. Therefore, exploring blockchain-based DKG protocols that use verifiable off-chain computations, e.g., through the application of ZKPs, is promising to reduce the costs.

3.5 Research Questions

Following the previously discussed research directions, we summarize the following research questions answered in this thesis:

RQ1: How can we utilize blockchain oracles to communicate arbitrary data between different blockchains to provide a foundation for higher-level cross-blockchain protocols while preserving decentralization?

RQ2: How can we transfer tokens between different blockchains in a decentralized and cost-efficient way?

RQ3: How can we realize smart contract calls across the boundaries of different blockchains to create decentralized cross-blockchain applications?

RQ4: How can we integrate existing DKG protocols with blockchains to ease the application in cross-blockchain protocols?

Part II

Contributions

4

Cross-Blockchain Protocols

This chapter introduces various cross-blockchain protocols. Initially, we present our contributions to cross-blockchain communication. Then, we discuss higher-level protocols allowing for cross-blockchain token transfers and smart contract calls.

4.1 Cross-Blockchain Communication

There are several solutions available for facilitating cross-blockchain communication. These solutions, for the most part, consist of blockchain relays, notary schemes, and blockchains of blockchains. Blockchain relays use a smart contract as a light client deployed on a target blockchain. The smart contract verifies the consensus of the source blockchain to verify that specific data is included and confirmed with overwhelming probability in the source blockchain. Currently, multiple relays, e.g., ETH Relay [FSS+20] or Verilay [WD22], which are relay solutions for Ethereum PoW and PoS, or relays using zk-SNARKs [WE20; XZC+22] are available. Further, some blockchains of blockchains including Cosmos and Polkadot, employ their own cross-blockchain communication protocols. Cosmos uses the Inter-Blockchain Communication (IBC) protocol [Goe20], and Polkadot uses the Cross-Consensus Message Format (XCM) with a message-passing protocol. Additionally, there are also cross-blockchain communication approaches based on oracles, i.e., notary schemes, including Chainlink [BCC+21] with its Cross-Chain Interoperability Protocol (CCIP) [Cha21] or other blockchain interoperability frameworks [AAA22]. Furthermore, there are various general oracle solutions [ABV+18; ARSS20] that connect blockchains with external systems.

These oracles usually distribute trust among a group of oracle nodes, which execute an aggregation mechanism to submit data to the target blockchain. The aggregation mechanism can be very costly to execute for a smart contract. Hence, optimizing the costs of executing the aggregation mechanism by moving the aggregation mechanism off-chain is essential for creating cost-efficient cross-blockchain oracles.

In our work [SSSS21], we present a cross-blockchain oracle that bridges different blockchains using an off-chain aggregation mechanism. The oracle retrieves arbitrary data from a source blockchain and submits it to a target blockchain. The oracle follows a decentralized design where a committee of oracle nodes consisting of an aggregator and multiple validators needs to collaborate and attest to the integrity of the transferred data.

More specifically, we define that a threshold number of oracle nodes must agree on whether the data is included and confirmed with overwhelming probability in the target blockchain before submitting it to the source blockchain. For that, the oracle nodes execute an off-chain aggregation mechanism based on threshold signatures to aggregate the different results and only submit the final result and a threshold signature to the target blockchain. The smart contract only needs to verify a single signature to verify the correctness of the submitted results. For that, the oracle nodes need to execute a DKG protocol to generate a distributed private key such that a threshold number of oracle nodes need to collaborate to use the private key. Each oracle node only has its private key share and shared public key to partially sign the retrieved data from the source blockchain. The validators retrieve the requested data from the source blockchain, verify that it is confirmed with overwhelming probability, sign it with their private key share, and send the partial signature and data to the current aggregator, which rotates based on a round-robin mechanism. Upon receiving a threshold number of responses with the same results from validators, the aggregator reconstructs the threshold signature and submits the result and the signature to the smart contract.

Further, we apply an incentive mechanism to ensure that aggregators and validators behave honestly, i.e., follow the protocol rules and receive a reward for contributing since they need to spend a certain amount of resources to execute the protocol. Therefore, a client needs to pay a fee consisting of a validator reward and an aggregator reward. However, only the aggregator submitting the result is rewarded since the threshold signatures do not disclose which validators contributed to the result. Further, since it is untrusted, the aggregator can not provide the list of validators. Therefore, an aggregator gets the aggregator reward and only has a chance of getting the validation reward. If an aggregator does not get the validation reward, the smart contract retains the reward, automatically increasing the possible validation reward for the next submission. Our results show that the oracle can submit data with almost constant costs independent of the number of oracle nodes and is more cost-efficient than relay solutions, depending on the request rate. However,

applying threshold signatures increases complexity by executing a DKG every time the committee of oracle nodes changes. Further, the anonymity of the validators that produce partial signatures hinders accountability.

Therefore, we present another cross-blockchain oracle, which strives for the same functionality of bridging different blockchains but with a different aggregation mechanism based on zk-SNARKs [SSS24]. Similar to our previous approach, the oracle relies on a committee of oracle nodes, including an aggregator and multiple validators. However, instead of storing the committee's account data on-chain, the smart contract stores the accounts in a Merkle tree and only stores the Merkle root. Further, instead of relying on threshold signatures to ensure that the oracle provides a correct answer, we apply zk-SNARKs to execute the complete aggregation mechanism, including signature verification and reward distribution off-chain.

During aggregation, the validators still retrieve the data from the source blockchain but only create a signature using the Edwards-curve Digital Signature Algorithm (EdDSA) before sending both to the current aggregator. After receiving enough responses from validators, an aggregator verifies the signatures, distributes rewards to each contributing validator, and creates a zk-SNARK to prove the correctness of these computations. After that, an aggregator only submits the result, the updated state root, and the proof to the smart contract. The smart contract verifies the constant-sized proof and, if it is valid, stores the new state root and data. Otherwise, it does not accept the submission.

Additionally, the added accountability enables the punishment of misbehaving validators. An aggregator can create a zk-SNARK proving that a validator returned a response that deviates from the majority answer to slash its stake. This mechanism further incentivizes honest behavior. While this approach does not require executing a complex DKG protocol and enables the accountability of validators, it requires substantial resources by the aggregator to generate the proof. Similar to our previous solution, it also requires only constant submission costs.

4.2 Cross-Blockchain Token Transfers

In the past years multiple cross-blockchain token transfer protocols have already been proposed. Many of these protocols involve two steps: burning tokens on the source blockchain and recreating tokens on the target blockchain. The major difference is how these protocols verify the burn step on the target blockchains. There are cross-blockchain token transfer protocols applying notary schemes [NKJ+23], blockchain relays [ZHL+19; SFS+21], Non-interactive Proofs of Proof of Work (NiPoPoWs) [KKZ20], or sidechain constructions using zk-SNARKs [GKO20]. Further, many protocols directly change the core blockchain to enable cross-blockchain token

transfers [VKD23]. Other approaches rely on synchronizing balances across multiple blockchains rather than burning and recreating the tokens [BSF+19] or using an auction model [LWM+21]. Many blockchains of blockchains approaches, e.g., Cosmos [KB20] and Polkadot [Woo16], also support cross-blockchain token transfers through their underlying cross-blockchain communication protocols. Additionally, atomic swaps [Her18; DH20] also allow for exchanging tokens across multiple blockchains. However, the tokens never leave the original blockchains. Unfortunately, many of these works do not cover all important parts, including the requirements, specification, and implementation of a cross-blockchain token transfer protocol. Further, exploring Layer 2 blockchain scaling solutions for cross-blockchain token transfers remains an unexplored approach with the potential to improve cost efficiency.

In [SSF+23], we present a protocol for cross-blockchain token transfers, which extends our base protocol presented in [SFS+21] by adding transfer confirmations. The presented protocol allows for transferring digital assets or tokens, i.e., a representation of value on a blockchain, from a source blockchain to a target blockchain. For that, we specify the basic requirements of a cross-blockchain token transfer protocol, including the corresponding design of such a protocol. The protocol requires two main steps: burning tokens on the source blockchain and claiming tokens on the target blockchain.

Initially, a user issues a burn transaction, including the recipient, target blockchain, and token amount, to the source blockchain, which results in the user's tokens being burned. After that, a user must submit the corresponding claim transaction to the target blockchain, including the receiver and the burn transaction. During the execution of the claim transaction, the token contract on the target blockchain needs to verify if the burn transaction is included in the source blockchain and confirmed with overwhelming probability before recreating the tokens. The protocol relies on a cross-blockchain communication mechanism to retrieve the block with the respective burn transaction from the source blockchain. For that, the protocol relies on blockchain relays and cross-blockchain oracles.

Additionally, the protocol foresees an optional third step where users submit a confirm transaction on the source blockchain to indicate the successful recreation of the tokens on the target blockchain, which also requires verifying that the claim transaction is included in the source blockchain and confirmed with overwhelming probability. Further, the protocol applies an incentive mechanism to ensure users execute cross-blockchain transfers completely. Users finishing orphaned cross-blockchain token transfers receive a share of the transferred amount of tokens as a reward. While this solution already provides a lightweight solution to cross-blockchain token transfers, there is still further potential to reduce the cost overhead from executing the protocol on Layer 1.

Hence, we propose another cross-blockchain token transfer protocol that follows similar mechanisms but operates primarily on Layer 2 instead of

Layer 1 [SLS24]. The protocol applies the concept of zk rollups to move transaction execution and storage off-chain. Users send transactions to an operator that executes the transactions locally and only submits a validity proof, i.e., a zk-SNARK, the updated state root, and transaction data to the token contract on the source blockchain. After that, the source operator uses a cross-blockchain communication mechanism to transfer the state information of the source rollup to the target rollup. An operator of the target rollup executes the same bundle of transactions, creates a validity proof, and submits the zk-SNARK, updated state root, and transaction data to the token contract on the target blockchain.

Additionally, this protocol relies on a different incentive mechanism where each transaction requires a dynamic fee selected by the user to reward the operators. However, to ensure that operators execute cross-blockchain transfers completely, both operators can only receive the transaction fees on the target blockchain. Therefore, the source operator is incentivized only to include transactions with transaction fees that are high enough to cover the costs of cross-blockchain communication and the execution of the transactions on both blockchains. This approach enables dynamic transaction fees without requiring collaterals or minimal transfer values. Additionally, since the token contracts deployed on both blockchains only store a single Merkle root representing the state of the token contract, the protocol supports rollback mechanisms in the case of possible inconsistencies due to forks. Further, by conducting most of the computations on Layer 2, we achieve a cost reduction of 88% for burning and 97% for claiming tokens compared to our previous solution operating on Layer 1.

4.3 Cross-Blockchain Smart Contract Calls

Although not many works address cross-blockchain smart contract calls, some do exist. These works include, e.g., solutions based on trusted validators to enable synchronous cross-blockchain smart contract calls [NSSB21; RR21]. Another solution relies on synchronizing smart contracts across multiple blockchains using state proofs to create remote counterparts that allow instant read-only cross-blockchain smart contract calls [WK23]. Further, other approaches offer complete platforms for executing cross-blockchain smart contract calls, such as HyperService [LXS+19] or appXChain [MSJ+21]. Unlike these approaches, our solution avoids multiple trusted intermediaries or additional blockchains, providing necessary abstractions and enabling asynchronous cross-blockchain smart contract calls while maintaining a high degree of decentralization.

Therefore, in [SSS+23], we present a novel framework to enable cross-blockchain smart contract calls inspired by RPC, a well-known concept for communication between nodes in computer networks. The protocol allows

smart contracts to asynchronously call functions of other smart contracts across the boundaries of EVM-based blockchains and receive the execution results. Further, the framework allows for creating stub contracts that abstract cross-blockchain communication details. The framework relies on blockchain relays to communicate the request and reply to messages regarding the stub contracts. A client contract can seamlessly interact with a server contract on another blockchain while the transaction sender provides the necessary information to verify the exchanged messages.

Initially, a user issues a normal transaction to execute the client contract on the target blockchain. After that, the client contract calls the client stub to call a function of the server contract on the target blockchain. The client stub creates the request message and stores it on the source blockchain. Then, relayers transfer the request message by submitting the respective block headers to the relay contract on the target blockchain. Then, the transaction sender can trigger the execution of the cross-blockchain smart contract call on the target blockchain and the proof. The server stub on the target blockchain calls the relay contract to verify the request message, calls the server contract, and stores the response message with the result. Finally, the transaction sender calls the client stub on the source blockchain, providing the response message with the respective proof. The client stub verifies the response message using the relay contract and calls the callback function. Further, the protocol applies an incentive mechanism to ensure the complete execution of a smart contract call. Each user wanting to issue transactions resulting in cross-blockchain smart contract calls must place collateral. Other users can finish orphaned cross-blockchain smart contract calls and receive part of the collateral as a reward. Our results show that using blockchain relays for cross-blockchain smart contract calls introduces a considerable cost overhead. Nevertheless, applying blockchain relays leads to a high degree of decentralization.

5

Distributed Key Generation with Smart Contracts

DKG is at the core of threshold cryptosystems. Therefore, there is a continuous effort to advance DKG protocols. From the introduction of the first DKG protocol in 1991 by Pedersen [Ped91] until today, many different protocols have been proposed. These protocols improve on Pedersen's DKG protocol to ensure a uniform distribution of the shared secret [GJKR99; GJKR03], create DKG protocols working in an asynchronous network setting [KG09; DYX+22] and others. More recent works also explore the application of blockchain technology to enhance DKG protocols. However, the number of works exploring this option is still relatively low. Schindler et al. [SJSW19] propose a DKG protocol based on Ethereum with dynamic participation and cryptoeconomic-incentives. In [SRMH21], the authors extend and simplify the DKG protocol by Schindler et al. to make it more efficient but restrict the applicability to only specific use cases. Another work uses the Bitcoin blockchain with a pegged sidechain to share secrets [DLD+21]. The authors of [ZQHK22] propose a publicly verifiable DKG protocol using different cryptographic techniques. Unfortunately, these works are very costly or add more complexity. Therefore, exploring more cost-efficient solutions is necessary. In particular, the application of zk-SNARKs to move the heavy computations of DKG protocols off-chain, such as the dispute mechanism and derivation of the public key, can help to reduce the overhead of introducing smart contracts to DKG protocols.

In [SKS+23], we present a blockchain-based DKG protocol based on zk-SNARKs. The protocol uses smart contracts to execute the different steps of the DKG protocol, including *Share Distribution*, *Dispute*, and *Key Derivation*. Further, the blockchain acts as a broadcast channel, so the participants do

not need to establish point-to-point connections. Initially, each participant acts as a dealer in n parallel executions of Feldman's Verifiable Secret Sharing (VSS) protocol [Fel87] and broadcasts the necessary information through the blockchain. Afterward, the protocol enables each participant to issue a dispute against malicious dealers. For that, a participant challenges the dealer to prove the shared information is correct. The dealer has to submit a zk-SNARK proving that it behaved honestly. Finally, to derive the shared public key, a participant computes the public key off-chain, generates a zk-SNARK to prove the correct computation of the public key, and submits it to the smart contract. The smart contract only accepts and stores the public key if the proof is valid. Afterward, the participants can use the generated private key to collaboratively execute cryptographic operations, while smart contracts can use the public key for verification.

The ability of smart contracts to directly access the public key is beneficial for applications that require the verification of threshold signatures by a smart contract, e.g., the cross-blockchain oracle in [SSSS21]. Otherwise, the public key is not directly available to the smart contract, which requires an additional mechanism to submit the public key. Further, adding crypto-economic incentives for the members of the oracle committee can ensure the correct execution of the DKG protocol. Additionally, our results show a considerable cost reduction of up to 66% for disputes and 46% for submitting the master public key compared to the related work [SJSW19] due to moving heavy computations off-chain.

6

Conclusion

Interoperability is one of the major obstacles to the mass adoption of blockchain technology in various use cases. While multiple solutions exist for creating interoperable blockchains, many have considerable overhead compared to regular intra-blockchain communication. Hence, we present different lightweight blockchain interoperability solutions, including cross-blockchain communication to enable the exchange of arbitrary data on a low level and cross-blockchain token transfer and smart contract call protocols on a higher level to enable application-based interoperability.

For cross-blockchain communication, we present two different cross-blockchain oracle solutions that enable the transfer of arbitrary data between two distinct blockchains. Both solutions apply well-known cryptographic techniques such as threshold signatures and ZKPs to reduce the cost overhead of the aggregation mechanism executed by the committee of oracle nodes. Our results show that both solutions incur lower costs than naive oracle approaches, which execute most computations directly on the blockchain. Further, we show that our oracle solutions can compete with other blockchain interoperability solutions while allowing for more flexibility.

Additionally, we present two cross-blockchain token transfer protocols that allow users to transfer their tokens freely between blockchains. The first protocol lays the groundwork for conducting cross-blockchain token transfers, and the second protocol enables more efficient transfers and rollbacks by moving computations and storage off-chain. The evaluation of both protocols shows their practical applicability through considerable cost reductions.

We also present a framework for asynchronous cross-blockchain smart contract calls to allow different smart contracts to interact with each other on

different blockchains, enabling the ability to create cross-blockchain DApps. Our work analyzes the cost overhead of conducting cross-blockchain smart contract calls and the average execution time. Our results show that conducting cross-blockchain smart contract calls with blockchain relays requires a considerable overhead in costs and execution time. However, it preserves a high degree of decentralization and does not require an additional blockchain.

Finally, we present a blockchain-based DKG protocol with crypto-economic incentives to ensure accountability, transparency, and the correct execution of the protocol. Our work analyzes the additional overhead of introducing smart contracts to a DKG protocol, lowering the costs compared to similar protocols. Further, we analyze the required resources for generating the proofs, showing the feasibility of running the protocol with consumer-grade hardware.

Open Problems

Cross-Blockchain Communication. In the past years, many different solutions for cross-blockchain communication were presented, including blockchain relays, notary schemes, and blockchains of blockchains to establish the ability to transfer arbitrary data between different blockchains. These protocols establish a link between two or more blockchains, resulting in a network of blockchains that can interact with each other. However, creating a fully connected network of blockchains seems infeasible. Therefore, looking into routing algorithms for cross-blockchain messaging across a network of blockchains is necessary to establish efficient cross-blockchain communication over a whole network of blockchains.

Cross-Blockchain Token Transfers. Most works focus on establishing a minimal viable protocol for cross-blockchain token transfers. Our work also revolves around the basics of cross-blockchain token transfers and making them cost-efficient to ensure their applicability in real-world scenarios. However, little effort is spent on the privacy aspect of cross-blockchain token transfers [Sto21; SSS22]. Introducing privacy to cross-blockchain token transfer protocols would protect users' privacy, e.g., identity and transaction details, which is essential for users who attach great importance to confidentiality. Therefore, there is a strong need to explore privacy in cross-blockchain token transfers. Especially, the application of zk-SNARKs, which we mostly use due to their succinctness, promises to be interesting to introduce privacy.

Cross-Blockchain Smart Contract Calls. Current protocols for cross-blockchain smart contract calls focus mainly on simple smart contract calls between two blockchains. However, more complex interactions, including nested calls, recursion, and exception handling, are mostly unexplored.

Therefore, it is necessary to improve upon existing protocols to enable the same capabilities as regular intra-blockchain smart contract calls while ensuring atomicity.

Distributed Key Generation with Smart Contracts. Our solution for DKG with smart contracts is based upon Pedersen's DKG protocol [Ped91], which is widely recognized for enabling secure and decentralized key generation. However, there is great potential to introduce blockchain technology into novel DKG protocols, which, e.g., provide better security guarantees or improve computational costs or communication complexity. Therefore, exploring if these protocols enable seamless integration with smart contracts is an interesting direction for future work.

7

Bibliography

[AAA22] Sidrah Abdullah, Junaid Arshad, and Muhammad Alsadi: *ChainNet: An Internet-inspired Framework for Interoperable Blockchains*. In *Distributed Ledger Technologies: Research and Practice*, volume 1 (2), 2022, pages 1–20. DOI: 10.1145/3554761.

[ABV+18] John Adler, Ryan Berryhill, Andreas G. Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania: *Astraea: A Decentralized Blockchain Oracle*. 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)*. IEEE, 2018, pages 1145–1152. DOI: 10.1109/CYBERMATICS_2018.2018.00207.

[ASZ22] Fouzia E. Alzhrani, Kawther A. Saeedi, and Liping Zhao: *A Taxonomy for Characterizing Blockchain Systems*. In *IEEE Access*, volume 10, 2022, pages 110568–110589. DOI: 10.1109/ACCESS.2022.3214837.

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick: *Hyperledger fabric: a distributed operating system for permissioned blockchains*. In *Thirteenth EuroSys Conference, EuroSys 2018*. ACM, 2018, pages 30:1–30:15. DOI: 10.1145/3190508.3190538.

[BVGC21] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia: *A survey on blockchain interoperability: Past, present, and future trends*. In *ACM Computing Surveys*, volume 54 (8), 2021, pages 1–41. DOI: 10.1145/3471140.

[BCR+19] Jorge Bernal Bernabé, José Luis Cánovas, José Luis Hernández Ramos, Rafael Torres Moreno, and Antonio F. Skarmeta: *Privacy-Preserving Solutions for Blockchain: Review and Challenges*. In *IEEE Access*, volume 7, 2019, pages 164908–164940. DOI: 10.1109/ACCESS.2019.2950872.

[BKN+21] Muhammad Nasir Mumtaz Bhutta, Amir A. Khwaja, Adnan Nadeem, Hafiz Farooq Ahmad, Muhammad Khurram Khan, Moataz A. Hanif, Houbing Song, Majed Alshamari, and Yue Cao: *A Survey on Blockchain Technology: Evolution, Architecture and Security*. In *IEEE Access*, volume 9, 2021, pages 61048–61073. DOI: 10.1109/ACCESS.2021.3072849.

[BSF+19] Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte: *Dextt: Deterministic Cross-Blockchain Token Transfers*. In *IEEE Access*, volume 7, 2019, pages 111030–111042. DOI: 10.1109/ACCESS.2019.2934707.

[BCC+21] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al.: *Chainlink 2.0: Next steps in the evolution of decentralized oracle networks*. 2021. URL: https://research.chain.link/whitepaper-v2.pdf (visited on 05/23/2024).

[ARSS20] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic: *Trustworthy blockchain oracles: review, comparison, and open research challenges*. In *IEEE Access*, volume 8, 2020, pages 85675–85685. DOI: 10.1109/ACCESS.2020.2992698.

[CL02] Miguel Castro and Barbara Liskov: *Practical byzantine fault tolerance and proactive recovery*. In *ACM Transactions on Computer Systems*, volume 20 (4), 2002, pages 398–461. DOI: 10.1145/571637.571640.

[Cha21] ChainLink: *Introducing the Cross-Chain Interoperability Protocol (CCIP) for Decentralized Inter-Chain Messaging and Token Movements*. 2021. URL: https://blog.chain.link/introducing-the-cross-chain-interoperability-protocol-ccip/ (visited on 05/23/2024).

[CB20] Yan Chen and Cristiano Bellavitis: *Blockchain disruption and decentralized finance: The rise of decentralized business models*. In *Journal of Business Venturing Insights*, volume 13, 2020, pages e00151. DOI: https://doi.org/10.1016/j.jbvi.2019.e00151.

[Com17] European Commission: *The European Interoperability Framework in detail*. 2017. URL: https://ec.europa.eu/isa2/sites/default/files/eif_brochure_final.pdf (visited on 05/02/2024).

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren: *Practical Asynchronous Distributed Key Generation*. In *43rd IEEE Symposium on Security and Privacy, SP 2022*. IEEE, 2022, pages 2518–2534. DOI: 10.1109/SP46214.2022.9833584.

[De 20] Alex De Vries: *Bitcoin's energy consumption is underestimated: A market dynamics approach*. In *Energy Research & Social Science*, volume 70, 2020, pages 101721. DOI: https://doi.org/10.1016/j.erss.2020.101721.

[DLD+21] Marina Dehez-Clementi, Jérôme Lacan, Jean-Christophe Deneuville, Hassan Asghar, and Dali Kaafar: *A Blockchain-enabled Anonymous-yet-Traceable Distributed Key Generation*. In *2021 IEEE International Conference on Blockchain, Blockchain 2021*. IEEE, 2021, pages 257–265. DOI: 10.1109/BLOCKCHAIN53845.2021.00042.

[DH20] Apoorvaa Deshpande and Maurice Herlihy: *Privacy-preserving cross-chain atomic swaps*. In *24th International Conference on Financial Cryptography and Data Security, FC 2020*. Springer, 2020, pages 540–549. DOI: 10.1007/978-3-030-54455-3_38.

[Des94] Yvo G Desmedt: *Threshold cryptography*. In *European Transactions on Telecommunications*, volume 5 (4), 1994, pages 449–458. DOI: https://doi.org/10.1002/ett.4460050407.

[Fel87] Paul Feldman: *A practical scheme for non-interactive verifiable secret sharing*. In *28th Annual Symposium on Foundations of Computer Science, SFCS 1987*. IEEE, 1987, pages 427–438. DOI: 10.1109/SFCS.1987.4.

[FF18] Tiago M Fernández-Caramés and Paula Fraga-Lamas: *A Review on the Use of Blockchain for the Internet of Things*. In *IEEE Access*, volume 6, 2018, pages 32979–33001. DOI: 10.1109/ACCESS.2018.2842685.

[Fou22] Hyperledger Foundation: *Introducing Hyperledger Cacti, a Multi-Faceted Pluggable Interoperability Framework*. 2022. URL: https://www.hyperledger.org/blog/2022/11/07/introducing-hyperledger-cacti-a-multi-faceted-pluggable-interoperability-framework (visited on 03/07/2024).

[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*. IEEE, 2020, pages 204–213. DOI: 10.1109/BLOCKCHAIN50366.2020.00032.

[GKO20] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov: *Zendoo: a zk-SNARK Verifiable Cross-Chain Transfer Protocol Enabling Decoupled and Decentralized Sidechains*. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020*. IEEE, 2020, pages 1257–1262. DOI: 10.1109/ICDCS47774.2020.00161.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin: *Secure applications of Pedersen's distributed key generation protocol*. In *Cryptographers' Track at the RSA Conference 2003, CT-RSA 2003*. Springer, 2003, pages 373–390. DOI: 10.1007/3-540-36563-X_26.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin: *Secure distributed key generation for discrete-log based cryptosystems*. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pages 295–310. DOI: 10.1007/3-540-48910-X_21.

[GKW+16] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun: *On the Security and Performance of Proof of Work Blockchains*. In *2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016*. ACM, 2016, pages 3–16. DOI: 10.1145/2976749.2978341.

[Goe20] Christopher Goes: *The Interblockchain Communication Protocol: An Overview*. 2020. arXiv: 2006.15918 [cs.DC].

[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*, volume 38 (3), 1991, pages 690–728. DOI: 10.1145/116825.116852.

[GMR19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff: *The knowledge complexity of interactive proof-systems*. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. ACM, 2019, pages 203–225. DOI: 10.1145/3335741.3335750.

[GY22] Huaqun Guo and Xingjie Yu: *A survey on blockchain technology and its security*. In *Blockchain: Research and Applications*, volume 3 (2), 2022, pages 100067. DOI: 10.1016/j.bcra.2022.100067.

[HJM+23] Md Arif Hassan, Mohammad Behdad Jamshidi, Bui Duc Manh, Nam H. Chu, Chi-Hieu Nguyen, Nguyen Quang Hieu, Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Nguyen Van Huynh, Mohammad Abu Alsheikh, and Eryk Dutkiewicz: *Enabling Technologies for Web 3.0: A Comprehensive Survey*. 2023. arXiv: 2401.10901 [cs.CY].

[HKG+20] Anton Hasselgren, Katina Kralevska, Danilo Gligoroski, Sindre A Pedersen, and Arild Faxvaag: *Blockchain in healthcare and health sciences—A scoping review*. In *International Journal of Medical Informatics*, volume 134, 2020, pages 104040. DOI: 10.1016/j.ijmedinf.2019.104040.

[Her18] Maurice Herlihy: *Atomic Cross-Chain Swaps*. In *2018 ACM Symposium on Principles of Distributed Computing, PODC 2018*. ACM, 2018, pages 245–254. DOI: 10.1145/3212734.3212736.

[KFP19] Andreas Kamilaris, Agusti Fonts, and Francesc X. Prenafeta-Boldu: *The rise of blockchain technology in agriculture and food supply chains*. In *Trends in Food Science & Technology*, volume 91, 2019, pages 640–652. DOI: 10.1016/j.tifs.2019.07.034.

[KKZ20] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros: *Proof-of-burn*. In *24th International Conference on Financial Cryptography and Data Security, FC 2020*. Springer, 2020, pages 523–540. DOI: 10.1007/978-3-030-51280-4_28.

[KG09] Aniket Kate and Ian Goldberg: *Distributed key generation for the internet*. In *29th IEEE International Conference on Distributed Computing Systems, ICDCS 2009*. IEEE, 2009, pages 119–128. DOI: 10.1109/ICDCS.2009.21.

[KMS20] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman: *Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures*. In *2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020*. ACM, 2020, pages 1751–1767. DOI: 10.1145/3372297.3423364.

[KSF+21] Benjamin Körbel, Marten Sigwart, Philip Frauenthaler, Michael Sober, and Stefan Schulte: *Blockchain-based result verification for computation offloading*. In *19th International Conference on Service-Oriented Computing, ICSOC 2021*. Springer, 2021, pages 99–115. DOI: 10.1007/978-3-030-91431-8_7.

[KB20] Jae Kwon and Ethan Buchman: *Cosmos Whitepaper: A Network of Distributed Ledgers*. 2020. URL: https://wikibitimg.fx994.com/attach/2020/12/16623142020/WBE16623142020_55300.pdf (visited on 05/03/2024).

[LWM+21] Weiwei Liu, Huaming Wu, Tianhui Meng, Rui Wang, Yang Wang, and Chengzhong Xu: *AucSwap: A Vickrey auction modeled decentralized cross-blockchain asset transfer protocol*. In *Journal of Systems Architecture*, volume 117, 2021, pages 102102. DOI: 10.1016/J.SYSARC.2021.102102.

[LXS+19] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu: *HyperService: Interoperability and Programmability Across Heterogeneous Blockchains*. In *2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*. ACM, 2019, pages 549–566. DOI: 10.1145/3319535.3355503.

[MSJ+21] Mohammad Madine, Khaled Salah, Raja Jayaraman, Yousof Al-Hammadi, Junaid Arshad, and Ibrar Yaqoob: *appXchain: Application-Level Interoperability for Blockchain Networks*. In *IEEE Access*, volume 9, 2021, pages 87777–87791. DOI: 10.1109/ACCESS.2021.3089603.

[Nak08] Satoshi Nakamoto: *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: http://www.bitcoin.org/bitcoin.pdf (visited on 05/23/2024).

[NSSB21] Markus Nissl, Emanuel Sallinger, Stefan Schulte, and Michael Borkowski: *Towards Cross-Blockchain Smart Contracts*. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2021*. IEEE, 2021, pages 85–94. DOI: 10.1109/DAPPS52256.2021.00015.

[NKJ+23] Xiangyu Niu, Lanju Kong, Fuqi Jin, Xiaolin Song, Xinping Min, and Qingzhong Li: *NFT Cross-Chain Transfer Method Under the Notary Group Scheme*. In *26th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2023*. IEEE, 2023, pages 996–1001. DOI: 10.1109/CSCWD57460.2023.10152551.

[PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova: *Pinocchio: Nearly practical verifiable computation*. In *Communications of the ACM*, volume 59 (2), 2016, pages 103–112. DOI: 10.1145/2856449.

[Ped91] Torben Pryds Pedersen: *A threshold cryptosystem without a trusted party*. In *Workshop on the Theory and Application of Cryptographic Techniques, EUROCRYPT 1991*. Springer, 1991, pages 522–526. DOI: 10.1007/3-540-46416-6_47.

[PBHM20] Babu Pillai, Kamanashis Biswas, Zhé Hóu, and Vallipuram Muthukkumarasamy: *The Burn-to-Claim cross-blockchain asset transfer protocol*. In *25th International Conference on Engineering of Complex Computer Systems, ICECCS 2020*. IEEE, 2020, pages 119–124. DOI: 10.1109/ICECCS51672.2020.00021.

[RMC+18] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz: *On blockchain and its integration with IoT. Challenges and opportunities*. In *Future Generation Computer Systems*, volume 88, 2018, pages 173–190. DOI: 10.1016/J.FUTURE.2018.05.046.

[RR21] Peter Robinson and Raghavendra Ramesh: *General Purpose Atomic Crosschain Transactions*. In *3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS 2021*. IEEE, 2021, pages 61–68. DOI: 10.1109/BRAINS52497.2021.9569837.

[RS04] Phillip Rogaway and Thomas Shrimpton: *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance*. In *11th International Workshop on Fast Software Encryption, FSE 2004*. Springer, 2004, pages 371–388. DOI: 10.1007/978-3-540-25937-4_24.

[Sal21] Fahad Saleh: *Blockchain without Waste: Proof-of-Stake*. In *The Review of Financial Studies*, volume 34 (3), 2021, pages 1156–1190. DOI: 10.1093/rfs/hhaa075.

[SSS22] Aleixo Sanchez, Alistair Stewart, and Fatemeh Shirazi: *Bridging Sapling: Private Cross-Chain Transfers*. In *2022 IEEE Crosschain Workshop, ICBC-CROSS*. IEEE, 2022, pages 1–9. DOI: 10.1109/ICBC-CROSS54895.2022.9793325.

[SJSW19] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl: *EthDKG: Distributed Key Generation with Ethereum Smart Contracts*. Cryptology ePrint Archive, Paper 2019/985. 2019. URL: https://eprint.iacr.org/2019/985.

[SSFB19] Stefan Schulte, Marten Sigwart, Philipp Frauenthaler, and Michael Borkowski: *Towards blockchain interoperability*. In *Business Process Management: Blockchain and Central and Eastern Europe Forum, BPM 2019*. Springer, 2019, pages 3–10. DOI: 10.1007/978-3-030-30429-4_1.

[SFS+21] Marten Sigwart, Philipp Frauenthaler, Christof Spanring, Michael Sober, and Stefan Schulte: *Decentralized cross-blockchain asset transfers*. In *3rd International Conference on Blockchain Computing and Applications, BCCA 2021*. IEEE, 2021, pages 34–41. DOI: 10.1109/BCCA53669.2021.9657007.

[SKS+23] Michael Sober, Max Kobelt, Giulia Scaffino, Dominik Kaaser, and Stefan Schulte: *Distributed Key Generation with Smart Contracts using zk-SNARKs*. In *38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023*. ACM, 2023, pages 231–240. DOI: 10.1145/3555776.3577677.

[SLS24] Michael Sober, Markus Levonyak, and Stefan Schulte: *Efficient Cross-Blockchain Token Transfers with Rollback Support*. In *2024 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2024*. IEEE, 2024, pages 9–18. DOI: 10.1109/DAPPS61106.2024.00009.

[SSS24] Michael Sober, Giulia Scaffino, and Stefan Schulte: *Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs*. In *Distributed Ledger Technologies: Research and Practice*, volume 3 (4), 2024, pages 27:1–27:24. DOI: 10.1145/3678187.

[SSS+23] Michael Sober, Giulia Scaffino, Marten Sigwart, Philipp Frauenthaler, Markus Levonyak, and Stefan Schulte: *A Framework for Asynchronous Cross-Blockchain Smart Contract Calls*. In *5th IEEE International Conference on Blockchain Computing and Applications, BCCA 2023*. IEEE, 2023, pages 294–301. DOI: 10.1109/BCCA58897.2023.10338866.

[SSSS21] Michael Sober, Giulia Scaffino, Christof Spanring, and Stefan Schulte: *A Voting-Based Blockchain Interoperability Oracle*. In *2021 IEEE International Conference on Blockchain, Blockchain 2021*. IEEE, 2021, pages 160–169. DOI: 10.1109/BLOCKCHAIN53845.2021.00030.

[SSF+23] Michael Sober, Marten Sigwart, Philipp Frauenthaler, Christof Spanring, Max Kobelt, and Stefan Schulte: *Decentralized cross-blockchain asset transfers with transfer confirmation*. In *Cluster Computing*, volume 26 (4), 2023, pages 2129–2146. DOI: 10.1007/S10586-022-03737-6.

[SRMH21] Oliver Stengele, Markus Raiber, Jörn Müller-Quade, and Hannes Hartenstein: *ETHTID: Deployable threshold information disclosure on Ethereum*. In *3rd International Conference on Blockchain Computing and Applications, BCCA 2021*. IEEE, 2021, pages 127–134. DOI: 10.1109/BCCA53669.2021.9657019.

[Sto21] Drew Stone: *Trustless, privacy-preserving blockchain bridges*. 2021. arXiv: 2102.04660 [cs.CR].

[SHM+22] Farhana Akter Sunny, Petr Hajek, Michal Munk, Mohammad Zoynul Abedin, Md Shahriare Satu, Md Iftekharul Alam Efat, and Md Jahidul Islam: *A Systematic Review of Blockchain Applications*. In *IEEE Access*, volume 10, 2022, pages 59155–59177. DOI: 10.1109/ACCESS.2022.3179690.

[Sza97] Nick Szabo: *Formalizing and Securing Relationships on Public Networks*. In *First Monday*, volume 2 (9), 1997. DOI: 10.5210/FM.V2I9.548.

[VKD23] Lokendra Vishwakarma, Amritesh Kumar, and Debasis Das: *CrossLedger: A Pioneer Cross-chain Asset Transfer Protocol*. In *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023*. IEEE, 2023, pages 568–578. DOI: 10.1109/CCGRID57682.2023.00059.

[WWC23] Gang Wang, Qin Wang, and Shiping Chen: *Exploring Blockchains Interoperability: A Systematic Survey*. In *ACM Computing Surveys*, volume 55 (13s), 2023, pages 290:1–290:38. DOI: 10.1145/3582882.

[WOY+19] Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang: *Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends*. In *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, volume 49 (11), 2019, pages 2266–2277. DOI: 10.1109/TSMC.2019.2895123.

[WHH+19] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim: *A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks*. In *IEEE Access*, volume 7, 2019, pages 22328–22370. DOI: 10.1109/ACCESS.2019.2896108.

[WD22] Martin Westerkamp and Maximilian Diez: *Verilay: A Verifiable Proof of Stake Chain Relay*. In *2022 IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2022*. IEEE, 2022, pages 1–9. DOI: 10.1109/ICBC54727.2022.9805554.

[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&P Workshops 2020*. IEEE, 2020, pages 378–386. DOI: 10.1109/EUROSPW51379.2020.00058.

[WK23] Martin Westerkamp and Axel Küpper: *Instant Function Calls Using Synchronized Cross-Blockchain Smart Contracts*. In *IEEE Transactions on Network and Service Management*, volume 20 (3), 2023, pages 2136–2150. DOI: 10.1109/TNSM.2023.3236437.

[Wik24] Bitcoin Wiki: *Script*. 2024. URL: https://en.bitcoin.it/wiki/Script (visited on 06/18/2024).

[Woo14] Gavin Wood: *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. 2014. URL: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 06/13/2024).

[Woo16] Gavin Wood: *Polkadot: Vision for a heterogeneous multi-chain framework*. 2016. URL: https://www.win.tue.nl/~mholende/seminar/references/ethereum_polkadot.pdf (visited on 01/11/2024).

[XZC+22] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song: *zkBridge: Trustless Cross-chain Bridges Made Practical*. In *2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*. ACM, 2022, pages 3003–3017. DOI: 10.1145/3548606.3560652.

[ZAZ+21] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt: *SoK: Communication Across Distributed Ledgers*. In *25th International Conference on Financial Cryptography and Data Security, FC 2021*. Springer, 2021, pages 3–36. DOI: 10.1007/978-3-662-64331-0_1.

[ZHL+19] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt: *XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets*. In *2019 IEEE Symposium on Security and Privacy, SP 2019*. IEEE, 2019, pages 193–210. DOI: 10.1109/SP.2019.00085.

[ZQHK22] Liang Zhang, Feiyang Qiu, Feng Hao, and Haibin Kan: *1-Round Distributed Key Generation With Efficient Reconstruction Using Decentralized CP-ABE*. In *IEEE Transactions on Information Forensics and Security*, volume 17, 2022, pages 894–907. DOI: 10.1109/TIFS.2022.3152356.

Part III

Original Research Papers: Blockchain Interoperability

Michael Sober, Giulia Scaffino, Christof Spanring, and Stefan Schulte: *A Voting-Based Blockchain Interoperability Oracle*. In *2021 IEEE International Conference on Blockchain, Blockchain 2021*. 2021.

Michael Sober, Max Kobelt, Giulia Scaffino, Dominik Kaaser, and Stefan Schulte: *Distributed Key Generation with Smart Contracts using zk-SNARKs*. In *38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023*. 2023.

Michael Sober, Giulia Scaffino, Marten Sigwart, Philipp Frauenthaler, Markus Levonyak, and Stefan Schulte: *A Framework for Asynchronous Cross-Blockchain Smart Contract Calls*. In *5th IEEE International Conference on Blockchain Computing and Applications, BCCA 2023*. 2023.

Michael Sober, Marten Sigwart, Philipp Frauenthaler, Christof Spanring, Max Kobelt, and Stefan Schulte: *Decentralized cross-blockchain asset transfers with transfer confirmation*. In *Cluster Computing*, volume 26 (4), 2023.

Michael Sober, Markus Levonyak, and Stefan Schulte: *Efficient Cross-Blockchain Token Transfers with Rollback Support*. In *2024 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2024*. 2024.

Michael Sober, Giulia Scaffino, and Stefan Schulte: *Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs*. In *Distributed Ledger Technologies: Research and Practice*, volume 3 (4), 2024.

8

A Voting-Based Blockchain Interoperability Oracle

DOI: 10.1109/BLOCKCHAIN53845.2021.00030

Keywords: *blockchain interoperability, blockchain oracles, threshold signature, smart contracts*

Abstract. Today's blockchain landscape is severely fragmented as more and more heterogeneous blockchain platforms have been developed in recent years. These blockchain platforms are not able to interact with each other or with the outside world since only little emphasis is placed on the interoperability between them. Already proposed solutions for blockchain interoperability such as naive relay or oracle solutions are usually not broadly applicable since they are either too expensive to operate or very resource-intensive.

For that reason, we propose a blockchain interoperability oracle that follows a voting-based approach based on threshold signatures. The oracle nodes generate a distributed private key to execute an off-chain aggregation mechanism to collectively respond to requests. Compared to state-of-the-art relay schemes, our approach does not incur any ongoing costs and since the on-chain component only needs to verify a single signature, we can achieve remarkable cost savings compared to conventional oracle solutions.

A Voting-Based Blockchain Interoperability Oracle

Michael Sober*†, Giulia Scaffino*‡, Christof Spanring§, Stefan Schulte*†
*Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
†Institute of Data Engineering, TU Hamburg, Hamburg, Germany
{michael.sober, stefan.schulte}@tuhh.de
‡Institute of Logic and Computation, TU Wien, Vienna, Austria
giulia.scaffino@tuwien.ac.at
§Pantos GmbH, Vienna, Austria
christof.spanring@bitpanda.com

***Abstract*—Today's blockchain landscape is severely fragmented as more and more heterogeneous blockchain platforms have been developed in recent years. These blockchain platforms are not able to interact with each other or with the outside world since only little emphasis is placed on the interoperability between them. Already proposed solutions for blockchain interoperability such as naive relay or oracle solutions are usually not broadly applicable since they are either too expensive to operate or very resource-intensive.**

For that reason, we propose a blockchain interoperability oracle that follows a voting-based approach based on threshold signatures. The oracle nodes generate a distributed private key to execute an off-chain aggregation mechanism to collectively respond to requests. Compared to state-of-the-art relay schemes, our approach does not incur any ongoing costs and since the on-chain component only needs to verify a single signature, we can achieve remarkable cost savings compared to conventional oracle solutions.

***Index Terms*—blockchain interoperability, blockchain oracles, threshold signature, smart contracts**

I. Introduction

In recent years, blockchain technology has become increasingly important not only as the underlying technology for cryptocurrencies [1] but also in many other application areas including supply chain management [2], healthcare [3], and others. This has led to the development of more and more heterogeneous blockchain platforms [4], [5]. These are often tailored to specific requirements, as there is not one blockchain that is capable of fulfilling the (often) disjunctive needs of different application areas.

Research and industry tend to not consider the interoperability between different blockchain platforms, turning these platforms into self-contained systems [4]. As a result, these platforms are not able to collaborate to benefit from novel features and properties of other or newly developed blockchain platforms. The problem of weak interoperability does not only exist within the world of blockchains but generally with systems that are outside the blockchain platform's boundaries. As an example, we can use the smart contract and decentralized application platform Ethereum [6]. Smart contracts running in the Ethereum Virtual Machine (EVM) cannot interact with the outside world, which means that a smart contract is only able to read and modify the state of the hosting blockchain. For many applications, however, it is important to be able to obtain data from the outside world. One can imagine, for example, a decentralized application that needs flight data from an airline to determine if a user has the right to compensation in case a flight is delayed or a cross-chain token transfer that requires the knowledge of whether the tokens were burned on the original blockchain. This problem is also known as the blockchain oracle problem [7].

Many attempts have been made to achieve interoperability between different blockchain platforms [5] and to solve the oracle problem [7], [8]. One possible approach to achieve interoperability between two blockchain platforms is the use of blockchain relays [9]. Such relays are usually provided as smart contracts running on a "target blockchain", i.e., the blockchain which needs data from a "source blockchain". With blockchain relays, the state of one blockchain is replicated within another blockchain, which enables callers of the relay contract to verify the existence of a transaction on the source blockchain.

Bitcoin and Ethereum use Merkle trees to store transactions in a block, whereby the root of the Merkle tree is stored in the block header. Therefore, a relay contract can use Simplified Payment Verification (SPV) which utilizes Merkle proofs to check whether a transaction is included in a block of a source blockchain [9]. Since block header validation is done by a smart contract on-chain, no trusted intermediary is needed to relay the block headers. However, this has the disadvantage that such relays also have considerable costs, since on-chain verification is very expensive and blocks have to be relayed continuously, even if they are perhaps not needed at all.

Another approach is to make use of blockchain oracles to get information from the outside world. Blockchain oracles are bridges between blockchain platforms and external data sources. The task of a blockchain oracle is to query data from external data sources and then to pass the data items to a smart contract. The problem here is to ensure the authenticity and integrity of the data because we have to trust the oracle that it behaves honestly [8].

One can differentiate between oracles that are based on a centralized or a decentralized approach [7]. Centralized oracles represent a single point of failure since trust in a single oracle node is required. Following the decentralized model, there is no single point of failure and the trust assumption moves from

one oracle node to multiple oracle nodes. Many decentralized oracles follow a voting-based approach to provide data, i.e., the votes are aggregated to determine the overall result. Unfortunately, the aggregation mechanism can become very expensive, if carried out on-chain.

To make a step towards blockchain interoperability and overcome the issues of current relay schemes and oracle solutions, we investigate the application of Boneh-Lynn-Shacham (BLS) threshold signatures to create an off-chain aggregation mechanism to reduce the operating costs of the oracle. Further, we examine the use of Distributed Key Generation (DKG) protocols to generate distributed private keys which are necessary for the creation of the threshold signatures while also preserving the decentralized nature of the blockchain. We provide the system design of a voting-based blockchain interoperability oracle that makes use of the aforementioned concepts and enables clients to verify that a transaction is included in another blockchain. Further, we deliver a prototypical implementation and show the applicability of our proposed solution by conducting a security and cost analysis.

The remainder of this paper is organized as follows: In Section II, we introduce some basic concepts needed in our approach. Subsequently, we present the design of the oracle in Section III. We follow up with implementation details of the prototype in Section IV and evaluate our solution in Section V. Afterward, we discuss related work in Section VI. Finally, Section VII concludes the paper.

II. Background

In the course of this section, we explain the basics of BLS signatures, followed by a short description of Verifiable Secret Sharing (VSS) which leads us to a discussion of DKG protocols.

A. BLS Signatures

The BLS signature scheme [10] is based on elliptic curve pairings ($e : G_1 \times G_2 \rightarrow G_T$) respectively bilinear maps. Using BLS, a public key PK is generated by multiplying the selected private key SK with the generator G of a cyclic group. To create a signature σ one has to hash the message m on the curve $H(m)$. This can be accomplished by using a hashing algorithm like the Secure Hash Algorithm (SHA)-256, whereby the resulting hash is used as the x-coordinate of a point on the curve.

If it is not possible to find such a point using this x-coordinate, it can simply be incremented until a valid point is found. This is an essential difference from other signature schemes in which the hash can be used directly. After that, the signature can be calculated by multiplying the point with the private key. To verify the signature, it comes down to checking the bilinear pairings as can be seen in Eq. 1.

$$e(\sigma, G) = e(H(m), PK) \tag{1}$$

This signature scheme has the advantage that it allows to generate particularly short signatures. Another very important aspect especially for this work is that it is possible to create threshold signatures [11] using a secret sharing scheme (see Section II-B), whereby multiple participants share a distributed private key and t out of n signature shares are required to create a valid signature.

Also, it enables the aggregation of multiple signatures [12], whereby we only need to verify two elliptic curve pairings to verify the aggregate signature. This is particularly interesting in the area of blockchain technology. For example, one could instead of verifying every single signature of the transactions in a block, only verify a single aggregate signature.

B. Verifiable Secret Sharing

With the help of a secret sharing scheme, it is possible to divide a secret S into n shares whereby each party gets a different share of S, but at least t shares need to be known to recover S. Otherwise, it is not possible to get any knowledge about S. These schemes are also known as (t, n) threshold schemes.

One of the first secret sharing schemes has been proposed by Shamir [13] in 1979. Using Shamir's scheme, a dealer picks a polynomial $f(x)$ (see Eq. 2) of degree $t-1$ whereby the constant term a_0 of the polynomial is equal to S and the coefficients are chosen randomly.

$$f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \tag{2}$$

The secret can be shared with n different parties by evaluating the polynomial at the positions $x_n = 1..n$ and distributing the values for $f(x_n)$ along with x_n to the respective parties. Since every polynomial of degree $t-1$ is defined by exactly t points, we can make use of polynomial interpolation (e.g., using the Lagrange interpolation formula) to recover the original polynomial and to evaluate it at position 0 to get the secret.

One problem with this, however, is that Shamir's scheme does not take into account that a dealer could distribute incorrect or inconsistent shares. Furthermore, the shareholders could also return incorrect shares. We cannot assume that the dealer and the shareholders are trustworthy, which is why we need secret sharing schemes that also take this kind of behavior into consideration, as is the case with VSS schemes. By using such schemes, each party can verify that it has received the correct information from the dealer and that shareholders submitted the correct shares. Hereby, we limit ourselves mainly to non-interactive VSS schemes, in which only the dealer sends messages and no communication between the other participants is necessary. This reduces the communication overhead.

One commonly used example of such a scheme is Feldman's VSS [14]. Feldman's VSS is based on Shamir's secret sharing scheme but additionally makes use of homomorphic encryption to commit to the secret and coefficients of the random polynomial. The commitments are broadcast while the shares are distributed through private channels. Each party can use the commitments to verify the validity of its share, due to the homomorphic property of the used encryption scheme.

A problem with this approach is that the commitment to the secret also leaks information about the secret. This is where Pedersen's VSS [15], another very well-known scheme, constitutes a more secure solution. Like Feldman's VSS, it is also based on Shamir's secret sharing scheme, but it makes use of a different commitment scheme that ensures that no information about the secret is leaked unless one can find a solution to the discrete logarithm problem.

C. Distributed Key Generation

In the previous section, we discussed VSS, whereby such schemes ensure that a dealer distributes the secret correctly and the shareholders cannot provide incorrect shares of the secret. However, the dealer yet has to be trusted as it still knows the secret. This is, among other things, a big concern in the area of threshold cryptography, where we want that only certain subsets of $t \leq n$ participants can jointly encrypt data or create a signature. Since the dealer knows about the distributed private key, it has the opportunity to encrypt data or to create signatures without the consent of at least t out of n participants.

To avoid this, a DKG protocol that allows the generation of distributed private keys without the dependency on any trusted third party can be used. In such protocols, no party owns the private key and the private key is never reconstructed. Many such protocols have been proposed over time [16]–[19], whereby we limit ourselves to the first proposed DKG protocol by Pedersen [16] to describe the basic concept.

In Pedersen's DKG protocol, every participant acts as a dealer during one of the n parallel executions of Feldman's VSS scheme to share a randomly picked secret. Each participant publishes its commitments using a broadcast channel, such as a blockchain. After that, every participant sends the signed private shares to the other participants through private channels. When receiving a share, each participant verifies the share by using the previously published commitments. If it is an invalid share, a complaint including the share and the signature is broadcast. Each participant computes its private share by summing up all shares received from the other parties which shared their secret correctly and computes the public key through the published commitments.

III. System Design

In this section, we propose the design of a voting-based blockchain interoperability oracle. Initially, we give a brief overview of the basic concept. We follow up with a description of the architecture and finally provide a comprehensive definition of the functionality.

A. Overview

The proposed design for a voting-based blockchain interoperability oracle (see Figure 1) allows clients to verify that a transaction is included in another blockchain. It uses an off-chain aggregation mechanism (see Section III-E) based on BLS threshold signatures. The oracle nodes are divided into one aggregator and multiple validators which can collectively

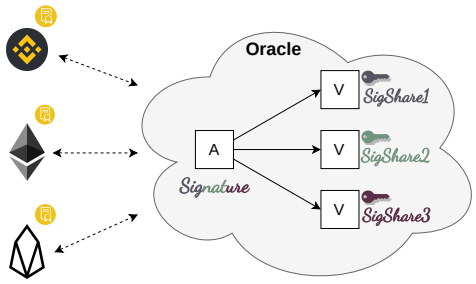

Fig. 1. Overview of the System

respond to requests issued by clients (i.e., parties which are interested in the verification of a transaction from another blockchain). For this, the oracle nodes use a DKG protocol to generate a distributed private key (see Section III-D), whereby each node only knows its private key share and the shared public key. Validators obtain the data from the other blockchain and sign it with their private key share, while the aggregator collects the data and the signature shares from the validators to recover the final signature to submit it along with the data to the smart contract. If the aggregator is not able to collect at least the threshold of signature shares with the same result, it cannot generate a valid signature. To ensure reliability, the aggregator changes over time so that each oracle node takes over the task of the aggregator at some point.

As an integral part of the system, we also apply an incentive mechanism (see Section III-F) to encourage oracle nodes to engage as part of the decentralized oracle and to behave honestly. For this, the client must provide compensation for the transaction fees that arise from submitting the result using the oracle contract and also offer two additional rewards. These additional rewards consist of the aggregation reward and the validation reward. Without these rewards, the aggregator, as well as the validators, would have no interest in participating as they would only be providing their resources without getting anything back.

B. Architecture

In our architecture, we can differentiate between the components that are on the blockchain, i.e., the smart contracts, and the components that are off-chain, i.e., the oracle nodes.

For the on-chain components, we make use of three different smart contracts. The first of these is a registry contract, which is responsible for managing all oracle nodes (see Section III-C) and selecting the current aggregator (see Section III-E). Via this smart contract, an oracle node discovers the other oracle nodes and checks if they got selected as the current aggregator.

The next component is the oracle contract, which receives requests from clients and sends them to all oracle nodes. Furthermore, the oracle contract is also responsible for receiving and verifying the responses from the aggregator (see Section III-E) and transferring the rewards. Further, it also stores all responses and makes them available to the clients.

The last smart contract is the key contract, which is responsible for managing the public key and initiating the generation

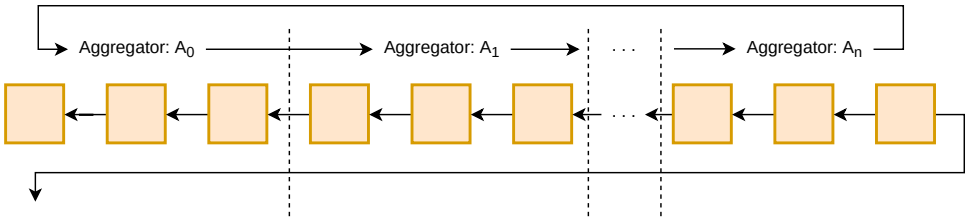

Fig. 2. Aggregator selection mechanism

of new keys (see Section III-D). The off-chain component consists of oracle nodes that can take on the tasks of the aggregator as well as the validator.

C. Oracle Registration

The first step of an oracle node to join the decentralized oracle is to register using the registry contract. During registration, the oracle nodes must provide their host address and BLS public key. Furthermore, a stake must be deposited which is used as a measure against Sybil attacks (see Section V-A3). After an oracle node has registered, it is eligible to take part in the run of the DKG protocol.

Another important point is that oracle nodes must also be able to deregister. Furthermore, oracle nodes can also be kicked out through a majority vote if they behave incorrectly and lose their stake.

Particular care is taken to ensure that the system remains fully operational. A distinction is made between the signature threshold and the threshold of qualified validators. The validator threshold is required to ensure that the system continues to work even if several validators fail or are misbehaving. Should the number of validators fall below the validator threshold, a new key generation process is triggered.

D. Distributed Key Generation

Each time a new oracle node registers, the registry contract checks whether a new run of the DKG protocol should be initiated. For this, we set a certain number, which indicates how many new registrations are necessary. Should the number be reached, the registry contract calls the key contract to trigger the start of the DKG protocol. The key contract then broadcasts a generation event containing the threshold which should be used.

On receiving the generation event, oracle nodes need to wait a certain amount of time (e.g., measured in blocks) to ensure that every oracle node had a chance to receive the event since it can take some time for a block to get propagated through the network. Then, the oracle nodes execute the DKG protocol with all currently registered oracle nodes. In our approach, we make use of Pedersen's DKG protocol (see Section II-C). However, other DKG protocols could also be used, as long as the run of the protocol is made transparent through the usage of blockchain technology.

The entire run of the DKG protocol takes place off-chain, which means that we need to get the generated public key as well as the number of qualified validators into the blockchain, i.e., we already have the oracle problem in our proposed solution. One possibility of getting the public key into the blockchain is to use an on-chain aggregation mechanism. However, this approach has the disadvantage that it gets more expensive the higher the number of oracle nodes becomes. These costs could be neglected if the frequency of generating new keys is very low. The same applies to the possibility of using a dispute mechanism, which is used by our prototype whereby only one oracle node submits the shared public key and the other oracle nodes can dispute the key. Finally, one could also assume that several other oracles already exist that can be used for this task. These could be oracles that may be pursuing a completely different approach, or it is already another instance of the proposed oracle solution. With the latter, however, we have a bootstrapping problem where the first instance cannot call an existing oracle. Therefore, one would have to use one of the first two approaches or a centralized solution and change it later on.

E. Off-chain Aggregation

The task of aggregating the results and the signature shares is taken over by an aggregator, which is one of the oracle nodes. The aggregator changes in a cycle of n blocks based on a round-robin mechanism (see Figure 2). This has several advantages over an approach in which anyone can aggregate and submit the voting results: It prevents multiple simultaneous submissions of which only the first one would be successful, while all the other oracle nodes which also try to submit an aggregation result have to pay the costs for the failed transaction without getting compensated. This would lead to the problem that oracle nodes are not incentivized to submit a result for which it is not certain whether the submission would be successful or not. Since this is relatively difficult to determine, nodes could become reluctant to act as an aggregator.

Another benefit is that this approach also consumes less bandwidth since not all oracle nodes try to aggregate a result, which reduces the number of exchanged messages. The fact that the aggregator changes over time, still ensures that if an aggregator should fail, the next aggregator will take its place and normal operation can be continued.

The aggregation mechanism (see Figure 3) starts when a client sends a request to the oracle contract. After that, all oracle nodes are notified about the request (Step 1). Subsequently, the aggregator begins to collect results from

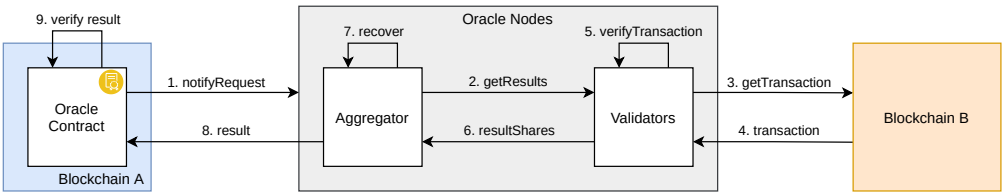

Fig. 3. Aggregation mechanism

the validators (Step 2), whereby the request only needs to contain the request number so that the validators know which request should be fulfilled. On receiving a request from the aggregator, a validator retrieves the transaction from the target blockchain (Step 3 and 4), verifies the transaction (Step 5), and returns the response consisting of the response to the request and the signature share (Step 6).

The aggregator collects results until it has at least threshold t identical results with valid signature shares. If the aggregator does not receive at least t results with valid signature shares, it will not be able to recover the signature. This can happen, as there can be temporary inconsistencies in blockchain systems, validators fail or behave incorrectly.

Regarding the first problem, however, one can assume that consistency will be achieved at some point (depending on the source blockchain) and that the threshold of the validators will agree. This is one of the advantages that is given by the fact that this oracle is limited to providing data from other blockchains. In this case, the aggregator can try again after a while, or if the aggregator changes in the meantime, the new aggregator will take over the request and repeat the voting.

If the aggregator has received enough shares, it recovers the full BLS signature using Lagrange interpolation (Step 7). After recovering the full BLS signature, the aggregator can submit the result to the oracle contract by providing the result and the signature (Step 8). The oracle contract then hashes the result to a point on the curve and checks the elliptic curve pairing (Step 9). If the elliptic curve pairing is successful, the smart contract calculates the reward (see Section III-F) and transfers it to the aggregator. Finally, the oracle contract notifies the client that the result is available. Here, however, one could also pursue a different approach in which the client also defines a callback function which is to be called when the result gets submitted. In this case, other problems would have to be considered, such as the costs caused by the callback function. Furthermore, the result is then not easy to obtain for other clients.

F. Incentive Mechanism

Since the aggregator only submits a response including the data and the threshold signature, the oracle contract does not have any information about the validators who were contacted, which makes it difficult to reward validators directly. We cannot rely on the aggregator to provide us with this information either, as it cannot be verified whether it is correct, as the entire DKG protocol is carried out off-chain. Furthermore, it is not particularly practical to reward each validator individually, as this would result in very high fees for the client, not only because the client has to pay each validator individually but also because each transfer causes additional costs during the execution of the smart contract.

Therefore, we propose to only reward the aggregator for submitting the result. The aggregator gets compensated for the transaction costs, receives an aggregation reward, and additionally has the chance to win the validation reward with a certain probability. The probability is scaled super-linearly based on the deposited stake to encourage the creation of only one identity [20]. If the aggregator is not lucky and does not win the validation reward, the reward is maintained by the oracle contract until an aggregator gets lucky enough to win the accumulated validation rewards.

It is particularly important to ensure that the validators are not able to predict whether the aggregator will win the validation reward. Otherwise, validators would benefit from not providing the aggregator with an answer to be able to increase their own chances of winning when they become aggregators themselves (see Section III-E). Therefore, we leverage the unpredictable randomness provided by the signature which is recovered by the aggregator to decide whether the aggregator receives the validation reward. In the case of the aggregator, it does not matter whether it knows if it will receive the validation reward before submitting, since the aggregator is still encouraged to submit the result, as to at least receive the aggregation reward. As for the validators, they are still incentivized to answer the current aggregator with the assumption that it will not receive the reward and thus increase their winnable reward when it is their turn to be an aggregator.

Since a round-robin mechanism is used for aggregator selection, care should be taken that the chance of winning the validation reward is not too high. Validators who were recently selected as the aggregator have a reason to assume that another aggregator will win the validation reward during the time they will have to wait to be selected again. They may be tempted to stop validating until they believe otherwise. Therefore, the chance should be small enough such that the validators can still assume that no one will win the reward in the current aggregation round.

IV. Implementation

After defining the system design of our solution, we created a prototypical implementation. Our prototype enables Ethereum-based blockchains to exchange data with each other. However, the solution can be implemented to work with other

blockchains. For this, the target blockchain needs to have smart contract capabilities and enable elliptic curve pairing checks. In this section, we discuss the used technologies as well as the implementation of the smart contracts and the oracle node which is available as open-source software on GitHub[1].

A. Smart Contracts

For the implementation of the prototype, we decided in favor of Ethereum because it is one of the most popular second-generation blockchains and accordingly a wide range of tools is available. Another important factor was that Ethereum already provides a precompiled contract that enables elliptic curve pairing checks on the alt_bn128 curve.

The registry contract stores all oracle nodes in an iterable mapping and provides clients with the necessary functions to retrieve them. For the aggregator selection mechanism, we defined a threshold of six blocks after which the aggregator will be switched. Further, after every third registration, the registry contract calls the key contract to trigger a new execution of the DKG protocol.

On receiving the call from the registry contract, the key contract calculates the threshold based on the number of currently registered oracle nodes and emits the key generation event. For the threshold, we defined that the majority of all registered nodes is necessary to produce a valid signature. For submitting the key, we provide a function in the prototype via which the public key can be set by one of the oracle nodes. The public key can be disputed if the majority of oracle nodes agree.

When implementing the oracle contract, we especially paid attention to implementing the contract in a gas-efficient way. Since storage operations are among the most expensive, we have tried to keep the number of these low. If a client sends a request to the oracle contract, the request is only emitted as an event and not saved in storage. This is possible because only oracle nodes need to be able to read the requests. However, this approach involves more work for the oracle nodes since they have to filter the blockchain for past events in case they missed some of them.

As has already been mentioned before, we use a precompiled contract that allows pairing checks for the alt_bn128 curve to verify the BLS signatures submitted by the aggregator. These precompiled contracts are already existing contracts that run outside the EVM and perform more complex tasks. One of the advantages of these contracts is that they are usually cheaper. Further, we make use of the try and increment approach (see Section II) to hash the response on the curve.

B. Oracle Node

The oracle node is implemented in the Go programming language. For the implementation, we used the advanced crypto library Kyber[2]. We adapted the library to work with the alt_bn128 curve used by Ethereum. The library provides the necessary packages for threshold BLS signatures and the implementation of a DKG protocol which is based on the protocol proposed by Pedersen (see Section II-C). For the DKG protocol, we needed a broadcast channel, for which we opted for the IOTA tangle [21], as it enables fee-less and publicly verifiable data exchange. Other broadcast channels (e.g., Ethereum) can also be used, but they may incur additional costs. The oracle nodes use a Go client to create transactions and send them to an IOTA node. These are zero-value transactions that are only used to exchange messages which are necessary to execute the DKG protocol.

Furthermore, the oracle nodes must be able to communicate directly with one another. The aggregator must be able to collect the results from the validators and all oracle nodes need a private channel to each other to distribute the private shares during the execution of the DKG protocol. Therefore, we make use of gRPC Remote Procedure Calls (gRPC) to connect the oracle nodes.

To interact with the smart contracts and retrieve data from an Ethereum blockchain, the oracle nodes need access to an Ethereum node whereby we make use of the Go Ethereum client to connect to the Remote Procedure Call (RPC) server. Further, we use a Go binding generator to create the counterparts of the smart contracts in Go to produce as little boilerplate code as possible.

V. Evaluation

In this section, we analyze the security and the costs of the proposed solution. This provides insight into if the solution is applicable, what problems can arise and what needs to be considered.

A. Security Analysis

To analyze the security of the oracle, we look at various attack scenarios and the consequences that can result from them. In particular, we are looking at lazy voting, free loading, Sybil attacks, and the key submission.

We can categorize the oracle nodes based on the Byzantine-Altruistic-Rational (BAR) model proposed in [22], which has already found application in other works for security analysis in the area of blockchain technology, e.g., [23]: *Rational* oracle nodes deviate from the protocol as long as they will be able to increase their benefits by doing so. *Byzantine* oracle nodes, however, can unexpectedly deviate from the protocol for unknown reasons, whereby it does not matter if it is intentional or unintentional misbehavior and whether they gain a benefit from it. *Altruistic* oracle nodes will always adhere to the protocol no matter if the rational choice would provide them with additional benefits.

The proposed oracle solution applies an incentive mechanism (see Section III-F) to encourage all oracle nodes to follow the protocol. However, it should be mentioned that even if all incentives are aligned properly, rational oracle nodes may deviate from the protocol. The reason for this is that also actors outside the oracle have to be taken into account. Hence, it may seem rational for oracle nodes to follow the protocol solely based on the applied incentive mechanism, but external

[1] https://github.com/pantos-io/ioporacle
[2] https://github.com/dedis/kyber

factors can influence their decision by providing better benefits for arbitrary reasons.

1) Lazy Voting: In the lazy voting problem, rational oracle nodes do not deliver the correct result but always provide the same response to maximize their benefits. If e.g., it is the case that a certain result occurs particularly often, it can be more beneficial to always return the same result directly instead of executing the request. This could lead to an incorrect result being successfully submitted. In our proposed solution, this would mean that a lazy validator would always respond to the request "Is transaction tx included in blockchain B and confirmed by at least n blocks?" with $true$. Since it is more important for many use cases that a certain transaction is included and confirmed, it can be assumed that requests will be answered more frequently with $true$ than with $false$. However, this problem can be circumvented by modifying the request. We expand the usual request and ask "In which block on blockchain B is transaction tx included and is it confirmed by at least n blocks?" instead. This question forces lazy validators to read from blockchain B since otherwise, they cannot know in which block the transaction is contained. The lazy voting problem is also discussed in related work (see Section VI-C) about decentralized oracles.

2) Free Loading: For the creation of a valid signature, the aggregator only needs to collect as many valid signature shares as the threshold t specifies. This means that in the best case only t validators have to execute the request. However, all other validators who have not contributed to the creation of the signature also have the chance to win the validation reward, when selected as an aggregator.

The problem here is that some validators may never respond for the aforementioned reason. Even though rational oracle nodes may be able to increase their benefits by not responding, they should still be encouraged to respond. The reason for this is that they minimize the risk of the aggregator not being able to get enough responses, which leads to a lower possible validation reward. Even if more than t validators behave altruistically, it may be the case that no signature can be created due to possible inconsistencies. This indicates that it is more beneficial for the validators to always respond to requests. Above all, validators do not have to perform any particularly resource-intensive tasks, which means that possible resource savings are low.

3) Sybil Attacks: Another important aspect that must be considered is to what extent Sybil attacks are possible and what the consequences are. A Sybil attack describes the threat that single faulty participants can control multiple identities, which enables them to compromise larger parts of a system. Douceur [24] has shown that it is not possible to prevent such attacks without a central authority except for conditions that are not practicable for large-scale distributed systems. Since we are in the area of blockchain technology where we usually do not have a central authority, we have to consider such attacks. Further, in permissionless blockchains, everyone is allowed to join the system, whereby it is an easy task to create new identities by simply generating a new key pair. The same applies to the proposed oracle solution where any number of oracle nodes can register.

We have already mentioned in Section III that oracle nodes have to deposit a stake to make Sybil attacks more difficult. The intention is to make the creation of new identities expensive so that it becomes more difficult to control larger parts of the oracle. On the other hand, we also make it more difficult for honest oracle nodes to join the oracle. Therefore, the trade-off for a more Sybil-resistant oracle is less decentralization considering only oracle nodes that can provide the stake can participate. Nevertheless, an attacker is still able to register multiple oracle nodes to be able to create more signature shares, which enables the attacker to gain more voting power.

In the worst case, an attacker could register threshold t or more oracle nodes and decide on the result alone if it can provide the necessary stake. To provide better Sybil resistance, we further encourage the creation of only one identity, by increasing the chance of winning the validation reward super-linearly based on the deposited stake. As a result, it is more beneficial for oracle nodes to create a single identity with a higher stake than to split the stake between multiple identities, as this gives them a greater chance of winning the validation reward. Therefore, if one is only interested in gaining more benefits, this approach resembles the rational choice.

4) Key Submission: The selected key submission mechanism also imposes some security risks. As has already been discussed, the public key can be submitted by using an on-chain aggregation mechanism, dispute mechanism, or another oracle solution for the key submission (see Section III-D). While the use of an oracle solution appears to be more cost-effective, we must be aware of the security risks that arise from this approach.

A central oracle solution represents a single point of failure, which can submit arbitrary public keys to then be able to create valid signatures. As a result, a single participant could gain full control over the oracle. However, if one uses a decentralized approach, the risk is lower.

It must be ensured that both oracles, i.e., the oracle used for the key submission and the interoperability oracle, are as independent of each other as possible. Otherwise, oracle nodes that participate in both oracles would be able to change the result in their favor. In the worst case, this would mean that a certain subset of oracle nodes could have complete control over the oracle used for key submission and can thus select the public key. This means that even if the attacker is unable to create more than threshold t oracle nodes in the interoperability oracle, it can get control over the interoperability oracle if it can control the oracle for key submission.

B. Cost Analysis

We analyze the costs of the proposed solution by comparing the implemented prototype with two alternative approaches which do not make use of BLS threshold signatures. Accordingly, we implement two additional oracle contracts (*On-chain Oracle*, *ECDSA Oracle*) which follow different aggregation mechanisms. Furthermore, we compare the costs of

our approach with the costs incurred by ETH Relay [23], a novel relay scheme (see Section VI-C), to examine how well our approach performs compared to state-of-the-art schemes. To ensure repeatability, the implementations, as well as the evaluation scripts, are also included in the open-source project on GitHub (see Section IV).

The *On-chain Oracle* implements an on-chain aggregation mechanism whereas each oracle node calls the oracle contract to submit a result. The *ECDSA Oracle* makes use of Elliptic Curve Digital Signature Algorithm (ECDSA) signatures to verify the result. In contrast to our proposed scheme, an aggregator does not submit a single BLS signature but rather submits several ECDSA signatures that are verified by the oracle contract. Since there is no reasonable source of randomness for either of these variants, the reward is paid out to each oracle node that is part of the majority.

For the experiment, we use a private Ethereum blockchain based on the Muir Glacier hard fork, on which we deploy all smart contracts. In this experiment, we request the verification of a transaction with every type of the aforementioned aggregation mechanisms. Besides, we are changing the number of participating oracle nodes to be able to determine how the costs develop with an increasing amount of oracle nodes.

By comparing the different mechanisms (see Figure 4), it can be seen that the costs for the on-chain aggregation mechanism are considerably higher than those of the other two. This is due to the reason that for the on-chain mechanism more storage space is needed and each participant has to create a transaction to submit its vote. In comparison, the other mechanism in which an aggregator only submits several ECDSA signatures, is more cost-efficient. The problem here, however, is that the costs continue to rise with the number of oracle nodes, even though the verification of ECDSA signatures is a relatively cheap operation on the Ethereum platform.

Our proposed solution based on BLS threshold signatures, on the other hand, causes almost constant costs that are independent of the number of oracle nodes, since only one signature has to be verified. The costs are only almost constant as the try and increment approach is used for hashing. As can be seen in Figure 5, submitting the result consumes on average 257,607 gas with a standard deviation of 21,671 gas. Verifying a BLS signature is an expensive operation, but it is considerably more cost-efficient as the number of oracle nodes increases. With more than three oracle nodes, it is already cheaper than the on-chain mechanism and with more than 15 nodes, it is also cheaper than the ECDSA mechanism. Therefore, by using this approach we can achieve a higher degree of decentralization without increasing the costs.

Now that we have compared our approach with two different oracle solutions, we examine how the costs differ compared to a relay solution. To conduct this analysis, we choose ETH Relay because it is an advanced relay solution that is specifically designed to be more cost-effective. For the comparison, we assume a period of 100 blocks. Every block header submission in ETH Relay consumes 284,041

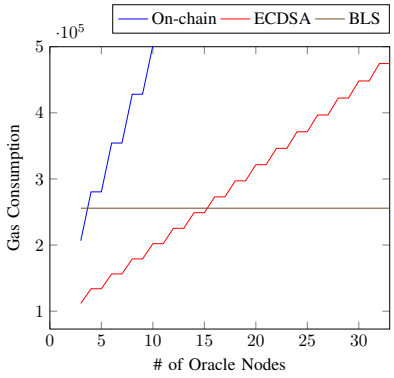

Fig. 4. Gas consumption of the different aggregation mechanisms

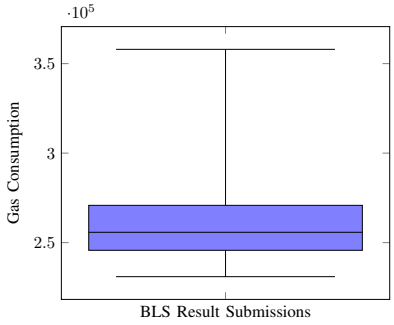

Fig. 5. Gas consumption of the result submission with BLS signatures

gas with a standard deviation of 3,679 gas. As a result, submitting 100 block headers consumes around 28,404,100 gas. Using our presented approach, we can submit 110 results that incur roughly the same gas costs as the 100 block header submissions of ETH Relay. The costs for the actual request must also be taken into account. In the case of ETH Relay, the relay contract needs to carry out a SPV and check the membership of the block in the longest chain, which means that the costs increase as the search depth increases. In the case of our oracle-based relay, however, a request only ever consists of an event being emitted, whereby the result can simply be accessed directly, which results in constant costs.

At the moment, the application of interoperability solutions is a rather infrequent occurrence, which presumably suggests that not many requests are made. In the worst case, this could mean that not a single transaction has to be verified for 100 blocks. With ETH Relay, the blocks still have to be submitted and thus the costs must be sustained. Hence, keeping the relay alive is a huge burden since these submitted block headers do not yield any profit for the submitter. In contrast, our presented oracle-based solution does not incur any costs in this scenario, as every request is fulfilled on demand. However, if the number of requests increases drastically such that it is more than 110, ETH Relay would be the more cost-efficient solution. One must also note that in this case further adjustments can be made to the oracle such that every request enables the verification of all the transactions within a block, rather than a single one, by providing the Merkle root of such a block.

VI. Related Work

So far, different blockchain interoperability solutions have been proposed. These include hash-locks, relay solutions, and oracles. In the course of this section, we examine solutions that are related to our work.

A. Hash-Locks

Hash-locks are a well-known technique to enable a basic form of blockchain interoperability without oracles and relays, e.g., to realize atomic cross-chain swaps [25]. Atomic swaps allow multiple parties to exchange their assets across multiple blockchains. The involved parties make use of Hashed Timelock Contracts (HTLCs) to escrow their assets with a hashlock h and timelock t. Ownership of the asset is only transferred if the receiver can provide the secret s such that $h(s) = h$ before t expires. However, attention must be paid to specify the right timelock values and use the correct deployment order of the contracts.

B. Relays

Frauenthaler et al. [23] propose ETH Relay, a novel relay scheme for Ethereum-based blockchains. A validation-on-demand pattern is used to keep the operating costs low by reducing the number of expensive full block header validations. Instead of validating every block header when it is submitted, off-chain clients have a certain time frame in which they can dispute submitted block headers. To make SPVs more efficient, the authors also optimize the traversal of the blockchain, by jumping from branching point to branching point instead of iterating over the whole data structure. While the authors achieve a remarkable cost reduction over traditional relay solutions, the costs remain quite high.

In [26], the authors present zkRelay, which is a relay solution that utilizes off-chain computations to validate batches of block headers through the usage of Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) proofs. Since these proofs are generated off-chain, the smart contract only needs to be able to verify the proof, removing the necessity of storing and validating every submitted block header, whereby only the last block header of a batch is stored. Although the solution offers improvements in the form of scalability and cost optimization, there are tradeoffs in terms of delay and hardware resources. Our solution, however, can immediately retrieve data from the source blockchain and has no excessive RAM consumption since no complex proofs are generated.

C. Oracles

Provable [27] (formerly known as Oraclize) is a centralized oracle service for various blockchain platforms, e.g., Ethereum and EOS. The Provable blockchain oracle utilizes TLSnotary authenticity proofs [28] to attest the authenticity of the data retrieved from the originating source. Within TLSnotary, an auditee can prove to an auditor the authenticity of information retrieved from a Web server that is using the Hypertext Transfer Protocol Secure (HTTPS) protocol, by utilizing the features of the underlying Transport Layer Security (TLS) protocols 1.0 and 1.1.

With Town Crier, the authors of [29] propose another centralized blockchain oracle, which uses the HTTPS/TLS protocol and additionally utilizes trusted hardware to ensure the authenticity of the data. The implementation uses Intel's Software Guard Extensions (SGX) which allows the execution of a process in a protected address space which guards the process against malicious software running outside of the enclave but also from various hardware attacks. While both of the aforementioned oracle services offer a solution to the oracle problem, the high level of centralization poses a major problem regarding scalability and single points of failure.

In [30], the authors present ChainLink, a decentralized oracle network. ChainLink offers a reputation-based voting system whereby users can issue queries to the ChainLink smart contracts. Queries are executed by the selected oracle nodes which retrieve the results from different or overlapping sets of data sources. These results are aggregated by a smart contract which is also responsible for the calculation of the outcome. Breidenbach et al. [31] further introduce a new off-chain reporting protocol for ChainLink, which however follows a different approach compared to our solution.

Peterson et al. [32] propose a decentralized oracle and prediction market platform called Augur. Within Augur, users can create prediction markets to get information that is external to the system. Market participants trade shares of those markets and reporters can vote by staking their REP tokens (Augur's native token) on one possible outcome. The reached consensus of reporters is considered as the outcome. Depending on the result, reporters receive a reporting fee from the markets. Augur's incentive mechanism encourages participants to behave honestly to maximize their profits, while misbehaving participants get penalized.

The authors of [33] propose ASTRAEA, another decentralized voting-based blockchain oracle. Submitters, voters, and certifiers play a voting game to decide on the truth value of boolean propositions. These propositions are added to the system by submitters, who pay fees to receive an answer to the submitted proposition. Voters play a low-risk/low-reward game by depositing a stake to answer a random proposition. Certifiers on the other hand play a high-risk/high-reward game whereby they can choose a proposition to certify but have to place a high stake.

Merlini et al. [34] propose an extension to this protocol to solve the lazy equilibrium problem, whereby all voters report the same answer on all propositions. The authors describe a paired-question protocol, in which submitters add queries with two antithetic questions, whereby the oracle additionally needs to check if both answers converge to different outcomes. With this approach, the protocol ensures that honest voters receive higher rewards than lazy voters.

While the approaches discussed above are interesting solutions to the oracle problem, they are not specifically aiming at providing blockchain interoperability and also implement their mechanisms on-chain, which results in high costs.

VII. Conclusion

Research on closing the gaps between different blockchains has already led to several concepts and solutions. However, these are usually too expensive or very resource-intensive.

To overcome these issues, we propose a voting-based blockchain interoperability oracle that uses an off-chain aggregation mechanism based on BLS threshold signatures. The oracle nodes are divided into one aggregator and multiple validators and generate a distributed private key to collectively decide on the result of a request. Validators read the data from the other blockchain and sign it with their private key share. The selected aggregator collects the results and the signature shares from the validators to create a valid signature which is submitted to and verified by the oracle contract. Our evaluation shows that the proposed solution is more cost-efficient than other oracle solutions and also incurs lower costs than state-of-the-art relay schemes depending on the request rate.

In future work, we will investigate how we can improve Sybil resistance and the submission of the shared public key. We will also examine if there is still potential to further reduce the costs by applying other signature schemes and enabling requests such that multiple transactions can be verified.

Acknowledgment

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association is gratefully acknowledged.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *International Journal of Production Research*, vol. 57, no. 7, pp. 2117–2135, 2019.

[3] E. J. De Aguiar, B. S. Faiçal, B. Krishnamachari, and J. Ueyama, "A survey of blockchain-based strategies for healthcare," *ACM Computing Surveys*, vol. 53, no. 2, 2020.

[4] S. Schulte, M. Sigwart, P. Frauenthaler, and M. Borkowski, "Towards blockchain interoperability," in *International Conference on Business Process Management*. Springer, 2019, pp. 3–10.

[5] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends," *arXiv:2005:14282*, 2020.

[6] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, 2014.

[7] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: review, comparison, and open research challenges," *IEEE Access*, vol. 8, pp. 85 675–85 685, 2020.

[8] J. Heiss, J. Eberhardt, and S. Tai, "From oracles to trustworthy data on-chaining systems," in *2019 IEEE International Conference on Blockchain*. IEEE, 2019, pp. 496–503.

[9] V. Buterin, "Chain interoperability," *R3 Research Paper*, 2016.

[10] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

[11] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *6th International Workshop on Theory and Practice in Public Key Cryptography*. Springer, 2002, pp. 31–46.

[12] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 416–432.

[13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[14] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science*. IEEE, 1987, pp. 427–438.

[15] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[16] ——, "A threshold cryptosystem without a trusted party," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 522–526.

[17] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 295–310.

[18] A. Kate and I. Goldberg, "Distributed key generation for the internet," in *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 2009, pp. 119–128.

[19] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures." in *2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1751–1767.

[20] Y. Cai, G. Fragkos, E. E. Tsiropoulou, and A. Veneris, "A truth-inducing sybil resistant decentralized blockchain oracle," in *2020 2nd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2020, pp. 128–135.

[21] S. Popov, "The tangle," *White paper*, 2018.

[22] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *Twentieth ACM Symposium on Operating Systems Principles*. ACM, 2005, pp. 45–58.

[23] P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober, and S. Schulte, "ETH relay: A cost-efficient relay for ethereum-based blockchains," in *2020 IEEE International Conference on Blockchain*. IEEE, 2020, pp. 204–213.

[24] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 251–260.

[25] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery, 2018, p. 245–254.

[26] M. Westerkamp and J. Eberhardt, "zkRelay: Facilitating Sidechains using zkSNARK-based Chain-Relays," in *2020 IEEE European Symposium on Security and Privacy Workshops*. IEEE, 2020, pp. 378–386.

[27] Provable. The provable blockchain oracle for modern dapps. Accessed: 2021-02-24. [Online]. Available: https://provable.xyz

[28] TLSnotary. (2014) Tlsnotary - a mechanism for independently audited https sessions. Accessed: 2021-03-01. [Online]. Available: https://tlsnotary.org/TLSNotary.pdf

[29] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, p. 270–282.

[30] S. Ellis, A. Juels, and S. Nazarov. (2017) Chainlink: A decentralized oracle network. Accessed: 2021-02-24. [Online]. Available: https://link.smartcontract.com/whitepaper

[31] L. Breidenbach, C. Cachin, A. Coventry, A. Juels, and A. Miller. (2021) Chainlink off-chain reporting protocol. Accessed: 2021-07-16. [Online]. Available: https://research.chain.link/ocr.pdf

[32] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander. (2019) Augur: a decentralized oracle and prediction market platform (v2.0). Accessed: 2021-02-24. [Online]. Available: https://augur.net/whitepaper.pdf

[33] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, "Astraea: A decentralized blockchain oracle," 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)*. IEEE, 2018, pp. 1145–1152.

[34] M. Merlini, N. Veira, R. Berryhill, and A. Veneris, "On public decentralized ledger oracles via a paired-question protocol," in *2019 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 2019, pp. 337–344.

9

Distributed Key Generation with Smart Contracts using zk-SNARKs

Michael Sober, Max Kobelt, Giulia Scaffino, Dominik Kaaser, and Stefan Schulte: *Distributed Key Generation with Smart Contracts using zk-SNARKs*. In *38th ACM/SIGAPP Symposium on Applied Computing, SAC 2023*. 2023

DOI: 10.1145/3555776.3577677

Keywords: *Distributed key generation, zero-knowledge proofs, smart contracts, blockchain*

Abstract. Distributed Key Generation (DKG) is an extensively researched topic as it is fundamental to threshold cryptosystems. Emerging technologies such as blockchains benefit massively from applying threshold cryptography in consensus protocols, randomness beacons, and threshold signatures. However, blockchains and smart contracts also enable further improvements of DKG protocols by providing a decentralized computation and communication platform. For that reason, we propose a DKG protocol that uses smart contracts to ensure the correct execution of the protocol, allow dynamic participation, and provide crypto-economic incentives to encourage honest behavior. The DKG protocol uses a dispute and key derivation mechanism based on Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) to reduce the costs of applying smart contracts by moving the computations off-chain, where the smart contract only verifies the correctness of the computation.

Distributed Key Generation with Smart Contracts using zk-SNARKs

Michael Sober
Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
TU Hamburg
Hamburg, Germany
michael.sober@tuhh.de

Max Kobelt
Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
TU Hamburg
Hamburg, Germany
max.kobelt@tuhh.de

Giulia Scaffino
Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
TU Wien
Vienna, Austria
giulia.scaffino@tuwien.ac.at

Dominik Kaaser
TU Hamburg
Hamburg, Germany
dominik.kaaser@tuhh.de

Stefan Schulte
Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
TU Hamburg
Hamburg, Germany
stefan.schulte@tuhh.de

ABSTRACT

Distributed Key Generation (DKG) is an extensively researched topic as it is fundamental to threshold cryptosystems. Emerging technologies such as blockchains benefit massively from applying threshold cryptography in consensus protocols, randomness beacons, and threshold signatures. However, blockchains and smart contracts also enable further improvements of DKG protocols by providing a decentralized computation and communication platform. For that reason, we propose a DKG protocol that uses smart contracts to ensure the correct execution of the protocol, allow dynamic participation, and provide crypto-economic incentives to encourage honest behavior. The DKG protocol uses a dispute and key derivation mechanism based on Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) to reduce the costs of applying smart contracts by moving the computations off-chain, where the smart contract only verifies the correctness of the computation.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; • **Computer systems organization** → **Distributed architectures**;

KEYWORDS

Distributed key generation, zero-knowledge proofs, smart contracts, blockchain

SAC '23, March 27-31, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3577677

ACM Reference Format:
Michael Sober, Max Kobelt, Giulia Scaffino, Dominik Kaaser, and Stefan Schulte. 2023. Distributed Key Generation with Smart Contracts using zk-SNARKs. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27-31, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3555776.3577677

1 INTRODUCTION

Threshold cryptography allows for the shared usage of cryptosystems with no central authority [16]. A threshold cryptosystem distributes trust among all participants, while conventional cryptosystems often involve a trusted third party. Fundamental to threshold cryptosystems is the ability to generate distributed private keys by executing a Distributed Key Generation (DKG) protocol. Such a protocol allows a group of participants to generate a distributed private key where each party only possesses a share of the private key while not having any knowledge about the private key itself. DKG protocols have been subject to ongoing research over the last two decades (see Section 2). Notably, the latest spike of interest in blockchain technology paved the way for new applications and improvements.

Blockchain technology is not only the underlying technology of cryptocurrencies like Bitcoin [35] but has also found applications in various other fields [13, 34]. A blockchain is a tamper-resistant distributed ledger of transactions managed by a peer-to-peer network of nodes. The second generation of blockchains, e.g., Ethereum [47], allows the execution of so-called smart contracts, which are deterministic programs stored on the blockchain. The decentralized nature of blockchains makes them an excellent application for threshold cryptosystems, e.g., for consensus, public randomness, or threshold signatures using a DKG protocol [31].

DKG protocols are important not only for applications in the field of blockchain technology but also the blockchain, especially smart contracts, can help to improve DKG protocols [41, 45]. Usually, in DKG protocols, misbehaving participants are excluded from the protocol's further execution and not held accountable for their

actions. Thus, trying to attack the protocol may seem beneficial as it does not entail serious consequences. As we will discuss in the paper at hand, a smart contract can augment a DKG protocol with crypto-economic incentives and the possibility for dynamic participation such that participants are encouraged to execute the protocol and behave honestly.

Further, external parties need to verify that a given public key was generated according to the DKG protocol's specification and belongs to a specific group of participants that generated the key. While it is immediately apparent to the participants, this verification process is more difficult for parties external to the protocol. Therefore, we discuss the utilization of smart contracts to ensure the correct execution of the DKG protocol and provide a transparent view of the actions taken by each participant.

While the application of smart contracts introduces additional overhead, there are mechanisms to keep it as low as possible. These include Verifiable Off-Chain Computations (VOCs) with the help of Zero-Knowledge Proofs (ZKPs), where a smart contract only needs to verify a short proof and does not need to perform all calculations on the blockchain [32].

Therefore, we propose a DKG protocol based on smart contracts. The smart contract allows the participants to execute the different phases of the protocol by facilitating the execution. The underlying blockchain constitutes the communication channel, whereby no direct connection between the participants is necessary. Further, the smart contract offers a lightweight dispute mechanism by enabling VOC with Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs). This mechanism allows for the punishment of misbehavior such as slashing provided collateral, thus making the participants accountable for their actions. Accordingly, we provide the protocol's specification and create a prototypical implementation for Ethereum Virtual Machine (EVM)-based blockchains. Further, we evaluate the costs of using the smart contract as well as the performance and memory usage for generating the proofs.

The rest of this paper is structured as follows: in Section 2, we present some background information and the underlying concepts. Then, we propose the protocol in Section 3 and describe its implementation in Section 4. We evaluate the protocol in Section 5 and discuss the related work in Section 6. Finally, we conclude the paper in Section 7.

2 BACKGROUND

In this section, we present the concepts behind the proposed protocol. For this, we briefly describe the most important aspects of Secret Sharing, Distributed Key Generation, and Zero-Knowledge Proofs.

2.1 Secret Sharing

Secret Sharing was introduced by Shamir [43] to divide a secret S into n pieces such that k pieces or more are required to recover S. Providing only $k-1$ parts, it is not possible to recover S and does not leak any information about S. Initially, a dealer picks a polynomial $f(x) = c_0 + c_1x + \cdots + c_{k-1}x^{k-1}$ of degree $k-1$ in which $c_0 = S$ and chooses the remaining coefficients $c_1 \dots c_{k-1}$ randomly [29].

After that, a dealer evaluates $f(x_i)$ with $x_i = 1 \dots n$ over a finite field $\mathbb{F}_p$ and sends the points $(x_i, f(x_i))$ to the participants through a private channel. For k distinct points, there is only one polynomial that passes through all of these points. Therefore, it is possible to use polynomial interpolation, e.g., Lagrange interpolation, to reconstruct the polynomial. With the collaboration of k shareholders, the shareholders can reconstruct the polynomial and evaluate $f(0)$ to recover S.

Shamir's scheme has the disadvantage that the dealer is a trusted party that can distribute incorrect shares. Further, shareholders can provide invalid shares during the reconstruction of the secret. This issue led to the emergence of Verifiable Secret Sharing (VSS) schemes. Ever since VSS first has been mentioned by Chor et al. [12], many different VSS schemes followed, e.g., [19, 38, 40, 42, 44], which exhibit distinct characteristics. In the following discussion, we limit ourselves to non-interactive VSS schemes, more specifically to Feldman's VSS scheme, as no additional communication between the dealer and the other participants is necessary.

In Feldman's VSS scheme, only the dealer sends messages, while all shareholders can verify if the received share is correct without sending any messages. In addition to sending the shares to each participant, the dealer broadcasts public commitments c_iG for $0 \leq i \leq k-1$ to the coefficients of the random polynomial, where G is a generator of a cyclic group of elliptic curve points. Every shareholder has access to the public polynomial $F(x)$ defined as

$$F(x) = c_0G + c_1Gx + \cdots + c_{k-1}Gx^{k-1}.$$

A shareholder can use this public polynomial to verify that the received share is correct by evaluating $f(x_i)G = F(x_i)$.

2.2 Distributed Key Generation

In threshold cryptography, no single party possesses all the necessary information to encrypt, decrypt or sign a message. Instead, a threshold cryptosystem requires that a certain subset of size at least t out of n parties need to collaborate on these actions. Unfortunately, with VSS schemes, there is still the problem that a dealer is a central authority that knows the shared secret. A dealer can still act without collaborating with other parties if it does not discard the secret, thus leading to some centralization of trust.

DKG enables multiple parties to collectively generate a distributed private key with no central point of trust. Each party only has a secret share of the private key, but neither party knows the complete private key, which is also never recovered. While many different DKG protocols already exist [1, 14, 15, 20, 27, 28, 31, 39], we limit ourselves to Pedersen's DKG [39] as it has been widely used to create threshold cryptosystems. Further, it forms the basis of our proposed protocol (see Section 3.5).

Pedersen's DKG protocol allows a set of participants $P = \{P_1, \dots, P_n\}$ to generate a distributed private key where at least t participants need to collaborate to use the private key. For that, every party P_i executing the DKG protocol shares a signed secret $s_{ij} = f_i(j)$ with all the other parties $P_j \in P \setminus P_i$ using Feldman's VSS scheme. Upon receiving an invalid share s_{ij} from any other participant, a participant can issue a complaint by publishing s_{ij} and the signature. More than t complaints against a specific participant ensure its disqualification, i.e., the exclusion from the protocol's execution. A participant P_j can compute its share of the

distributed private key by calculating $s_j = \sum_{i=1}^{i=n} s_{ij}$. Finally, the participants can compute the shared public key via the previously published commitments to the random polynomials by calculating $H(x) = \sum_{i=1}^{i=n} F_i$ and evaluating $H(0)$.

2.3 Zero-Knowledge Proofs

A ZKP allows a prover to convince a verifier that a statement is true without revealing any information, i.e., the verifier has zero knowledge besides the validity of the statement [23]. Many distinct blockchains utilize ZKPs to enhance privacy or scalability [7, 26]. Today the most popular ZKP systems comprise zk-SNARKs [6], Zero-Knowledge Succinct (Scalable) Transparent Arguments of Knowledge (zk-STARKs) [5], and Bulletproofs [11]. With the help of these proof systems, it is possible to verify computational integrity, i.e., whether the output of a program given some public and private inputs is correct according to its specification without re-executing the whole program to verify the given output. Instead, a verifier only needs to verify a small proof [33].

The verification complexity and proof size of zk-STARKs and Bulletproofs depend on the computational complexity of the program, while for zk-SNARKs, it is constant. A disadvantage is that zk-SNARKs depend on a trusted setup, whereas this dependency is non-existent for zk-STARKs and Bulletproofs. However, the short proof size and the constant verification complexity make zk-SNARKs an excellent candidate for the application within smart contracts. Therefore, in the following, we only focus on zk-SNARKs.

The first practical implementation of zk-SNARKs was the Pinocchio protocol [37], followed by Groth16 [25], which uses pairing-friendly elliptic curves to validate the proofs. For the protocol, a program is transformed into an arithmetic circuit, then a Rank-1 Constraint-System (R1CS), and finally encoded as a Quadratic Arithmetic Program (QAP). The output of a program is correct if the respective solution to the QAP is also valid. The succinctness and zero-knowledge aspect of such proofs come from using random sampling and homomorphic encryption.

As discussed in Section 4.2, the proposed DKG protocol requires executing elliptic curve operations within a circuit. Therefore, we need to clarify possible limitations which affect our solution. The limiting factor to implementing elliptic curve cryptography within a circuit is that a circuit uses modular arithmetic over a finite field $\mathbb{F}_r$. Therefore, only elliptic curves defined over $\mathbb{F}_r$ are usable within a circuit. These curves can then enable the implementation of specific cryptographic protocols [3]. Another possibility is to create a cycle of elliptic curves where the proof is generated using another curve and verify this proof in a circuit using $\mathbb{F}_r$.

3 SYSTEM

In this section, we propose a DKG protocol that uses smart contracts for decentralized computation and communication in combination with zk-SNARKs to reduce its costs. We start with a brief overview and continue with an explanation of the communication model, cost model, smart contracts, and dynamic participation. After that, we explain the design of the protocol and provide an exact specification, including the different phases: *Share Distribution*, *Dispute*, and *Key Derivation*.

The execution of the protocol starts with the *Share Distribution* phase. Each participant generates a secret and distributes the shares among the other participants using Feldman's VSS and a smart contract. Afterward, the protocol reaches the *Dispute* phase, during which the participants verify the received shares and issue disputes against dealers who distributed invalid shares. The accused dealer can generate a zk-SNARK to justify that the distributed share is correct and the dispute is unjustified. Not resolving a dispute leads to exclusion from the protocol and crypto-economic punishments. Finally, the protocol reaches the *Key Derivation* phase, in which every participant uses the correctly distributed shares to derive the shared public key. For submitting the public key to the smart contract, any participant can generate a zk-SNARK to prove that the derived public key is correct.

3.1 Communication Model

We assume a synchronous communication model in which all messages are delivered within a fixed time bound Δ and consider the blockchain a public broadcast channel where parties can send and receive messages.

Further, instead of establishing point-to-point connections between participants, the participants use the public broadcast channel to exchange direct messages. For that, the participants generate a public/private key pair and disclose the public key to all other participants using the blockchain. The participants use the public key and some unique round-specific information to generate a shared key with its counterparts. This shared key is then used as part of a symmetric key encryption procedure to encrypt communication over the public channel.

3.2 Cost Model

Additionally, we assume that a public blockchain requires transaction fees to be paid with the blockchain's native currency. The transaction fees incentivize miners to spend their computational resources to create new blocks and execute transactions. Further, the transaction fees help to protect the platform from misuse since wasting resources also entails considerable costs.

The transaction fees depend on the actual computation steps and required storage space. As such, transactions which only transfer value between different accounts incur lower transaction fees than transactions that trigger the execution of a smart contract performing complex calculations or require a lot of storage [10]. Especially, elliptic curve operations are very cost intensive. Therefore, our DKG protocol uses zk-SNARKs to move these computations off-chain.

3.3 Smart Contracts

Initially, the protocol requires a trusted setup ceremony, which is necessary for using zk-SNARKs (see Section 2.3). This procedure generates the proving and verification keys for the circuits of Alg. 1 and 2 (see Section 3.5) to off-chain the computations for justification and key derivation. The execution of the trusted setup generates a simulation trapdoor referred to as toxic waste, which needs to be discarded because it would enable the generation of fake proofs [8]. Therefore, it is recommended to use Multiparty Computation (MPC) to distribute the trust among multiple parties [9]. This approach

ensures that as long as one honest party discards its part of the secret, it prevents the possibility of recovering the overall secret.

After executing the trusted setup ceremony, the smart contracts are deployed on the blockchain. The protocol requires three different smart contracts: the *Justification*, *Key Derivation*, and *zkDKG* contracts. The *Justification* and *Key Derivation* contracts provide the same functionality as both contracts only have to verify zk-SNARKs for the different circuits used during the *Dispute* and *Key Derivation* phases of the protocol. They only differ with respect to the inputs and verification keys. The *zkDKG* contract manages the whole execution of the DKG protocol. The participants use it to distribute the shares, issue disputes, justify their actions during disputes, and derive the public key. As justification and key derivation are based on zk-SNARKs (see Sections 3.5.2 and 3.5.3), the *zkDKG* contract interacts with the *Justification* and *Key Derivation* contracts to verify the validity of the proofs.

Further, the *zkDKG* contract requires the configuration of different parameters, including the addresses of the verification contracts, threshold, time limits of the different phases, curve parameters, and others that might be necessary to dynamically form the set of participants.

3.4 Dynamic Participation

Finally, after finishing the deployment of the smart contracts, different parties can start the formation process to qualify as participants. In contrast to conventional DKG protocols, which do not use a blockchain, several options are available to dynamically form the set of participants, where we do not want to limit ourselves to a specific one. Here, the *zkDKG* contract defines and strictly enforces who is allowed to participate in the execution of the protocol.

Among other things, one can imagine a permissioned setting in which the *zkDKG* contract defines that only its owner can decide who is allowed to join. The owner only has to provide a predefined list of the selected parties to the smart contract. The *zkDKG* contract checks if the potential party is allowed to join and only after successful authentication adds it to the current set of participants. The problem here is that the owner represents a central authority.

However, other permissionless mechanisms potentially allow everyone to join without having a central authority. The *zkDKG* contract could implement a naive mechanism to allow everyone to join until reaching a certain number of participants as long as they deposit some tokens as collateral. However, this approach is impractical in most cases since it does not provide any Sybil resistance, which would allow anyone to register with multiple identities [17]. In the worst case, a single party could register enough identities to gain sole control and knowledge of the generated key.

An alternative solution is to let the potential participants stake tokens not only as collateral but also to let the number of tokens decide who is allowed to participate, as is done in systems based on Proof of Stake (PoS) [30]. Following this direction, the *zkDKG* contract specifies, e.g., that only the n parties with the highest deposited stake may participate. Should participants misbehave during the execution of the protocol, they might get penalized by slashing their stake. This approach encourages participants to behave honestly and improves Sybil resistance by encouraging the creation of only one identity staking enough tokens. Another modification of this approach would also allow regular token holders to delegate their tokens to potential candidates to elect them as participants.

3.5 Protocol

After clarifying the necessary steps prior to the execution of the protocol, we continue with its description and specification (see Protocol 1). For that, we examine the *Share Distribution*, *Dispute*, and *Key Derivation* phases and describe the different execution steps sequentially.

3.5.1 Share Distribution. After determining the participants, the actual execution of the protocol starts with the *Share Distribution* phase (see Step 1 of Protocol 1) in which every participant acts as a dealer in one of n parallel executions of Feldman's VSS. For that, a participant P_i has to retrieve the targeted threshold t from the *zkDKG* contract. The next step is to generate the random polynomial $f_i(x)$ of degree $t-1$ and commit to the polynomial's coefficients to create its public counterpart $F_i(x)$ with coefficients $C_i = \{c_0^i G, ..., c_{t-1}^i G\}$. After that, a participant generates its shares S_i by computing $f_i(j)$ for every other participant $P_j \in P \setminus P_i$. Each share $s_{ij} \in S_i$ gets encrypted using a symmetric encryption scheme (e.g., the One-Time-Pad [41]). For that, a participant retrieves the public keys of the other participants from the *zkDKG* contract to compute a shared key with each participant using the Diffie-Hellman key exchange. Additionally, to ensure that each shared key is unique, as it is necessary for the One-Time-Pad, we concatenate it with $c_0^i G$ and take its hash as the shared key.

Following, a participant only has to broadcast its key shares S_i and the coefficients C_i using the blockchain. For that, the *zkDKG* contract offers a *distributeShares* function which is only callable for registered participants during the *Share Distribution* phase of the protocol. The *zkDKG* contract also ensures that the participants can only submit the right number of shares and commitments. After these checks, the *zkDKG* contract stores $c_0^i G$ and only the hashes of S_i and C_i to save storage costs. Further, the *zkDKG* contract either notifies the participants, or the participants have to poll the *zkDKG* contract to check the current status. After that, a receiving participant P_j can query the blockchain to get their share $s_{ij} \in S_i$, decrypt it, and verify that $F_i(j) = f_i(j)G = s_{ij}G$. The *Share Distribution* phase ends after reaching the timeout or if all participants distributed their shares.

3.5.2 Dispute. After the distribution of the shares, the *zkDKG* contract reaches the *Dispute* phase (see Step 2 of Protocol 1) of the protocol. If a participant receives an invalid share, it can issue a dispute against the dealer, which then has to justify by proving the correctness of the distributed share. A participant issues disputes by calling the *zkDKG* contract's *dispute* function and providing the index of the dealer and the shares. The *zkDKG* contract verifies that it reached the *Dispute* phase, checks if the hash of the provided shares matches the stored hash and that no dispute concerning this dealer is ongoing. Afterward, the *zkDKG* contract notifies the dealer or respectively the dealer has to check if there are any ongoing disputes to resolve within a certain time limit. This time limit is extended with every dispute to provide the dealer enough time to justify.

Protocol 1 Distributed Key Generation Protocol

A set of participants $P = \{P_1, \ldots, P_n\}$ generates a distributed private key with threshold t using a *zkDKG* smart contract.

(1) **Share Distribution**
- (a) A participant $P_i \in P$ retrieves the threshold computed by the *zkDKG* contract.
- (b) P_i generates a random polynomial $f_i(x)$ of degree $t-1$ and its corresponding public polynomial $F_i(x)$ with coefficients $C_i = \{c_0^i G, ..., c_{t-1}^i G\}$.
- (c) P_i computes the set of shares $S_i = \{f_i(j) \mid 0 < j < n, j \neq i\}$ and encrypts each share $s_{ij} \in S_i$.
- (d) P_i calls the *distribute* function of the *zkDKG* contract and provides C_i and S_i. The *zkDKG* contract then verifies that:
  - (i) the distribution phase is ongoing,
  - (ii) P_i has not already distributed S_i,
  - (iii) provided $|S_i| = |P \setminus P_i|$ shares,
  - (iv) $|C_i| = t - 1$ coefficients

  and otherwise reverts.
- (e) The *zkDKG* contract stores $c_0^i G$, $hash(C_i)$, and $hash(S_i)$ and notifies P that P_i submitted a share.
- (f) A participant $P_j \in P$ verifies the received share s_{ij} by checking that $F_i(j) = f_i(j)G = s_{ij}G$. If the share is invalid, P_j continues with Step 2 and otherwise with Step 3.

(2) **Dispute** (optional)
- (a) P_j calls the *dispute* function of the *zkDKG* contract providing the index i of the disputee P_i and S_i. The *zkDKG* contract then verifies that:
  - (i) the dispute phase is ongoing,
  - (ii) P_i has not already been disputed,
  - (iii) S_i matches the stored $hash(S_i)$

  and otherwise reverts.
- (b) The *zkDKG* contract notifies P_i about the dispute including i, j, S_i, and the time limit τ to justify.
- (c) P_i generates a zk-SNARK π_J for the execution of Alg. 1 and uses the *justify* function of the *zkDKG* contract to submit π_J.
- (d) The *zkDKG* contract validates π_J and if π_J is valid resolves the dispute. In case π_J is invalid, the dispute remains unresolved.

(3) **Key Derivation**
- (a) The *zkDKG* contract verifies that the key derivation phase is ongoing.
- (b) The *zkDKG* contract checks for expired disputes and sets the coefficients of excluded participants to zero.
- (c) P_i derives the public key pk by calculating $H(x) = \sum_{j=1}^{j=n} F_j$ and evaluating $H(0)$.
- (d) Any participant generates a zk-SNARK π_D for the execution of Alg. 2 and uses the *derive* function of the *zkDKG* contract to submit pk and π_D.
- (e) The *zkDKG* contract verifies π_D, notifies P about the successful submission, and stores pk. If π_D is invalid the *zkDKG* contract reverts.
- (f) The *zkDKG* contract slashes disqualified participants.

This approach requires an additional interaction as the disputer first issues a dispute, and the respective dealer then has to justify its actions. By having the disputer generate a zk-SNARK and proving that the share is invalid, it is possible to avoid additional interactions with the dealer. However, the smart contract then needs to verify the inputs by the dealer during the *Share Distribution* phase to ensure that the receiver can generate a zk-SNARK which is only possible with correct input values. Therefore, it is better to leave it to the dealer, who has the incentive to provide correct inputs, because otherwise, the dealer cannot generate a valid zk-SNARK. This decreases the costs since there is no need for the smart contract to verify the inputs.

The dealer continues with the generation of the zk-SNARK for Alg. 1 to prove the correctness of the disputed share. For that, the dealer has to provide its commitments c, secret key sk, public key pk, the public key of the disputer pk_d, index i, encrypted share s_{enc}, and the hash of the public parameters h. The costs of verifying the zk-SNARK increase with the number of public input parameters. Therefore, we declare all public parameters as private and only pass h to verify them within the circuit by comparing the hashes. Further, the dealer has to provide the commitments as uncompressed points as decompressing is too complex within the circuit.

The program (see Alg. 1) for the justification of a disputed share starts by checking whether the dealer knows the secret key to the provided public key. After that, the program needs to verify that the dealer provided the correct public input parameters. For that, it hashes the commitments and subsequently hashes the result again with the remaining public input parameters. The hash is then compared with the hash provided by the dealer and only if both hashes match, it is possible to continue with the computation. Since the input parameters are correct, the program can derive the shared key from sk, pk_d, and c, to get the decrypted share s_{dec}. Finally, it checks if the public polynomial evaluates to $s_{dec}G$ and returns the result.

After computing the proof, the dealer calls the *justify* function of the *zkDKG* contract providing the previously computed proof. Initially, the *zkDKG* contract checks if there is a dispute to resolve. After that, the *zkDKG* contract computes the hash of the public input parameters and calls the *Justification* contract. The *zkDKG* contract reverts if the proof is invalid, and otherwise, it deletes the dispute and excludes the disputer for the unjustified dispute.

3.5.3 Key Derivation. After reaching the time limit and allowing all participants to verify their received shares and file a dispute, the final *Key Derivation* phase (see Step 3 of Protocol 1) of the protocol begins. A participant computes its overall share of the distributed private key by summing up the shares received from qualified participants, i.e., participants with no unresolved disputes. The derivation of the public key is very cost-intensive using a smart contract. Therefore, any participant can generate a zk-SNARK for Alg. 2 to derive the public key off-chain and only submit the public key with the proof to the *zkDKG* contract. A participant can derive the public key by computing $pk = \sum_{j=1}^{j=n} c_0^j G$. However, generating a proof of this computation also requires the hash of the public input parameters, i.e., the first public coefficients, to verify that a participant uses the correct input parameters.

Algorithm 1 Compute the justification that a distributed share is correct and was wrongly disputed

```
function JUSTIFY(c, sk, pk, pk_d, i, s_enc, h)
    assert(mult(sk, G) == pk)
    h_c ← hash(compress(c))
    h_c ← hash(h_c, pk, pk_d, i, s_enc)
    assert(h == h_c)
    s_dec ← s_enc − sharedKey(sk, pk_d, c)
    actual ← mult(s_dec, G)
    expected ← evalPubPoly(i, c)
    return actual == expected
end function
function SHAREDKEY(sk, pk, c)
    k ← scalarMult(sk, pk)
    return hash(k, c[0])
end function
function EVALPUBPOLY(i, c)
    result ← Infinity
    for k ← 1 to len(c) do
        result ← mult(result, i)
        result ← add(c[len(c) − 1 − k]
    end for
    return result
end function
```

After generating the proof, a participant calls the *derive* function of the *zkDKG* contract providing the public key and the proof. The *zkDKG* contract checks if it is in the *Key Derivation* phase and if participants have any unresolved disputes. If a participant has unresolved disputes, it is excluded by setting its first public coefficient to 0, i.e., the point at infinity. Further, the *zkDKG* contract aborts the execution of the protocol if the number of excluded participants exceeds the limit specified during deployment. In the worst case, the number of participants and the threshold are equal, meaning that a single failure is enough to make the distributed private key unusable. This case requires a restart of the protocol with an updated set of participants, i.e., misbehaving participants get replaced and lose their collateral.

After that, the *zkDKG* contract computes the hash of all the stored first coefficients and the submitted public key. Here also, only providing the hash of the public inputs reduces the verification costs. Then, the *zkDKG* contract calls the *Key Derivation* contract, providing the previously computed hash and the public key. The *zkDKG* contract stores the public key if the proof is valid and otherwise reverts. Another possibility is to let the *zkDKG* contract optimistically accept the public key and introduce another dispute round after the submission. This approach reduces the costs of submitting the public key and the necessary resources since the generation and verification of the proof are only required if the submitted public key is invalid. However, another dispute round introduces further complexity.

Finally, the *zkDKG* contract allows multiple runs such that only a single deployment is necessary to reduce the overall costs. It is up to the creator of the *zkDKG* contract to specify when to execute a reset.

Algorithm 2 Derive the distributed public key

```
function DERIVE(coef, h)
    h_c ← hash(compress(coef))
    assert(h == h_c)
    pk ← Infinity
    for k ← 1 to len(coef) do
        pk ← mult(pk, coef[i])
    end for
    return pk
end function
```

4 IMPLEMENTATION

In this section, we describe the implementation of the client software and the smart contracts available as open-source software on GitHub[1].

4.1 Smart Contracts

We implemented the smart contracts in Solidity for the Ethereum smart contract and decentralized application platform, as it is today the most popular smart contract platform with a wide range of tools and support [4]. Further, Ethereum offers precompiled contracts for elliptic curve addition, scalar multiplication, and pairing checks on the *alt_bn128* curve, which is necessary to verify the zk-SNARKs. Since the implemented prototype works for Ethereum, it also supports other Ethereum-based blockchains out of the box. However, the solution can also be implemented for other blockchains as long as these platforms provide the necessary smart contract capabilities and verification of zk-SNARKs.

We decided to implement the naive registration mechanism, which has already been discussed in Section 3.4, as it is sufficient for the first prototypical implementation and allowed us to focus on the more important parts of the protocol. The *zkDKG* contract only allows participants to join until reaching a certain number and requires them to provide collateral. The participants can withdraw their placed collateral from the *zkDKG* contract after the execution of the protocol if they behave honestly.

The used threshold depends on the number of registered participants n. The *zkDKG* contract computes $\lceil (n+1)/2 \rceil$, specifying that the reconstruction of the distributed private key requires more than half of the registered participants. Further, we define that at least $\lceil (2n+1)/3 \rceil$ need to successfully share their secrets to enable the submission of the public key to ensure that the system continues to work if participants misbehave. These are usually the minimum requirements considering the related work (see Section 6).

The *zkDKG* contract has to store the distributed shares and commitments of the participants. Storage space on the blockchain is one of the biggest cost factors. Therefore, the *zkDKG* contract keeps the storage as low as possible. For that, the *zkDKG* contract only stores the first coefficients of each participant directly in the smart contract's storage while leaving the shares and commitments as part of the *calldata* and only storing the Keccak256 hash, which is considerably cheaper. Unfortunately, Ethereum does not provide a cost-efficient implementation of arithmetization-oriented hash functions such as MiMC [2], Poseidon [24], or Rescue-Prime [46],

[1] https://github.com/soberm/zkDKG

which would make computing the hash inside a circuit more efficient. The *zkDKG* contract emits an event to notify all participants about the recent distribution from a specific participant.

The *Justification* and *Key Derivation* contracts required no manual implementation. For these functionalities, the ZoKrates toolbox (see Section 4.2) provides the possibility to generate the verification contracts for the respective circuits. These generated contracts use the precompiled contracts provided by Ethereum to ensure the cost-efficient verification of the proofs.

4.2 Client

In addition to the smart contracts, we also provide a prototypical implementation of the client software written in the Go programming language. The client software is the off-chain component that uses the *zkDKG* contract to generate the distributed private key. For generating the zk-SNARKs, we use the ZoKrates [18] toolbox, which enables zk-SNARKs on Ethereum. ZoKrates offers a Domain-Specific Language (DSL) to create off-chain programs and compile them to arithmetic circuits. After that, a client can use ZoKrates to compute the witness for a specific circuit and generate a proof of computation using the selected proving scheme. We selected Groth16 as the proofs only consist of three elliptic curve points, which makes the verification on the blockchain cheaper.

We used the DSL provided by ZoKrates to implement Alg. 1 and 2. The standard library of ZoKrates already provides the implementation of elliptic curve operations over the Baby Jubjub curve [3]. Hence, our prototype uses this curve for elliptic curve cryptography. However, the protocol supports any other curve as long as its base field matches the scalar field of the curve used for the zk-SNARKs.

The client software uses Go language bindings for Ethereum smart contracts to avoid boilerplate code and directly interact with the smart contracts through their interfaces. Unfortunately, ZoKrates does not provide Go bindings. Therefore, the client uses Docker to spin up new containers using the ZoKrates Docker image and execute the respective commands to compute the witness and generate the proofs for Alg. 1 and 2 within a container. The client retrieves the result from the container to submit the proof to the *zkDKG* contract when executing the *justify* or *derive* functions.

5 EVALUATION

After describing the implementation of the protocol, we use this section to evaluate the protocol concerning costs, performance, and memory. The results give insight into the overhead introduced by smart contracts and the necessary resources needed to generate the proofs for justification and key derivation.

We execute the protocol ten times each before doubling the number of participants until we reach 256 participants resulting in 70 protocol runs. During the executions, we measure the gas consumption, proof generation time, and memory usage. We perform the experiments on an Amazon EC2 instance equipped with an Intel(R) Xeon(R) Platinum 8259CL running Ubuntu 22.04 using eight cores with a clock speed of 2.50GHz. The machine has access to 32GB DDR4 RAM with a frequency of 3200MHz and an SSD with a throughput of 128 MB/s max. Further, we spin up a local blockchain instance using Hardhat network 2.11.1 to deploy and run the smart contracts.

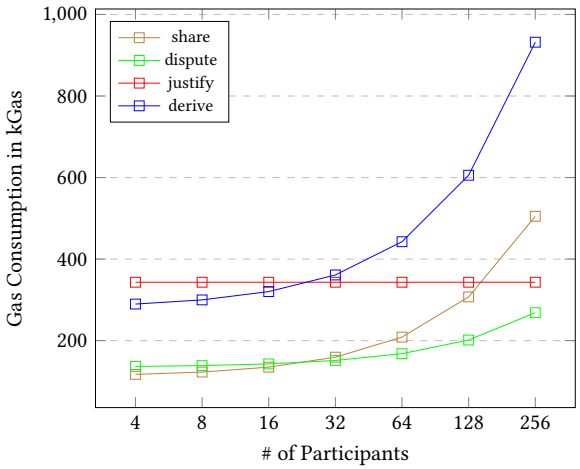

Figure 1: Gas consumption of the different phases

5.1 Costs

The execution of smart contracts on the Ethereum platform requires gas to protect the platform by preventing the unnecessary consumption of resources. Further, it serves as a reward for miners and incentivizes them to provide their resources. However, this is not only an inherent characteristic of the Ethereum platform but also of other smart contract platforms that need to compensate miners. Therefore, we examine the costs incurred by the protocol through executing smart contracts. We specifically look at the gas consumption of the *share*, *dispute*, *justify* and *derive* functions.

The results (see Fig. 1) show that the *share* and *dispute* functions consume the least amount of gas. The gas consumption of the *share* function averages 117 kGas for 4 participants reaching 505 kGas for 256 participants. These costs are relatively low since the shares and commitments are part of the calldata, while the *zkDKG* contract only stores the respective hash values. The *dispute* function consumes even less gas since only the number of shares impacts its gas consumption. Here we get 137 kGas for 4 participants and 269 kGas for 256 participants. However, the gas consumption of both functions increases linearly with the number of participants that execute the protocol.

Further, we can see that the gas consumption of the *justify* function is constant at 343 kGas respecting the number of participants. The reason is that the *justify* function only expects the zk-SNARK consisting of three elliptic curve points as input. The *zkDKG* contract does not execute any operations based on the number of participants and removes the dispute upon successful proof verification. However, the gas consumption increases slightly if multiple disputes are stored since the smart contract needs to loop over all inputs to find the respective index.

On the contrary, executing the *derive* function consumes more gas with an increasing number of participants since the *zkDKG* contract needs to compute the hash of all first coefficients to verify the zk-SNARK. The results show that executing the *derive* function consumes 290 kGas for 4 participants up to 932 kGas for 256 participants. The number of expired disputed also increases the gas consumption since the smart contract has to exclude the nodes

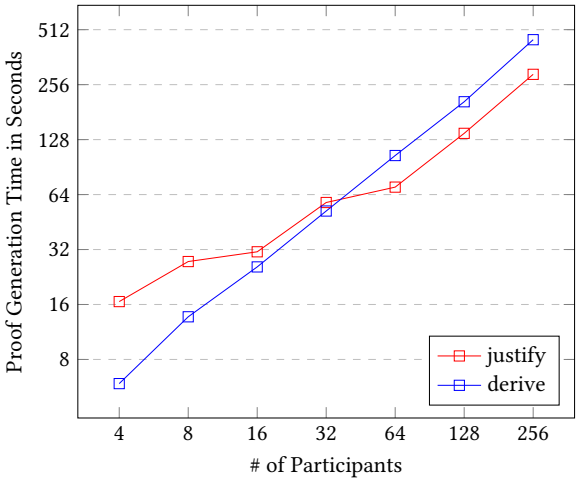

Figure 2: Time to generate the proofs

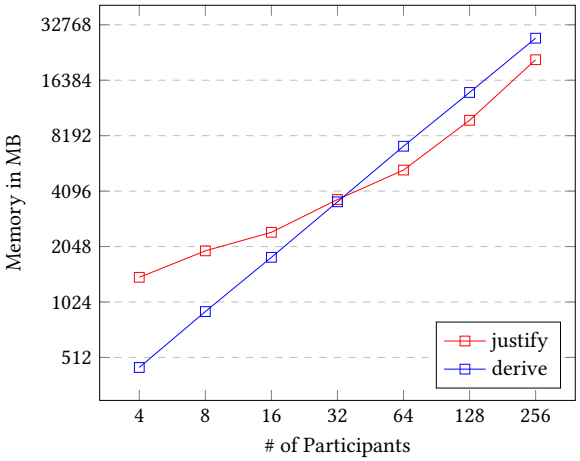

Figure 3: Memory usage during proof generation

before deriving the public key. However, since disputes are an exception, the impact on gas consumption is low.

5.2 Performance

Not only do the smart contracts introduce additional overhead but also the generation of zk-SNARKs for executing the *justify* and *derive* functions. Although this approach saves costs when executing the smart contracts, all participants have to use more of their local resources to generate the proofs. An interesting aspect here is how much time a participant spends on generating these proofs with an increasing number of participants.

The results of the executed experiments (see Fig. 2) show that proof generation time for justification and key derivation linearly depends on the number of participants. The proof generation time for key derivation gets higher and grows faster than for justification. The difference is that the former directly depends on the number of participants, while the latter depends on the threshold. The generation of the proofs with 4 participants takes 17 and 6 seconds, respectively. However, it already takes 292 and 451 seconds with 256 participants.

The proof generation time is usually not a large concern since blockchains are not suitable for time-critical applications. However, configuring the timeouts of the *zkDKG* contract requires attention to ensure that each participant has enough time to generate the proofs, which is particularly important for justification. Further, there is also a lot of optimization potential to reduce the proof generation time by using arithmetization-oriented hash functions and enabling multi-threading for ZoKrates.

5.3 Memory

Additionally, we investigate memory consumption during proof generation. Memory consumption is usually a bigger concern than proof generation time since memory consumption tends to be very high, considering that certain computations (e.g., keccak256) are not very efficient with zk-SNARKs. Here, we also examine how the number of participants influences memory consumption.

The measurements of memory consumption (see Fig. 3) show that it also increases linearly with the number of participants. The memory consumption behaves the same as the proof generation time. The generation of the proof for justification consumes 1.39 GB for 4 participants up to 21.2 GB for 256 participants. For key derivation, we get a memory consumption of 452 MB for 4 participants and 27.7 GB for 256 participants. Therefore, we get a higher entry barrier for the participants since they need the resources to ensure they can generate the proofs. However, there is also potential to optimize memory consumption by using arithmetization-oriented hash functions.

6 RELATED WORK

DKG protocols have always received a lot of attention in research, as they constitute the primary building block of threshold cryptosystems. Although blockchain technology already benefits from the application of threshold cryptography, only a few works, to the best of our knowledge, explore the application of smart contracts to enhance the protocols themselves.

In 1991, Pedersen [39] introduced the first DKG protocol, which allows multiple participants to generate a shared secret, where each party only possesses a share of the secret and has no further information about the shared secret. Every participant acts as a dealer in one of n parallel executions of Feldman's VSS scheme (see Section 2). In later work, Gennaro et al. [20] propose the Joint-Feldman DKG protocol, which builds upon Pedersen's DKG protocol but also ensures the uniform distribution of the shared secret. Pedersen's DKG protocol provides the foundation for our proposed protocol. While it does not guarantee uniform randomness of the shared secret, it remains secure, and only specific use cases require this property [21, 22].

Kate and Goldberg [27] propose the first DKG protocol, which works in an asynchronous network setting. For that, the authors define an asynchronous VSS scheme adopting a hybrid model for a network with $n > 3t+2f+1$ nodes, where an adversary may control t nodes, and f nodes can fail. The VSS protocol includes a sharing and reconstruction protocol and a recovery mechanism. The work

shows that asynchronous DKG protocols require a Byzantine agreement scheme. Therefore, the authors use a leader-based agreement protocol in combination with the previously specified VSS scheme to create a DKG protocol. In subsequent work, Kate et al. [28] provide another protocol version to guarantee uniform randomness of the shared secret. Further, they provide an implementation of the protocol and analyze its performance.

In [31], the authors present another asynchronous DKG protocol. However, the protocol enables the generation of a shared secret with a dual $(f, 2f + 1)$ threshold with f faulty participants. The authors introduce an asynchronous high-threshold VSS scheme that separates the reconstruction and recovery thresholds using an asymmetric bivariate polynomial. This VSS scheme forms the basis for a weak DKG protocol which eventually leads to a common key. By adding another mechanism to produce common randomness and implementing an agreement protocol, the authors were able to create the final asynchronous DKG protocol. Following a different approach, Abraham et al. [1] present a more efficient solution improving on the results presented in [31]. Further, a work by Das et al. [14] also promises improvements while providing compatibility with current threshold cryptosystems.

Schindler et al. [41] present a DKG protocol that uses Ethereum as its communication and verification platform. The protocol is based on the Joint-Feldman DKG protocol [20] and incorporates further improvements from [36] to address biasing attacks. The application of blockchain technology allows for the dynamic selection of participants and the augmentation of the protocol with crypto-economic incentives. Similar to our protocol (see Section 3.5), it uses Feldman's VSS to create and distribute the shares through the blockchain. However, there are some differences regarding the management of disputes. The authors describe a dispute mechanism where shareholders have to publish an invalid share and the key to decrypt the share. A disputer has to provide a Non-interactive Zero-Knowledge Proof (NIZK) to show the equality of two discrete logarithms to ensure the correctness of the published key. After that, a smart contract can then decrypt and verify the share. In our protocol, we stick to the notion that a dealer needs to prove that it behaved correctly instead of letting the shareholder prove that the dealer violated the protocol. Further, our protocol requires that the dealer submits a zk-SNARK to prove that it behaved correctly, which reduces the on-chain computations to a bare minimum. Despite these differences, the approach by Schindler et al. comes closest to the work at hand.

In [45], the authors present a smart contract for Threshold Information Disclosure (TID) based upon the work in [41]. This approach allows users to disclose certain pieces of information simultaneously such that all or no users disclose their information. The main building block of the TID mechanism is a DKG protocol that allows for the scheduled reconstruction of the private key. The authors extended and simplified the DKG protocol by Schindler et al. [41] to generate a threshold ElGamal key pair for encryption and decryption, ultimately making it more gas-efficient for the TID use case. After t or more participants have contributed to restoring the private key, any participant can submit it to the smart contract. The smart contract enables the disclosure of the information to the public by providing the information and the submitted private key. While the authors were able to optimize the DKG protocol, it is only applicable to specific use cases.

The authors of [15] propose a blockchain-based DKG protocol that ensures privacy and traceability. The protocol utilizes a blockchain, e.g., Bitcoin, with a pegged sidechain to improve scalability. Here, the main blockchain only acts as the coordinator and enables the anonymization of participants through a coin mixing service. The participants lock their coins on the main blockchain, generate the shares using the protocol from [22] and execute the sharing phase on the pegged sidechain. Participants can also issue dispute claims on the sidechain in case another participant violates the rules of the protocol. Finally, the participants can unlock their coins after successfully executing the protocol. While this solution provides better scalability through the pegged sidechain, it also introduces additional complexity.

Zhang et al. [48] present a publicly verifiable DKG protocol that uses Ethereum for trustless computation and communication. The authors explore the application of Ciphertext-Policy Attribute-Based Encryption (CP-ABE) instead of Publicly Verifiable Secret Sharing (PVSS) to lower the communication and computation overhead. Similar to PVSS, CP-ABE also enables the participants to create a shared publicly verifiable secret. The protocol only needs a single round as the smart contracts directly verify the proofs, thus removing the dispute phase. Unfortunately, the protocol incurs very high verification costs.

While our solution also entails a cost, time, and memory overhead, it is more cost-efficient than the current solutions based on blockchain technology since we move the expensive computations off-chain. Further, the availability of additional precompiled contracts to enable complex cryptographic operations besides the verification of the proof does not restrict our solution since there is no need for the smart contract to execute these complex calculations on the blockchain.

7 CONCLUSION

Threshold cryptography is an essential building block for creating threshold cryptosystems used by various applications, especially in the blockchain field. In this work, we showed that blockchains also open new possibilities which help to advance threshold cryptosystems by offering a platform for decentralized computation and communication.

For this, we introduced a DKG generation protocol that uses the potential of integrating blockchain technology into the protocol. Smart contracts enable the dynamic participation of participants, augment the protocol with crypto-economic incentives to establish accountability, ensure the correct execution of the protocol and provide a public broadcast channel through the blockchain. The protocol enables participants to use zk-SNARKs for justification and key derivation to execute heavy computations off-chain to save gas costs. We implemented the proposed protocol for the Ethereum smart contract platform and evaluated it regarding costs, performance, and memory to demonstrate its applicability.

In future work, we want to examine if there is potential to reduce the overhead of using smart contracts even further and integrate improvements from other protocols.

ACKNOWLEDGMENT

The financial support from the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, and the Christian Doppler Research Association is gratefully acknowledged.

REFERENCES

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *2021 ACM Symposium on Principles of Distributed Computing*. ACM, 363–373.

[2] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 191–219.

[3] Marta Bellés-Muñoz, Barry Whitehat, Jordi Baylina, Vanesa Daza, and Jose Luis Muñoz-Tapia. 2021. Twisted Edwards elliptic curves for zero-knowledge circuits. *Mathematics* 9, 23 (2021), 3022.

[4] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. 2019. A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3796–3838.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable zero knowledge with no trusted setup. In *39th Annual International Cryptology Conference*. Springer, 701–732.

[6] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium*. USENIX Association, 781–796.

[7] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Mina: Decentralized Cryptocurrency at Scale. https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf Accessed: 2022-06-24.

[8] Sean Bowe, Ariel Gabizon, and Matthew D Green. 2018. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. In *International Conference on Financial Cryptography and Data Security*. Springer, 64–77.

[9] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. *Cryptology ePrint Archive* (2017).

[10] Tamara Brandstätter, Stefan Schulte, Jürgen Cito, and Michael Borkowski. 2020. Characterizing Efficiency Optimizations in Solidity Smart Contracts. In *2020 IEEE International Conference on Blockchain (Blockchain)*. 281–290.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*. IEEE, 315–334.

[12] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science*. IEEE, 383–395.

[13] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. 2019. Blockchain for Internet of Things: A survey. *IEEE Internet of Things* 6, 5 (2019), 8076–8094.

[14] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2021. Practical asynchronous distributed key generation. *Cryptology ePrint Archive* (2021).

[15] Marina Dehez-Clementi, Jérôme Lacan, Jean-Christophe Deneuville, Hassan Asghar, and Dali Kaafar. 2021. A Blockchain-enabled Anonymous-yet-Traceable Distributed Key Generation. In *2021 IEEE International Conference on Blockchain*. IEEE, 257–265.

[16] Yvo G Desmedt. 1994. Threshold cryptography. *European Transactions on Telecommunications* 5, 4 (1994), 449–458.

[17] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[18] Jacob Eberhardt and Stefan Tai. 2018. 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)*. IEEE, 1084–1091.

[19] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*. IEEE, 427–438.

[20] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 295–310.

[21] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2002. Revisiting the distributed key generation for discrete-log based Cryptosystems. *RSA Security'03* (2002).

[22] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2003. Secure applications of Pedersen's distributed key generation protocol. In *Cryptographers' Track at the RSA Conference*. Springer, 373–390.

[23] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* 18, 1 (1989), 186–208.

[24] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *30th USENIX Security Symposium*. USENIX Association, 519–535.

[25] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 305–326.

[26] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash Protocol Specication. https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf Accessed: 2022-06-24.

[27] Aniket Kate and Ian Goldberg. 2009. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*. IEEE, 119–128.

[28] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed key generation in the wild. *Cryptology ePrint Archive* (2012).

[29] Jonathan Katz and Yehuda Lindell. 2020. *Introduction to modern cryptography*. CRC press.

[30] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Annual International Cryptology Conference*. Springer, 357–388.

[31] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1751–1767.

[32] Benjamin Körbel, Marten Sigwart, Philip Frauenthaler, Michael Sober, and Stefan Schulte. 2021. Blockchain-based result verification for computation offloading. In *International Conference on Service-Oriented Computing*. Springer, 99–115.

[33] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. 2020. {MIRAGE}: Succinct Arguments for Randomized Algorithms with Applications to Universal {zk-SNARKs}. In *29th USENIX Security Symposium (USENIX Security 20)*. 2129–2146.

[34] Muhammad Baqer Mollah, Jun Zhao, Dusit Niyato, Kwok-Yan Lam, Xin Zhang, Amer MYM Ghias, Leong Hai Koh, and Lei Yang. 2020. Blockchain for future smart grid: A comprehensive survey. *IEEE Internet of Things* 8, 1 (2020), 18–43.

[35] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf Accessed 2022-07-02.

[36] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. 2016. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks* 9, 17 (2016), 4585–4595.

[37] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 238–252.

[38] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*. Springer, 129–140.

[39] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 522–526.

[40] Tal Rabin and Michael Ben-Or. 1989. Verifiable secret sharing and multiparty protocols with honest majority. In *Twenty-first Annual ACM Symposium on Theory of Computing*. ACM, 73–85.

[41] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2019. Ethdkg: Distributed key generation with ethereum smart contracts. *Cryptology ePrint Archive* (2019).

[42] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*. Springer, 148–164.

[43] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[44] Markus Stadler. 1996. Publicly verifiable secret sharing. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 190–199.

[45] Oliver Stengele, Markus Raiber, Jörn Müller-Quade, and Hannes Hartenstein. 2021. ETHTID: Deployable threshold information disclosure on Ethereum. In *2021 Third International Conference on Blockchain Computing and Applications*. IEEE, 127–134.

[46] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. 2020. Rescue-prime: a standard specification (SoK). *Cryptology ePrint Archive* (2020).

[47] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[48] Liang Zhang, Feiyang Qiu, Feng Hao, and Haibin Kan. 2022. 1-Round Distributed Key Generation With Efficient Reconstruction Using Decentralized CP-ABE. *IEEE Transactions on Information Forensics and Security* 17 (2022), 894–907.

10

A Framework for Asynchronous Cross-Blockchain Smart Contract Calls

DOI: 10.1109/BCCA58897.2023.10338866

Keywords: *blockchain interoperability, smart contracts, cross-blockchain communication*

Abstract. Missing interoperability is a prevalent issue in today's blockchain landscape. Due to this lack of interoperability, only smart contracts deployed on the same blockchain can call each other. Smart contract interactions across the boundaries of blockchains would, however, provide an opportunity to create cross-blockchain applications. Hence, an additional mechanism for the execution of cross-blockchain smart contract calls is desirable. Unfortunately, already proposed solutions usually require one or more trusted intermediaries, which eventually leads to a lower degree of decentralization.

Therefore, we propose a novel framework to enable smart contract interoperability across different blockchains. The framework enables the execution of asynchronous cross-blockchain smart contract calls. It uses blockchain relays to maintain a high degree of decentralization while placing only minor trust in intermediaries. We provide a prototypical implementation for Ethereum-based blockchains and evaluate it regarding costs and execution time.

A Framework for Asynchronous Cross-Blockchain Smart Contract Calls

Michael Sober*†, Giulia Scaffino*‡, Marten Sigwart‡, Philipp Frauenthaler‡,
Markus Levonyak§, Stefan Schulte*†
*Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
†TU Hamburg, Hamburg, Germany, {michael.sober, stefan.schulte}@tuhh.de
‡TU Wien, Vienna, Austria, {giulia.scaffino, marten.sigwart, philipp.frauenthaler}@tuwien.ac.at
§Pantos GmbH, Vienna, Austria, markus.levonyak@bitpanda.com

***Abstract*—Missing interoperability is a prevalent issue in today's blockchain landscape. Due to this lack of interoperability, only smart contracts deployed on the same blockchain can call each other. Smart contract interactions across the boundaries of blockchains would, however, provide an opportunity to create cross-blockchain applications. Hence, an additional mechanism for the execution of cross-blockchain smart contract calls is desirable. Unfortunately, already proposed solutions usually require one or more trusted intermediaries, which eventually leads to a lower degree of decentralization.**

Therefore, we propose a novel framework to enable smart contract interoperability across different blockchains. The framework enables the execution of asynchronous cross-blockchain smart contract calls. It uses blockchain relays to maintain a high degree of decentralization while placing only minor trust in intermediaries. We provide a prototypical implementation for Ethereum-based blockchains and evaluate it regarding costs and execution time.

***Index Terms*—blockchain interoperability, smart contracts, cross-blockchain communication**

I. Introduction

With Bitcoin [1], blockchain technology found its first application in the design of a payment system that does not rely on a third party. After that, the advent of the second generation of blockchains, such as Ethereum [2], introduced smart contracts. Smart Contracts are deterministic programs stored on the blockchain and executed by multiple or all nodes of the underlying peer-to-peer network. With this and further recent advancements in research and development, blockchains have also found applications in many other areas besides cryptocurrencies [3]–[5].

However, this development has also led to high fragmentation in the blockchain space [6], [7]. Blockchains are, for the most part, closed worlds that are not interoperable with external systems. This lack of interoperability means that users depend on the initially selected blockchain platform. Switching to a new platform with other characteristics is usually not possible, resulting in so-called vendor lock-ins. Thus, users cannot fully benefit from other blockchain platforms' new features or properties.

The lack of interoperability is particularly noticeable when using smart contracts. A smart contract can only modify its state on the hosting blockchain while having no access to other blockchains or external systems. Often, however, it is the case that smart contracts need data from outside its hosting blockchain.

Allowing interactions of smart contracts running on different blockchains takes the hassle out of choosing a single right platform for users and allows them to collaborate even though different blockchains are used. However, users are not the only beneficiaries of improved blockchain interoperability. These also include the developers of decentralized applications since they no longer have to limit themselves to one blockchain. Developers can choose the right tool for the job and also benefit from code reuse by using smart contracts running on different blockchains. This leads to shorter development times, resource savings, and less code redundancy, enabling the development of higher-quality (cross-blockchain) applications.

For that reason, we present a framework inspired by the Remote Procedure Call (RPC) protocol to enable interaction between smart contracts residing on different blockchains that use the Ethereum Virtual Machine (EVM). The smart contracts can call each other's functions and return the result to the caller. The framework provides the opportunity to create stub contracts which serve as an abstraction layer to hide the background details of cross-blockchain communication. By using blockchain relays, the stubs can transfer information to other blockchains while only requiring one honest intermediary for the correct execution of the relays. A client contract can conveniently call a stub contract to call a smart contract on another blockchain, as the transaction sender facilitates the communication between the stubs by forwarding the necessary information with the respective proof data.

Additionally, we implement a prototype for two popular EVM-based blockchains in Solidity. We discuss possible extensions of the framework to support other blockchains and evaluate the costs and execution time of asynchronous cross-blockchain smart contract calls.

The rest of this paper is structured as follows: In Section II, we discuss related work. Afterward, we describe the background concepts applied in our solution in Section III. Following, we present the enabling framework for cross-blockchain smart contract calls in Section IV and deliver the evaluation in Section V. After that, we discuss security and possible extensions in Section VI and conclude the paper in Section VII.

II. Related Work

To the best of our knowledge, cross-blockchain smart contract calls are not covered extensively in recent research. Nevertheless, we discuss some existing concepts and solutions.

Nissl et al. [8] propose a framework that enables smart contract invocations across multiple blockchains. Initially, a caller registers a call on the source blockchain, for which intermediaries can place offers. After the caller accepts an offer, the intermediary forwards the request to the target blockchain and returns the result. Then, the validators determine the correctness and finalize the call. While the authors offer an interesting solution to enable cross-chain smart contract calls, it relies on multiple trusted validators and entails a considerable time and cost overhead.

Robinson and Ramesh [9] describe a general-purpose atomic cross-chain transaction protocol for Ethereum in which events transfer the information between the participating blockchains. A threshold number of trusted validators must sign these events and execute a simulation of a call graph. On each blockchain, a *Cross-blockchain Control Contract* offers the necessary functions to execute the cross-chain calls in an atomic and synchronous way. Similar to our framework (see Section IV), this solution also uses Ethereum events to transfer information. However, it relies on a trusted set of signers and offers synchronous function calls. In [10], Robinson further discusses the performance implications of atomic cross-chain transactions.

Westerkamp and Küpper present a solution to enable the synchronization of smart contracts initially deployed on a specific blockchain with their respective counterparts on other blockchains [11]. These client contracts mirror the logic and state of the original smart contract and provide instant read-only access to the target blockchain. Instead of replaying all transactions on the target blockchain, the solution uses state proofs and transition confirmations to synchronize contract states. While this solution allows smart contracts on the target blockchain to retrieve information from a remotely hosted smart contract, it is not possible to execute write operations.

The Ion Interoperability Framework [12] emerged as a part of the Clearmatics Ion project to provide an interface to create cross-blockchain smart contracts. The execution of these smart contracts is dependent on state transitions that have already occurred on another blockchain. In the case of Ethereum, information is passed as events that are part of the transaction receipt. The framework provides a mechanism to check the stored state from another blockchain. Based on these checks, the execution of a smart contract is triggered. This solution differs from our framework in that smart contracts never call each other directly.

HyperService [13] is a platform for creating cross-blockchain applications. For this, a programming framework provides the necessary abstractions, including a high-level programming language and a unified state model. Further, the authors describe a protocol that enables the correct execution of decentralized applications, requiring multiple transactions to be posted on different blockchains. A blockchain of blockchains provides a transparent view of the execution of the decentralized applications while misbehaving parties are being held accountable for their actions. On the other hand, our approach does not require an additional blockchain. Further, developers can still use existing programming languages.

Cosmos [14] is a multi-blockchain network that aims, among other things, to improve blockchain interoperability. For communication, Cosmos makes use of the Interblockchain Communication (IBC) protocol proposed in [15]. The IBC protocol allows multiple ledgers to establish connections with each other to create channels via which packages can be transmitted to different smart contracts. Merkle proofs provide the necessary evidence that packages were sent and received. Unfortunately, Cosmos does not support already existing blockchains out of the box. The integration requires peg zones that act as adaptors.

Polkadot [16] is another multi-blockchain framework. A central relay blockchain is responsible for the system's coordination and connects multiple parachains to enable interchain transactions. Cross-chain communication is enabled through the Cross-chain Message Passing (XCMP) protocol which uses a queuing mechanism based on Merkle trees. Like Cosmos, Polkadot has the disadvantage that existing blockchains cannot be integrated directly but only through bridges.

In [17], the authors present appXChain, an application-based solution for blockchain interoperability. A cross-chain hub enables cross-chain transactions using blockchain-specific APIs with an additional translation layer between the different blockchains. Verifiers on each blockchain offer light client services and additionally enable off-chain data access and communication. To showcase the approach, the authors adapt their approach to a use case in the healthcare industry. Here too, several trusted intermediaries are required.

In contrast to the approaches discussed above, our solution does not rely on multiple trusted intermediaries or requires an additional blockchain. Nevertheless, it provides the necessary abstractions and enables the execution of asynchronous cross-blockchain smart contract calls while also retaining a high degree of decentralization.

III. Background

Cross-blockchain communication refers to the ability to transfer arbitrary information between blockchains [6] and is, therefore, a prerequisite to achieving interaction between smart contracts running on different blockchains. Preferably, this information transfer is secure, decentralized, and trustless.

Blockchain relays are a technique to transfer state information between different blockchains [18]. A blockchain relay is a smart contract on the target blockchain which can receive block headers and verify them according to the source blockchain's consensus mechanism to replicate a chain of block headers. Instead of trusted intermediaries, blockchain relays depend on off-chain clients. These off-chain clients continuously relay block headers from the source blockchain to the target blockchain. A smart contract on the target

blockchain can use the blockchain relay's light client functionality to verify state information about the source blockchain.

Blockchain relays make use of a technique called Simplified Payment Verification (SPV) [1] to verify that a transaction is part of a specific block. Transactions in Bitcoin or Ethereum are stored using Merkle trees, whereby the block header includes the Merkle root [19]. The smart contract on the target blockchain can verify submitted Merkle proofs by recalculating the root of the Merkle tree and verifying that it matches the root hash stored in the block header. Further, for blockchains with probabilistic finality, the relay contract checks that the block is considered final. For Bitcoin and Ethereum, this requires that the block is part of the main chain and confirmed by at least six, respectively twelve blocks.

The advantage of relays is that they do not need any trusted intermediaries, as the smart contract validates the block headers directly on-chain. Nevertheless, blocks need to be forwarded continuously to keep the blockchain relay up-to-date, and the on-chain block header validation is quite costly. However, novel relay schemes already offer more-cost efficient solutions [20], [21]. Therefore, we use blockchain relays to implement cross-blockchain smart contract calls.

IV. Framework

In this section, we present a framework for asynchronous cross-blockchain smart contract calls. First, we provide a high-level overview of the framework and then give a detailed description of the functionality.

A. Overview

We propose a framework that allows smart contracts residing on two different blockchains to call each other. The framework (see Figure 1) is inspired by the RPC protocol, which enables client-server communication in computer networks. Similar to RPC, where a local computer calls a procedure on a remote computer, a smart contract on a source blockchain calls a function of a smart contract on a target blockchain. On both blockchains, stub contracts must be deployed. The stubs completely hide the complexity of executing cross-blockchain calls from the clients. They provide the respective interface to the called smart contract, encode/decode the sent messages, and verify the state of the blockchain from which the transmitted messages originated.

Since blockchains cannot communicate directly with each other, the framework needs a mechanism for cross-blockchain communication to enable the transmission of the messages (see Section IV-B). The framework relies on blockchain relays, through which it can preserve a high degree of decentralization. Relayers continuously transfer block headers from the source blockchain to the target blockchain and vice versa. These block headers contain the necessary information to verify the exchanged messages between the stubs.

A cross-blockchain smart contract call always requires actions by an external user. This external user is usually the account's owner that sent the transaction that resulted in the initialization of a cross-blockchain call and is therefore responsible for its further execution. It also has to prepare, execute and finally acknowledge the call. However, any external user can continue other cross-blockchain calls not initiated by them. Therefore, to guarantee that external users will perform this task, the framework applies an incentive mechanism (see Section IV-C) to ensure that calls are not orphaned.

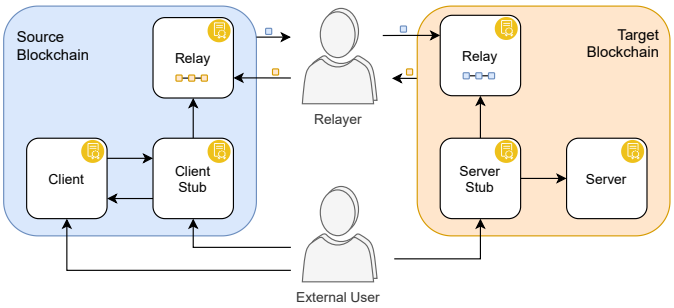

Fig. 1. High-level overview of the framework

B. Information Transfer

An essential part of the framework is the information transfer between stubs. A transaction in Ethereum consists of a type, the nonce, the gas price, the gas limit, the recipient, the value, and the signature composed of the r and s values. Specific transaction types may contain additional fields but are not relevant to this approach. Further, each transaction has a corresponding transaction receipt containing the type of the transaction, the status code indicating if the transaction failed, the cumulative gas used, all logs generated during the execution of the transaction, and a Bloom filter for the logs.

The framework uses two important mechanisms for information transfer: Ethereum events and blockchain relays. Smart contracts can emit Ethereum events to store log data on the blockchain. The log data consists of multiple (max. 4) topics and additional data. The topics include an identifier and data to describe the event, while the additional data includes the event parameters. Since the transaction receipt includes all logs generated during the execution of a transaction, stubs can emit events to log data on the blockchain, which can be verified on another blockchain by using blockchain relays.

Because the framework uses blockchain relays, it has the advantage that it does not depend on multiple trusted intermediaries to work. Other solutions would require that a certain number of intermediaries attest to the correctness of the transferred information. A blockchain relay, on the other hand, only requires at least one honest intermediary while being fully decentralized and secure [20]. It is much easier to ensure that there is one honest intermediary than, e.g., the majority of multiple intermediaries like in [8].

External users provide the stubs with the necessary block header, transaction, and transaction receipt, along with the respective Merkle proofs for the execution and acknowledgment of a cross-blockchain call. The stubs then call the relay contracts to verify the validity of the provided information.

Initially, the relay contract verifies the main chain membership and if the block has enough confirmations. After that, it verifies the Merkle proofs for the transaction and the transaction receipt. If these checks are successful, the stubs can be sure that the information is correct.

Then, the stubs can extract the call and reply message from the transaction and receipt data for the execution or acknowledgment of a cross-blockchain call. Ethereum uses Recursive Length Prefix (RLP)-encoding to serialize objects, which is why the stubs have to decode the transactions and the transaction receipts to retrieve the original data. The stubs also have to filter out the event with the correct identifier for the cross-blockchain call since a transaction can include multiple events. For that, the stubs can examine the first topic of an event. This topic stores the event's identifier, which is the hash of the event's signature. The stubs can then directly extract the respective event knowing the identifier.

C. Incentive Mechanism

External users can always deviate from the protocol for arbitrary reasons, i.e., delivering incomplete or fake data resulting in partially or not executed calls. The incomplete execution of a cross-blockchain call can lead to undesirable inconsistencies in the involved blockchains as not all necessary state changes are made. Therefore, to avoid such problems, the framework applies an incentive mechanism that ensures that external users fully execute and finalize already initiated cross-blockchain calls.

In the presented framework, a client stub requires that an external user deposits enough collateral before it can execute cross-blockchain calls. If it does not deposit enough collateral or no collateral, the transaction that results in a cross-blockchain call is reverted. The amount of collateral depends on how many calls the external user should be able to execute in parallel. The external user must deposit enough funds for each call to incentivize other external users to continue with the execution if it misbehaves. The collateral must be high enough such that it always covers the execution cost. Otherwise, there would be no incentive for other external users to continue with the execution, as the execution cost would be greater than the reward.

There are several ways to ensure that the originator placed enough collateral in the client stub. A naive approach is to set the collateral extraordinarily high, such that it is not possible to exceed it even when the transactions reach the block gas limit. However, every cross-blockchain call would require a large deposit, which is then no longer accessible to the external user. Furthermore, it poses a higher barrier to entry since only users who deposit the appropriate sum can execute cross-blockchain calls. On the other hand, the external user then has more to lose and is encouraged to make the call.

Another option is to calculate the amount of collateral needed to cover the expenses of the cross-blockchain call and reward. However, to compute the fees paid on the target blockchain, the client stub needs the gas price, consumed gas, and the current exchange rate between the blockchain's native currencies. For this, the client can use a liquidity pool on the source blockchain to get the current exchange rate, use the gas limit as it is the maximum amount of consumed gas, and get the average gas price from the relay contract. While this approach keeps the required collateral as low as possible, it is more complex and expensive.

During the initialization of a cross-blockchain call, the client stub locks the required collateral until the acknowledgment. Every call specifies a timeout to determine when other external users can acknowledge the call to retrieve the collateral as a reward. Should an external user acknowledge the call before reaching the timeout, the collateral is unlocked and still belongs to its original owner.

D. Cross-Blockchain Smart Contract Calls

The execution of a cross-blockchain smart contract call consists of four phases (see Figure 2): the *Initiate*, *Prepare*, *Execute*, and *Acknowledge* phases. An external user submits a single transaction in each phase, resulting in four transactions required to execute a cross-blockchain call. In the following, we explain the different phases in more detail:

a) Initiate: The originator of a cross-blockchain smart contract call is always an external user, which calls a client contract deployed on a source blockchain (Step 1) that uses a client stub to call a smart contract on a target blockchain (Step 2). The client stub offers the client contract an asynchronous version of the server contract's interface. Thus, the caller has to provide an additional parameter that specifies the options for the cross-blockchain call besides the parameters for the function call. In these options, the client specifies the (i) gas limit, (ii) timeout, (iii) callback contract, (iv) callback function, and (v) callback gas limit. The gas limit specifies the maximum amount of gas allowed for the call on the target blockchain, and the timeout specifies how long the call may last. Further, the callback options define how the call should be continued on the source blockchain, i.e., which contract to call, what function of this contract, and the maximum amount of gas allowed to execute this function.

After that, the client stub creates the call data for the smart contract call on the target blockchain. In Ethereum, a smart contract has access to special variables (e.g., the sender of the message), which is no longer the case with a cross-blockchain call. These special variables are, among other things, necessary when the server contract has to authorize the caller. The server stub provides the same functions as the server contract, whereby the function signature also includes the origin and the sender. Therefore, these variables are passed as additional function parameters included in the call data and accessible by the server stub on the target blockchain.

Then, the client stub continues with the initialization of the cross-blockchain call. During the initialization, the client stub checks that the external user has placed enough collateral to execute a cross-blockchain call and assigns the call a unique identifier. The client stub also stores the metadata for each cross-blockchain call. The metadata includes the following information: *id*, *origin*, *gas*, *callData*, *callback*, *callbackTo*,

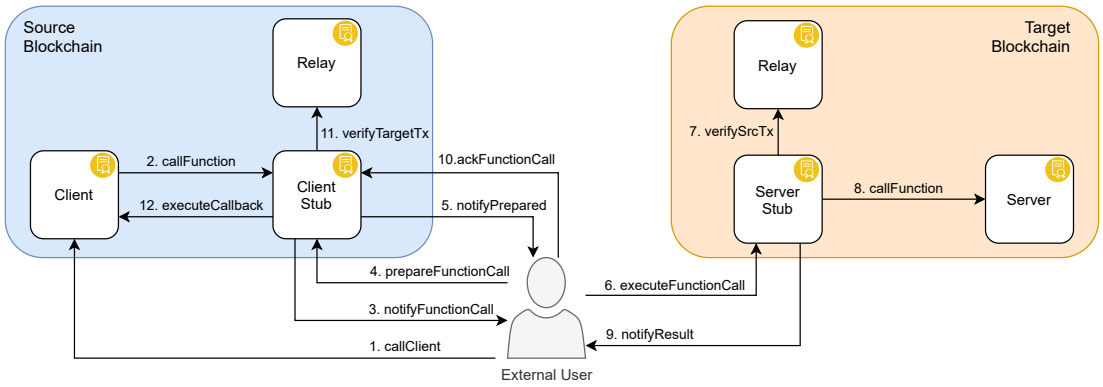

Fig. 2. Sequence of operations executed during a cross-blockchain smart contract call

callBackGas, and *timeout*. Since the direct storage of the metadata in the client stub would result in considerable costs, the client stub only hashes the data and stores the hash. Finally, the client stub emits an event to notify external users (Step 3) about the initialization of a new cross-blockchain call. Here the event must contain the necessary metadata, as the users have to provide the metadata in the next phases, where the hash is then recomputed and checked.

b) Prepare: After the initialization of the cross-blockchain call, an external user has to prepare it. This phase has the sole purpose of creating a transaction that addresses the client stub directly because a transaction does not include information about the entire call chain but only the address of the first contract. As a result, the transaction created in the *Initiate* phase does not include the address of the client stub. However, the server stub needs to verify that the client stub is the call message's creator, which is only possible if the transaction data includes the address of the client stub. Otherwise, it would be possible to create another client stub that sends invalid requests to the server stub.

An external user calls the client stub (Step 4) providing the call's metadata to prepare the cross-blockchain call. The client stub then hashes the data and verifies that it matches the stored hash. After that, the client stub emits an event to include the call id, the gas limit, and the calldata in the transaction receipt and notify external users (Step 5). Therefore, the transaction and transaction receipt of the *Prepare* phase include the necessary information to extract the call message on the target blockchain. The call message contains the call id, gas limit, sender, call data, and status. The call id uniquely identifies a cross-blockchain call, and the gas limit specifies the maximum amount of gas allowed for the call on the target blockchain. The sender specifies the origin of the cross-blockchain call, i.e., the address of the client stub, and the calldata represents the function, including its parameters to call the server contract. Finally, the status provides information on whether the *Prepare* transaction was successful.

c) Execute: Before the execution phase can begin, the external user must wait until the *Prepare* transaction, which includes the call message stored by the client stub, is included and confirmed in the relay on the target blockchain. How long this takes depends on the replicated blockchain and its consensus mechanism. During this time, the external user generates the Merkle proofs for the transaction and the receipt submitted in the *Prepare* phase.

As soon as the transaction is included, the user calls the server stub (Step 6), providing the transaction, transaction receipt, Merkle proofs, and the corresponding block header. The first task of the server stub is to decode the provided transaction and receipt and extract the call message from the client stub. After that, the server stub has to perform multiple checks before executing the smart contract call. Initially, the server stub checks that the receiver of the transaction is the client stub. It also verifies the successful execution of the transaction on the source blockchain by examining the status of the receipt. Further, to avoid replay attacks, the server stub stores and checks the identifier of each call.

After that, the server stub uses the blockchain relay (Step 7) to verify the transaction and the receipt. Before the server executes the call, it also checks whether there is enough gas left, as specified in the message by the client stub. Finally, if all checks are successful, the server stub executes the call and returns the result (Step 8) by emitting an event. This event includes the call id, a flag indicating the status of the call, and the actual result of the call. Similar to the call message, the transaction and transaction receipt include the information for the client stub to extract the reply message. The reply message consists of the call id, the flag indicating the status of the call, the result from the server contract, the status of the *Execute* transaction, and the address of the server stub. Therefore, both the *Prepare* and *Execute* transactions carry the information to exchange messages between the client and the server stub.

d) Acknowledge: Once the server stub has executed the cross-block-chain call, the external user waits for the *Execute* transaction, encompassing the reply message, to be included and confirmed by the blockchain relay on the source blockchain. Then, it calls the client stub (Step 10) and provides the transaction, transaction receipt, Merkle proofs, the block header, and the metadata of the call.

After decoding the transaction and the receipt to retrieve the

reply message, the client stub checks if the cross-blockchain call exists by checking the metadata. Like the server stub, it also checks the status of the transaction receipt and if the call timed out. In the event of a timeout, the client stub transfers the collateral to the external user acknowledging the call. Otherwise, the collateral is unlocked, and the owner can withdraw it. The client stub also uses the blockchain relay to verify the transaction and the receipt (Step 11). The proxy also checks whether there is still enough gas left for the callback function. After that, it continues with the execution of the callback function (Step 12) and deletes the cross-blockchain call so that it cannot be acknowledged multiple times.

V. Evaluation

After describing our framework, we analyze the costs and the execution time of cross-blockchain smart contract calls. For this purpose, we provide a prototypical implementation of the framework, which is available as an open-source project on GitHub[1].

A. Setup

For the evaluation, we perform cross-blockchain smart contract calls between Ethereum and the BNB Smart Chain (BSC). There are several reasons for this technical setup. First, both of these platforms are among the largest and most widely-used smart contract platforms. Second, they apply different consensus mechanisms and have distinct block times. While Ethereum uses Proof of Work (PoW) (recently Proof of Stake (PoS)) with a block time of approximately 15 seconds, BSC applies Proof of Staked Authority (PoSA) with a block time of roughly 5 seconds. Therefore, we can also examine whether this affects the presented approach.

Consequently, we need to operate relay solutions for both blockchains. For Ethereum, we make use of ETH-Relay [20], which is a relay scheme for Ethereum-based blockchains that adopts a validation-on-demand pattern to reduce operating costs. We set ETH-Relay's verification fee and lock time to zero. The framework also supports Ethereum with PoS consensus by applying relay solutions such as Verilay [22].

For BSC, we implemented a relay solution that directly verifies the block headers on-chain by verifying the signature and checking that the current validator set includes the signer. For both relays, we run relayers, which continuously submit the new blocks to the corresponding relay contract.

We implemented a cross-blockchain caller listening for newly initialized cross-blockchain calls to execute them automatically. Furthermore, we implemented a general client and server stub, which can be used to create the stubs for specific smart contracts. We created the stubs for a simple storage contract that require external users to deposit fixed collateral and offers a function to save and update a number and another function to retrieve it. Additionally, we created a client contract, which only calls the storage contract through the client stub and offers the necessary callback functions to process the result. Even though this is a simple example, it contains the most important features, like reading from and writing to another blockchain.

We deploy the smart contracts on the respective public test networks, i.e., the Ropsten testnet and the BSC testnet. By that, we can test the approach under near real-world conditions on networks similar to the Ethereum and BSC main networks.

[1] https://github.com/soberm/ccsc

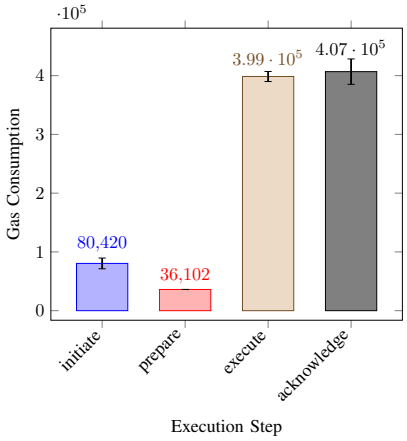

Fig. 3. Average gas consumption of the different execution steps

B. Cost Analysis

We analyze the costs by executing 110 cross-blockchain smart contract calls and examining the gas consumption of the four execution steps (see Figure 3). The costs for initializing and preparing a cross-blockchain smart contract call remain mostly constant for a specific function. These only change with the size of the call data. In our experiment, where we only had to pass two addresses and an integer for the server contract call, the initialization consumed on average 80,420 gas and the preparation 36,102 gas. These costs are relatively low, as hardly any data is stored on the blockchain, and only events are emitted. For larger payloads, the price is expected to rise in line with the gas costs for storing and hashing operations.

Most of the costs arise during the execute and acknowledge phase. The costs of the execute phase also include the costs of the server contract call. Therefore, the costs rise and fall accordingly. The same applies to the execution of the callback function in the acknowledge phase. Thus, it makes a big difference in which smart contract the client calls and how the client proceeds with the execution. In our example, these costs represent only a small part, as there is almost no logic behind the client and server contracts.

The other tasks, however, consume a considerable amount of gas. These tasks include, among other things, the verification by the relay contract and the decoding of the transaction and receipt. The gas consumption increases with the size of the Merkle proofs and the number of blocks that follow the corresponding block since the relays also have to verify main chain membership. Our results show that the execution of the call on the target blockchain consumes on average 398,520 gas, and the acknowledgment on the source blockchain consumes

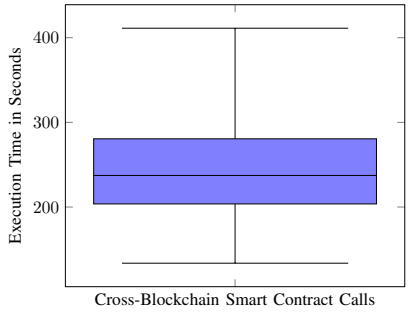

Fig. 4. Execution time of cross-blockchain smart contract calls

406,771 gas. Adding up all the costs, we get an average gas consumption of 921,815 gas per cross-blockchain smart contract call with a standard deviation of 28,615 gas.

In addition to the costs that arise from executing the cross-blockchain smart contract calls, there are also indirect costs from operating the blockchain relays. The operation of blockchain relays causes substantial costs since the relayers have to submit blocks continuously. Therefore, the relays charge a verification fee to incentivize relayers which varies for different relays.

C. Execution Time Analysis

This section analyzes the execution time of cross-blockchain smart contract calls. The execution time of a smart contract call depends on the execution of a single transaction. A cross-blockchain smart contract call, however, consists of several transactions. How quickly the individual transactions are executed depends to a large extent on the current network status and the selected gas price. To take this into account, we conducted our experiments on different days at different times. We also estimate the gas price automatically to ensure that miners include the transactions in the following blocks with a high probability.

Further, the execution time depends on the blockchain relays to include and confirm the necessary blocks. For our experiments, the off-chain clients continuously update the blockchain relays. We consider a block to be confirmed in Ethereum only if it is followed by at least six other blocks, with twelve being the optimum. For BSC, we require that $2/3 \times n + 1$ out of n validators from the current validator set have sealed the block by building new blocks upon it. We also assume that external users execute cross-blockchain calls at the earliest possible point in time, i.e., the external user continues with the execution as soon as the respective transactions are included and confirmed in the relays.

To analyze the execution time, we also measure the elapsed time of the executed cross-blockchain smart contract calls between their initialization and acknowledgment. The results (see Figure 4) show that the average execution time is 242 seconds with a standard deviation of 55 seconds. A considerable part of this is the waiting time until the corresponding blocks are included in the relays and confirmed. The waiting time averages 170 seconds with a standard deviation of 46 seconds. Thus, the execution of cross-blockchain smart contract calls would be much faster with blockchains that provide immediate block finality. As we removed the lock time of ETH-Relay for the sake of simplicity, this would need to be added on top, increasing the waiting time substantially as the original implementation foresees a lock time of 5 minutes.

VI. Discussion

To take a closer look at some important aspects of the presented approach, we use the following section to discuss its security, potential improvements, or extensions.

A. Security

The security of the presented approach depends primarily on the involved blockchains and the applied relay schemes. While the execution of a smart contract call is only dependent on the security of its hosting blockchain, the security of a cross-blockchain smart contract call depends on the security of all involved blockchains. This dependency is especially an issue when a cross-blockchain smart contract call is executed between blockchains that do not have the same security level. The blockchain with a weaker security level represents an additional attack vector. An attacker could, e.g., easier revert transactions executed during a cross-blockchain smart contract call on the blockchain with weaker security than on the other blockchain. If both blockchains have approximately the same security level, it is equally difficult to attack both blockchains, but overall they offer a broader field of attack. Therefore, this is an important factor to be considered.

The applied relay scheme profoundly influences security as it enables cross-blockchain communication. The blockchain relays ensure the authenticity and integrity of the transferred information, such that attackers cannot tamper with or provide wrong information. Should a malicious actor still manage to include invalid blocks in the relay, it could also manipulate cross-blockchain calls. Each blockchain relay can follow a different approach and therefore has other security risks.

Furthermore, external users making cross-blockchain smart contract calls have limited possibilities of misbehaving. The placed collateral ensures that users are always required to complete a cross-chain call. However, this only works if the execution cost does not exceed the amount of the placed collateral. Without a reward, other users would have no incentive to finish the execution of calls not initiated by them. Additionally, users have to provide Merkle proofs, and the stubs call the relay contracts to verify that the information from the other blockchain is correct. Hence, users also cannot manipulate the execution of cross-blockchain calls unless they can sabotage the blockchain relays or the originating blockchains.

B. Extensions

While the framework already provides the necessary functionalities for cross-blockchain smart contract calls, it still leaves some room for improvements and extensions. Currently, the smart contracts' stubs have to be manually implemented by the developers. With traditional RPC, on the other hand,

developers can usually generate the stubs automatically. For that purpose, the developers define the interface with an Interface Definition Language (IDL). Likewise, the Application Binary Interface (ABI) of the server contract could be used to generate the corresponding stubs to support developers in the creation of cross-blockchain applications.

Another issue is value transfer between the different blockchains. A smart contract can also send and receive the blockchain's native currency. The values of the native currencies are different for each blockchain. Therefore, the transferred amount would have to be adjusted using the current exchange rate. However, the receiving smart contract cannot query the exchange rate without an oracle and consequently cannot verify whether the exchange rate is correct. Another way to transfer value without this problem would be to use a cross-blockchain token.

Further, the execution of nested calls may require the possibility of pausing the execution of the cross-blockchain smart contract call until the result of the nested call is ready. This waiting operation is necessary if the result depends on the result of the nested call. Otherwise, the server can return the result immediately. For this purpose, additional events could be used, which indicate to wait and which result to return.

While Ethereum-based blockchains already provide a broad field of application, it is nonetheless desirable to make the framework applicable to different blockchain protocols. A prerequisite that these blockchains have to provide is basic smart contract capabilities. In Ethereum, a smart contract can emit events that are recorded in the blockchain. These events provide the means to communicate information to other blockchains since they are part of the transaction receipt. To support another blockchain, it also must provide a way to include additional information in the transaction data. A blockchain relay or another light client solution is also needed to verify the information on the other blockchains.

VII. Conclusion

Cross-blockchain smart contract calls represent an essential mechanism that enables the concept of cross-blockchain applications. However, already existing solutions usually require several trusted intermediaries.

Therefore, to improve blockchain interoperability, we introduced an RPC-inspired framework for asynchronous cross-blockchain smart contract calls using blockchain relays. We created a prototypical implementation of the framework for two popular EVM-based blockchains and evaluated the described approach concerning cost and execution time. Our experiments show that cross-blockchain calls cause a considerable overhead compared to regular smart contract calls. In future work, we will investigate the support of other blockchains and the extension of the framework with the automatic generation of the stub contracts, value transfers, and the ability to wait for calls, thus allowing nested calls.

Acknowledgment

The financial support from the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, as well as the Christian Doppler Research Association, is gratefully acknowledged.

References

[1] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. Accessed 2021-11-26. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] T. Guerpinar, G. Guadiana, P. Asterios Ioannidis, N. Straub, and M. Henke, "The current state of blockchain applications in supply chain management," in *2021 The 3rd International Conference on Blockchain Technology*. Association for Computing Machinery, 2021, p. 168–175.

[4] L. Lao, Z. Li, S. Hou, B. Xiao, S. Guo, and Y. Yang, "A survey of iot applications in blockchain systems: Architecture, consensus, and traffic modeling," *ACM Comput. Surv.*, 2020.

[5] E. J. De Aguiar, B. S. Faiçal, B. Krishnamachari, and J. Ueyama, "A survey of blockchain-based strategies for healthcare," *ACM Computing Surveys*, vol. 53, no. 2, 2020.

[6] S. Schulte, M. Sigwart, P. Frauenthaler, and M. Borkowski, "Towards blockchain interoperability," in *International Conference on Business Process Management*. Springer, 2019, pp. 3–10.

[7] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[8] M. Nissl, E. Sallinger, S. Schulte, and M. Borkowski, "Towards cross-blockchain smart contracts," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2021, pp. 85–94.

[9] P. Robinson and R. Ramesh, "General purpose atomic crosschain transactions," in *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2021, pp. 61–68.

[10] P. Robinson, "Performance overhead of atomic crosschain transactions," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

[11] M. Westerkamp and A. Küpper, "Smartsync: Cross-blockchain smart contract interaction and synchronization," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022.

[12] Clearmatics. General interoperability framework for trust-less cross-system interaction. Accessed 21 April 2021. [Online]. Available: https://github.com/clearmatics/ion

[13] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2019, p. 549–566.

[14] J. Kwon and E. Buchman. (2020) Cosmos whitepaper: A network of distributed ledgers. Accessed 2021-06-29. [Online]. Available: https://v1.cosmos.network/resources/whitepaper

[15] C. Goes, "The interblockchain communication protocol: An overview," *arXiv preprint arXiv:2006.15918*, 2020.

[16] G. Wood. (2016) Polkadot: Vision for a heterogeneous multi-chain framework. Accessed 2021-06-29. [Online]. Available: https://polkadot.network/PolkaDotPaper.pdf

[17] M. Madine, K. Salah, R. Jayaraman, Y. Al-Hammadi, J. Arshad, and I. Yaqoob, "appxchain: Application-level interoperability for blockchain networks," *IEEE Access*, vol. 9, pp. 87 777–87 791, 2021.

[18] V. Buterin, "Chain interoperability," *R3 Research Paper*, 2016.

[19] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[20] P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober, and S. Schulte, "ETH relay: A cost-efficient relay for ethereum-based blockchains," in *2020 IEEE International Conference on Blockchain*. IEEE, 2020, pp. 204–213.

[21] M. Westerkamp and J. Eberhardt, "zkRelay: Facilitating Sidechains using zkSNARK-based Chain-Relays," in *2020 IEEE European Symposium on Security and Privacy Workshops*. IEEE, 2020, pp. 378–386.

[22] M. Westerkamp and M. Diez, "Verilay: A verifiable proof of stake chain relay," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022, pp. 1–9.

11

Decentralized cross-blockchain asset transfers with transfer confirmation

Michael Sober, Marten Sigwart, Philipp Frauenthaler, Christof Spanring, Max Kobelt, and Stefan Schulte: *Decentralized cross-blockchain asset transfers with transfer confirmation*. In *Cluster Computing*, volume 26 (4), 2023

DOI: 10.1007/S10586-022-03737-6

Keywords: *Blockchain interoperability, Decentralized asset transfers, Cross-blockchain communication, Digital assets*

Abstract. Today, several solutions for cross-blockchain asset transfers exist. However, these solutions are either tailored to specific assets or neglect finality guarantees that prevent assets from getting lost in transit. In this paper, we present a cross-blockchain asset transfer protocol that supports arbitrary assets, is adaptable to different means of cross-blockchain communication, and adheres to requirements such as finality. The ability to freely transfer assets between blockchains may increase transaction throughput and provide developers with more flexibility by allowing them to design digital assets that leverage the capacities and capabilities of multiple blockchains. We define the general requirements and specifications for a cross-blockchain asset transfer protocol and provide a proof-of-concept implementation for EVM-based blockchains. Further, we evaluate the protocol concerning costs, transfer duration, and security.

Cluster Computing (2023) 26:2129–2146
https://doi.org/10.1007/s10586-022-03737-6

Decentralized cross-blockchain asset transfers with transfer confirmation

Michael Sober[1,4] · Marten Sigwart[2] · Philipp Frauenthaler[2] · Christof Spanring[3] · Max Kobelt[1,4] · Stefan Schulte[1,4]

Received: 1 April 2022 / Revised: 8 August 2022 / Accepted: 25 August 2022 / Published online: 15 September 2022

Abstract
Today, several solutions for cross-blockchain asset transfers exist. However, these solutions are either tailored to specific assets or neglect finality guarantees that prevent assets from getting lost in transit. In this paper, we present a cross-blockchain asset transfer protocol that supports arbitrary assets, is adaptable to different means of cross-blockchain communication, and adheres to requirements such as finality. The ability to freely transfer assets between blockchains may increase transaction throughput and provide developers with more flexibility by allowing them to design digital assets that leverage the capacities and capabilities of multiple blockchains. We define the general requirements and specifications for a cross-blockchain asset transfer protocol and provide a proof-of-concept implementation for EVM-based blockchains. Further, we evaluate the protocol concerning costs, transfer duration, and security.

Keywords Blockchain interoperability · Decentralized asset transfers · Cross-blockchain communication · Digital assets

1 Introduction

With its ability to store data and perform computations in a decentralized and immutable manner, blockchain technology shows potential in application areas such as finance [1], supply chain management [2], healthcare [3], business process management [4], smart cities [5], the Internet of Things [6, 7] and others [8]. Multiple independent and unconnected blockchains have been developed [9] to address the diverse requirements of these areas. As it is unlikely that a single blockchain caters to the requirements of all different areas [10], there is a strong need for interoperability between distinct blockchains.

Especially in scenarios where assets, i.e., digital representations of value, are managed on-chain, the lack of interoperability leads to a vendor lock-in as assets cannot leave the blockchain platform on which they were issued. Users would have to continue using the blockchain they initially chose since switching to another blockchain would only be possible with considerable effort or not at all. This vendor lock-in exposes projects to significant risks such as limited scalability [11], the danger that the underlying blockchain sinks into insignificance, and the inability to

✉ Michael Sober
michael.sober@tuhh.de

Marten Sigwart
m.sigwart@dsg.tuwien.ac.at

Philipp Frauenthaler
p.frauenthaler@dsg.tuwien.ac.at

Christof Spanring
christof.spanring@bitpanda.com

Max Kobelt
max.kobelt@tuhh.de

Stefan Schulte
stefan.schulte@tuhh.de

[1] TU Hamburg, Hamburg, Germany

[2] TU Wien, Vienna, Austria

[3] Pantos GmbH, Vienna, Austria

[4] Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg, Germany

take advantage of new features offered by novel blockchains [9]. Of course, a centralized entity can be deployed to migrate assets from one blockchain to another. However, this contradicts the blockchain's original idea of decentralization [12].

The ability to transfer assets to arbitrary blockchains in a decentralized way would remove the need to commit to a particular blockchain. Instead, assets could be migrated to new blockchains offering novel functionality or better security at any time [9]. Another potential use case of cross-blockchain asset transfers arises in the context of sidechains [13, 14]. The idea is that an asset can be transferred to and processed on multiple "side" blockchains, thus reducing the workload of the original blockchain.

One way to exchange assets between independent blockchains is via atomic swaps [15]. With atomic swaps, multiple parties can atomically exchange assets, whereby no party ends up worse off if one of them misbehaves. However, atomic swaps do not constitute actual cross-blockchain asset transfers. The different assets are not leaving their respective blockchain but rather ownership of assets changes in an atomic fashion. Cross-blockchain asset transfers, on the contrary, achieve that an asset moves from one blockchain to the other, enabling users to hold different denominations of the same asset type on multiple blockchains.

While schemes for cross-blockchain asset transfers have been proposed before, most of these solutions are designed with specific assets in mind and neglect important requirements for cross-blockchain asset transfers, such as finality to prevent assets from getting lost in transit.

Thus, in this paper, we formally define a set of general requirements that need to be fulfilled by cross-blockchain asset transfers and then propose a protocol that complies with the defined requirements.

This paper extends our previous work [16] by (i) providing more background information about cross-blockchain communication, (ii) adding additional functionality to the protocol, (iii) conducting a more detailed evaluation of the extended protocol while using different cross-blockchain communication mechanisms, and (iv) updating the related work.

The contributions of this paper can be summed up as follows:

- We formally define functional requirements for cross-blockchain asset transfers.
- We define a protocol specification for enabling cross-blockchain asset transfers that are decentralized, secure, and not tailored to specific assets.
- We evaluate the protocol on public Ethereum test networks using a proof-of-concept implementation for blockchains based on the Ethereum Virtual Machine (EVM).

To this end, Sect. 2 provides important background information. In Sect. 3, we formally define the requirements and specify the protocol for cross-blockchain asset transfers. Section 4 evaluates the proposed protocol concerning costs, duration, security, and features using a proof-of-concept implementation. Section 5 provides an overview of the related work. Finally, Sect. 6 concludes the paper.

2 Background

This section introduces some notations and definitions necessary for describing the requirements and specifications of the proposed cross-blockchain asset transfer protocol. Further, we look into different means of cross-blockchain communication, which is an essential building block of the protocol proposed in Sect. 3.

2.1 Notations and definitions

As has already been mentioned in Sect. 1, cross-blockchain asset transfers ideally enable users to hold different denominations of the same asset on multiple blockchains simultaneously. By that, users are free to choose on which blockchain they want to keep their assets. An asset can be seen as anything holding some value with a corresponding representation on a blockchain.

Assets can generally be divided into fungible and non-fungible assets [17]. Fungibility implies that two entities of the same asset are interchangeable. Cryptocurrencies like Bitcoin or Ether are fungible assets. Another example of fungible assets is Ethereum tokens following the ERC20 standard [18]. In contrast, non-fungible assets are uniquely identifiable, i.e., one entity cannot be substituted by another entity. For instance, ERC721 tokens (e.g., Cryptokitties) are non-fungible assets [19].

One can further distinguish between native and user-defined assets [17]. Native assets are inherently part of a particular blockchain. One cannot exist without the other, e.g., the Bitcoin and Ether cryptocurrencies and the Bitcoin and Ethereum blockchains, respectively. On the other hand, certain blockchains allow the implementation of use case-specific assets with their own set of rules, e.g., the already mentioned ERC20 or ERC721 tokens. Contrary to native assets, these user-defined assets are not bound to specific blockchains. Instead, they are implemented using smart contracts and can thus be potentially deployed on any blockchain with the necessary scripting capabilities to express the asset's rules.

This work concentrates on user-defined assets since the goal is to provide an asset that allows users to hold different amounts of the same asset on multiple blockchains at the same time. We formally define an asset A as a set where the set's members represent the asset's smallest indivisible entities (asset entities). For instance, the smallest indivisible entity of a fungible asset like Bitcoin is a Satoshi (i.e.,

0.00000001 BTC). For a non-fungible asset like Cryptokitties, the smallest indivisible entity is a single "cryptokitty".

We define the set of blockchains between which asset transfers can take place by the finite set B (also referred to as the *cross-blockchain ecosystem*). Each blockchain $b \in B$ can host multiple smart contracts. Out of these smart contracts, one contract is responsible for managing asset A on b. We denote this particular smart contract as c_b for each $b \in B$.

We assume blockchains to roughly follow the model devised by Satoshi Nakatomo [1]: The state of the blockchain is updated through transactions that can be used to transfer the native asset of the blockchain, store arbitrary data, or trigger the execution of smart contracts. In the latter case, the transaction's payload contains parameters based on which the smart contracts may change their associated state. For instance, a transaction payload containing a sender, a recipient, and an amount could trigger a smart contract causing the transfer of some user-defined asset from the sender to the recipient.

We specify transactions as a tuple containing the elements of the payload, which serve as parameters for the invoked contract. In particular, we use $tx := \langle param_1, \ldots, param_n \rangle$ to denote a transaction tx with a payload containing n parameters. Further, we define the function $calledContract(tx)$ to return the address of the smart contract that was triggered by transaction tx. The execution of transactions may fail, for instance, if a user does not have enough funds for a transfer. For this, we define the function $isSuccessful(tx)$ to return *true* or *false* depending on whether the transaction has been executed successfully or not.

Every transaction is signed by some off-chain user $u \in U$ before being submitted to the blockchain. The function $submitter(tx)$ denotes the user that signed tx. Users can be the owners of a subset of asset A on each participating blockchain $b \in B$. Subsequently, the set $A_u^b \subseteq A$ defines the entities of asset A that are owned by a particular user $u \in U$ on blockchain b.

Finally, for two blockchains $src, dest \in B$, and two users $sender, recipient \in U$, we define a cross-blockchain asset transfer as the transfer of some $X \subseteq A$ from user *sender* on source blockchain *src* to user *recipient* on destination blockchain *dest*.

2.2 Cross-blockchain communication

Fundamental to the protocol proposed in Sect. 3 is the ability to communicate state information between blockchains. In particular, the protocol relies on the ability to verify the inclusion of transactions across blockchains, i.e., the destination blockchain *dest* must be able to verify that certain transactions are included in the source blockchain *src*. Ideally, this cross-blockchain communication is decentralized such that no trust in a centralized party is required to ensure the validity of the information.

2.2.1 Oracles

Oracles, or more generally, data on-chaining solutions [20], offer one way to realize cross-blockchain communication. An oracle acts as a bridge between a blockchain and external data sources. The task of the oracle is to retrieve the data from the external data source (e.g., another blockchain) and submit it to a smart contract. While different oracle solutions exist, one approach to verify the inclusion of transactions is voting-based oracles. Several concepts and solutions for voting-based oracles have already been proposed [21–25].

A voting-based oracle requires a special oracle contract on the destination blockchain. Clients can call the oracle contract to get state information about the source blockchain *src*, e.g., to check for the inclusion of a specific transaction. The oracle contract starts a voting period during which other users can post their votes ("yes" or "no"). The answer which reaches a previously defined threshold (e.g., the majority) wins. Users are encouraged to participate in the voting process through crypto-economic incentives. Unfortunately, the voting mechanism incurs high costs if the aggregation of the votes is done on-chain. However, there are already oracle solutions, e.g. [25], that use an off-chain aggregation mechanism and only submit the aggregated result to the oracle contract. The oracle contract then only has to verify the validity of the aggregated result, which is cheaper since it requires only a single transaction.

2.2.2 Blockchain relays

Another technique for realizing transaction inclusion verifications is using blockchain relays [26]. Blockchain relays operate by having a set of off-chain clients relay block headers (see Fig. 1) from a source blockchain to a destination blockchain, replicating the source blockchain within the destination blockchain. Each relayed block header is validated on the destination blockchain according to the validation rules of the source blockchain. As block headers do not contain any transactions, only a small fraction of the space needed to store full blocks is consumed on the destination blockchain.

With the block headers of the source blockchain replicated, clients can query state information about the source blockchain on the destination blockchain, e.g., the current longest branch of the source blockchain, or if a certain block header is confirmed by at least x succeeding block headers.

Further, transactions in a blockchain (i.e., in a block) are stored in a special data structure called Merkle tree [27], or variants thereof. Since the root hash of the Merkle tree is

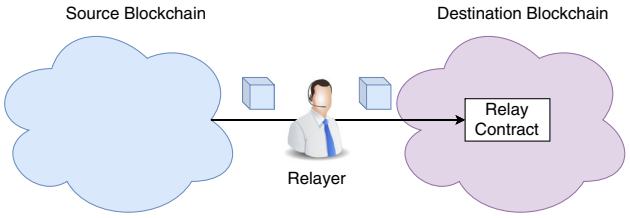

Fig. 1 Blockchain relay

also stored in the block header, it becomes possible on the destination blockchain to verify whether some transaction is included within a particular block of the source blockchain. For that, users construct a so-called Merkle proof of membership [27]. The proof is submitted to the destination blockchain, on which the proof is used to recalculate the root hash of the corresponding Merkle tree. If the calculated root hash matches the root hash of the stored block header, the destination blockchain can be certain that the transaction is included within the corresponding block of the source blockchain. This technique is also known as Simplified Payment Verification (SPV) [1].

Blockchain relays provide an on-chain answer to whether a certain transaction is included in the source blockchain. Contrary to voting-based oracles where multiple clients have to act honestly, blockchain relays only require one honest participating client [28]. However, blockchain relays cause higher costs since relayers have to continuously submit block headers to keep the relay up to date. In the following section, we use blockchain relays and oracles to realize the proposed asset transfer protocol.

3 Cross-blockchain asset transfers

In this section, we define the requirements for cross-blockchain asset transfers. Then, we use these requirements as the foundation to define a decentralized cross-blockchain asset transfer protocol.

3.1 Requirements

As defined in Sect. 2, a cross-blockchain asset transfer for an asset A constitutes the transfer of ownership of some subset $X \subseteq A$ from some user *sender* on a source blockchain *src* to another user *recipient* on a destination blockchain *dest*.

Before the transfer, X must only exist on blockchain *src*, and after the transfer, the asset must only exist on blockchain *dest*. At no point should X exist on both blockchains in parallel since the accidental duplication of asset entities can potentially lead to a deflation of the asset's value. Hence, a cross-blockchain asset transfer should only be successful, i.e., X is created on *dest*, if X has been burned (i.e., destroyed) before by its owner on *src*.

Therefore, before X can be recreated on *dest*, *dest* needs some kind of evidence that X has already been burned on *src*. If we assume that it is possible to provide such evidence guaranteeing that X has been burned on *src* and that this evidence can be used to recreate X on *dest*, two further requirements emerge. First, faking the evidence needs to be prevented at all costs. Users should not be able to counterfeit evidence certifying that X has been burned on *src* without it having occurred. Second, if the evidence is correct, it should only be used once to recreate X on a different blockchain, i.e., on blockchain *dest*. Hence, evidence of X having been burned on *src* cannot be used multiple times to recreate X on other blockchains. Essentially, disregard of any of these requirements would enable users to illegally create new entities of asset A out of nothing—again potentially deflating the value of the asset and decreasing trust in this particular asset.

A further requirement comes up when trying to prevent the opposite, accidental inflation of the asset's value. Accidental inflation could take place if X is burned on *src* without ever being recreated on *dest*. This reduces the total supply of A. Hence, cross-blockchain asset transfers need to be eventually finalized to not decrease the total supply of A over time. That is, either the transfer is executed completely or it fails with no intermediate state persisting.

Finally, the source blockchain *src* possibly needs to perform some action if a certain cross-blockchain asset transfer has been executed successfully (i.e., X has been successfully recreated on destination blockchain *dest*). For instance, imagine a business deal where some user *buyer* wants to buy asset entities $Y \subseteq A'$, with $A' \cap A = \emptyset$, from another user *seller*. While asset A' lives exclusively on the source blockchain *src*, user *buyer* aims to buy Y with cross-blockchain asset A, transferring X from themselves on blockchain *src* to user *seller* on blockchain *dest*. Therefore, ownership of Y can change on *src* if it is ensured that the transfer of X was successful, i.e., blockchain *src* must retrieve a confirmation of the successful transfer.

To sum up, we define the general requirements for a cross-blockchain asset transfer as follows:

Requirement 1 When a user *sender* wants to burn X on blockchain *src*, X should only be burned if $X \subseteq A^{src}_{sender}$.

Requirement 2 When transferring some $X \subseteq A$ from the source blockchain *src* to the destination blockchain *dest*, X should only be recreated on *dest* if it can be proven that X has already been burned on *src*. That is, it should not be possible to counterfeit the burning of asset entities.

Requirement 3 Double spending must be prevented at all times. That is, if X is burned on one blockchain, X can only be recreated once on one other blockchain.

Requirement 4 If X is burned on one blockchain, X is always recreated on another blockchain within a certain

time limit t. Further, finality should not be dependent on a single actor (i.e., not be centralized).

Requirement 5 After burning X on source blockchain *src*, *src* will eventually receive a confirmation that X has been successfully recreated on another blockchain. Analogous to decentralized finality, any user should be able to submit confirmations. This requirement is *optional* since not every use case requires that the source blockchain knows whether a cross-blockchain asset transfer has been completed successfully.

In the next subsection, we define a cross-blockchain asset transfer protocol that fulfills these requirements. To this end, we first define a base protocol that fulfills Requirements 1 to 4 (Sect. 3.2). We then provide an extension of the protocol to also account for Requirement 5 (Sect. 3.3).

3.2 Base protocol

Protocol 1 Protocol for cross-blockchain asset transfers

Goal: For two blockchains $src, dest \in B$ and two users $sender, recipient \in U$, transfer $X \subseteq A$ from *src* to *dest* and change ownership of X from *sender* to *recipient*.

1. **Burn.** User *sender* creates a new BURN transaction $tx_{\text{BURN}} := \langle recipient, dest, X \rangle$.
   (a) User *sender* signs and submits tx_{BURN} to source blockchain *src* invoking contract c_{src}, i.e., the contract managing asset A on *src*.
   (b) When being invoked, contract c_{src} performs the following operations.
      i. Verify $dest \in B$ to make sure that the specified blockchain *dest* is part of the cross-blockchain ecosystem.
      ii. Verify $X \subseteq A^{src}_{sender}$ to make sure that user *sender* owns the asset entities it wants to transfer on blockchain *src*.
      iii. When all checks are successful, the asset entities to be transferred are burned, i.e., $A^{src}_{sender} = A^{src}_{sender} \setminus X$.
2. **Claim.** Once tx_{BURN} is included in blockchain *src*, any user $u \in U$ can construct the CLAIM transaction $tx_{\text{CLAIM}} := \langle tx_{\text{BURN}}, proof_{tx_{\text{BURN}}} \rangle$. Variable $proof_{tx_{\text{BURN}}}$ contains the Merkle proof of membership of tx_{BURN} certifying the inclusion of tx_{BURN} in blockchain *src*.
   (a) User u signs and submits tx_{CLAIM} to blockchain $b \in B$ invoking contract c_b, i.e., the contract managing asset A on b.
   (b) When being invoked, contract c_b utilizes the verifier contract $c_{verifier}$ to verify the inclusion and confirmation of tx_{BURN} in blockchain *src*, i.e., c_b calls $c_{verifier}.verifyInclusion(tx_{\text{BURN}}, proof_{tx_{\text{BURN}}}, src)$.
   (c) If $c_{verifier}$ confirms the inclusion of tx_{BURN}, contract c_b performs the following steps.
      i. Verify $b = dest$ to ensure that the executing blockchain b is the intended destination blockchain *dest*. Note that *dest*, *recipient*, and X are available within c_b as these variables are contained within the payload of tx_{BURN}.
      ii. Verify $tx_{\text{BURN}} \notin T_{\text{BURN}}$ where T_{BURN} is the set of BURN transactions that have already been used to claim entities of asset A on *dest*. This ensures that BURN transactions cannot be used multiple times for claiming.
      iii. Verify $calledContract(tx_{\text{BURN}}) = c_{src}$ to make sure that the contract that has been invoked by tx_{BURN} is a contract authorized for managing asset A on blockchain *src*.
      iv. Verify that $isSuccessful(tx_{\text{BURN}})$ returns true to ensure that the execution of c_{src} has been completed without error.
      v. If $c_{verifier}.confirmations(tx_{\text{BURN}}, src) > t$, user *recipient* has not submitted tx_{CLAIM} within time t. Hence, the user $u = submitter(tx_{\text{CLAIM}})$ that submitted tx_{CLAIM} receives a transfer fee $X_{fee} \subseteq X$ as reward for finalizing the transfer, i.e., $A^{dest}_u = A^{dest}_u \cup X_{fee}$. Otherwise, no fee will be paid to u (i.e., $X_{fee} = \emptyset$), resulting in all asset entities being transferred to *recipient*.
      vi. (Re-)create the asset entities and assign ownership to user *recipient*, i.e., $A^{dest}_{recipient} = A^{dest}_{recipient} \cup (X \setminus X_{fee})$.
      vii. Add tx_{BURN} to the set of already used BURN transactions, i.e., $T_{\text{BURN}} = T_{\text{BURN}} \cup \{tx_{\text{BURN}}\}$.

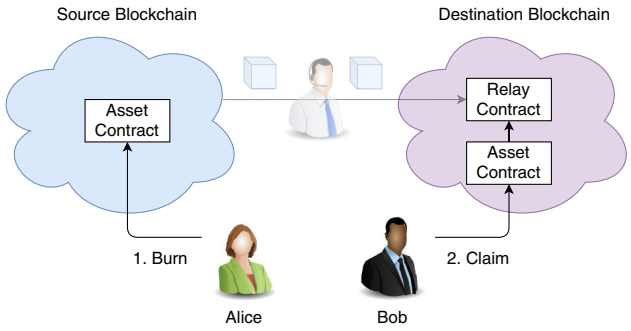

Fig. 2 Base protocol for cross-blockchain asset transfers leveraging a blockchain relay

As mentioned above, a cross-blockchain asset transfer should only be successful if the asset is first burned on the source blockchain and then recreated on the destination blockchain (Requirement 2). Therefore, the transfer requires at least two steps: one "burn" step on the source blockchain and one "claim" step on the destination blockchain. The protocol uses oracles or blockchain relays for decentralized cross-blockchain communication to verify the "burn" step. In particular, these provide an on-chain answer to whether a specific transaction is included in the source blockchain via SPV [1]. With this in mind, we can outline a minimal protocol for cross-blockchain asset transfers (see Fig. 2).

The protocol consists of a BURN transaction tx_{BURN} submitted to source blockchain *src* and a CLAIM transaction tx_{CLAIM} submitted to destination blockchain *dest*. Further, a verifier contract allows the asset contract on blockchain *dest* to verify the inclusion of BURN transactions in blockchain *src*. The exact specification is outlined in Protocol 1.

Initially, some user *sender* creates transaction tx_{BURN}. The payload of tx_{BURN} contains the user intended as the recipient of the transfer (*recipient*), an identifier representing the desired destination blockchain (*dest*), and the asset entities to be transferred (X). It may also be that additional data is required, as is the case with ERC721 tokens, which have a *tokenID* or *tokenURI*. These must also be part of tx_{BURN} or an additional data structure (e.g., the transaction receipt) that can be verified on the destination blockchain.

User *sender* then signs and submits tx_{BURN} to the source blockchain *src* invoking smart contract c_{src}, which manages asset A on blockchain *src* (Step 1a). The smart contract then verifies that the specified destination blockchain *dest* is part of the cross-blockchain ecosystem (Step 1(b)i). Second, the contract ensures that the user *sender* is the current owner of X on blockchain *src* (Step 1(b)ii). If both checks are successful, X is burned on *src* (Step 1(b)iii).

Once tx_{BURN} is included in blockchain *src*, any user $u \in U$ can construct the CLAIM transaction tx_{CLAIM}. The payload of tx_{CLAIM} consists of transaction tx_{BURN} and a Merkle proof of membership of tx_{BURN} ($proof_{tx_{\text{BURN}}}$). The verifier contract $c_{verifier}$ on blockchain *dest* uses the proof to verify the inclusion of tx_{BURN} in blockchain *src*. Note that if only the sender or only the recipient of the transfer were allowed to submit tx_{CLAIM}, the finality of the transfer (Requirement 4) would be entirely dependent on that particular user, e.g., the user could decide not to submit tx_{CLAIM}.

User u then signs and submits tx_{CLAIM} to some blockchain $b \in B$ invoking the contract c_b managing A on blockchain b (Step 2a).

By invoking the verifier contract $c_{verifier}$, the contract c_b checks whether tx_{BURN} is included and confirmed in blockchain *src* (Step 2b). If the verifier contract does not confirm the inclusion of tx_{BURN}, the claim request is rejected. Otherwise, contract c_b performs the following steps: First, it verifies that b is the intended destination blockchain *dest* (Step 2(c)i). Second, it is verified that tx_{BURN} has not been used to claim X on b before (Step 2(c)ii). Third, if both checks are successful, contract c_b verifies that the contract that burned X on *src* is a valid contract authorized for managing A on *src* (Step 2(c)iii). If this is the case, contract c_b further checks that tx_{BURN} was successful, i.e., the execution of contract c_{src} has been completed without any error, e.g., constraint violations (Step 2(c)iv). This check covers the case in which some blockchains transactions may be included even if the triggered smart contract execution was not successful. While Ethereum also includes failed transactions, other blockchains may not include such transactions at all.

The above checks ensure that tx_{CLAIM} is only successful if the corresponding tx_{BURN} was also executed successfully. To further account for Requirement 4 (transfer finality), the protocol must ensure that when transaction tx_{BURN} takes place on blockchain *src*, the corresponding tx_{CLAIM} is eventually submitted to destination blockchain *dest*.

Usually, the incentive for transfer finalization lies with the recipient of the transfer since the recipient wants to receive the transferred asset entities. However, in case the recipient is indisposed to submit tx_{CLAIM} for some reason, the protocol offers an incentive in the form of a transfer fee to other users. That is, any user u that successfully submits tx_{CLAIM} gets assigned a subset $X_{fee} \subseteq X$ as a reward (-Step 2(c)v). However, to provide the user *recipient* with the chance to receive all entities of X, other users are only eligible to receive the fee if they submit tx_{CLAIM} after a certain time t has elapsed.

Time t is defined by the number of blocks that succeed the block containing tx_{BURN} on source blockchain *src*. Hence, when being invoked by tx_{claim}, c_b additionally queries the verifier contract $c_{verifier}$ whether the block containing tx_{BURN} is confirmed by more than t succeeding blocks. If this is the case, the time t is considered elapsed, and the user that submitted tx_{CLAIM} receives the transfer fee

X_{fee}, while user *recipient* receives the rest $X \setminus X_{fee}$. If not, the user *recipient* always receives the entire set X (i.e., $X_{fee} = \emptyset$), even if another user submitted tx_{CLAIM} (Step 2(c)vi). Asset entities are (re-)created on blockchain b (= *dest*) by incrementing the balance of the recipient (in case of fungible assets) or by copying the transferred asset's data structure from tx_{BURN} into the storage of contract c_b (in case of non-fungible assets).

Finally, to ensure that tx_{BURN} cannot be used to claim X on b again, tx_{BURN} is added to the set of already used BURN transactions T_{BURN} (Step 2(c)vii).

3.3 Protocol extension for transfer confirmations

Protocol 2 Extension for transfer confirmations on the source blockchain

Goal: For two blockchains *src*, *dest* $\in B$ and two users *sender*, *recipient* $\in U$, transfer $X \subseteq A$ from *src* to *dest* and change ownership of X from *sender* to *recipient*. Confirm finalization of transfer on *src*.

1. **Burn.** Contrary to Protocol 1, tx_{BURN} contains an additional variable $Y \subseteq A$ representing the stake that will act as reward for users submitting a confirmation transaction to *src* once the transfer has been completed on *dest*.
   (a) User *sender* signs and submits tx_{BURN} to source blockchain *src* invoking contract c_{src}.
   (b) When being invoked, contract c_{src} performs the following steps.
      i. Perform all verification steps from Step 1b of Protocol 1.
      ii. Verify $Y \subseteq A^{src}_{sender}$ to make sure that user *sender* owns the asset entities she wants to use as stake on blockchain *src*.
      iii. Verify $Y \cap X = \emptyset$, i.e., X and Y are disjoint, to ensure that the asset entities intended to be transferred are not used as stake and vice versa.
      iv. When all checks are successful, the asset entities to be transferred are burned and the stake is locked, i.e., $A^{src}_{sender} = A^{src}_{sender} \setminus (X \cup Y)$.
2. **Claim.** To claim the entities of asset A on destination blockchain *dest*, the same steps as for tx_{CLAIM} in Protocol 1 are performed.
3. **Confirm.** Once tx_{CLAIM} is included in blockchain *dest*, any user $u \in U$ can construct the CONFIRM transaction $tx_{\text{CONFIRM}} := \langle tx_{\text{CLAIM}}, proof_{tx_{\text{CLAIM}}} \rangle$. Variable $proof_{tx_{\text{CLAIM}}}$ contains the Merkle proof of membership of tx_{CLAIM} certifying the inclusion of tx_{CLAIM} in blockchain *dest*.
   (a) User u signs and submits tx_{CONFIRM} to the source blockchain *src* invoking c_{src}.
   (b) When being invoked, contract c_{src} utilizes the verifier contract $c_{verifier}$ to verify the inclusion and confirmation of tx_{CLAIM} in blockchain *dest*, i.e., c_{src} calls $c_{verifier}.verifyInclusion(tx_{\text{CLAIM}}, proof_{tx_{\text{CLAIM}}}, dest)$.
   (c) If $c_{verifier}$ confirms the inclusion of tx_{CLAIM}, contract c_{src} performs the following steps.
      i. Verify $tx_{\text{CLAIM}} \notin T_{\text{CLAIM}}$ where T_{CLAIM} is the set of CLAIM transactions that have already been used for confirming transfer finalization on *src*. This ensures that CLAIM transactions cannot be used multiple times.
      ii. Verify $calledContract(tx_{\text{BURN}}) = c_{src}$ to ensure that tx_{CONFIRM} invokes the same contract that has been invoked by tx_{BURN}.
      iii. Verify $calledContract(tx_{\text{CLAIM}}) = c_{dest}$ to make sure that the contract that has been invoked by tx_{CLAIM} is the contract managing asset A on *dest* as intended by tx_{BURN}.
      iv. Verify that $isSuccessful(tx_{\text{CLAIM}})$ returns true to ensure that the execution of c_{dest} has been completed without any error.
      v. Check if timeout for submitting tx_{confirm} is reached. If so, user *sender* has not submitted tx_{CONFIRM} in time. Hence, the user $u = submitter(tx_{\text{CONFIRM}})$ that submitted tx_{CONFIRM} receives the locked stake from tx_{BURN} as reward for providing the confirmation of the corresponding claim, i.e., $A^{src}_u = A^{src}_u \cup Y$. If not, user *sender* gets back control of the locked stake, i.e., $A^{src}_{sender} = A^{src}_{sender} \cup Y$, regardless of which user submitted tx_{confirm}.
      vi. Add tx_{CLAIM} to the set of already used CLAIM transactions, i.e., $T_{\text{CLAIM}} = T_{\text{CLAIM}} \cup \{tx_{\text{CLAIM}}\}$.

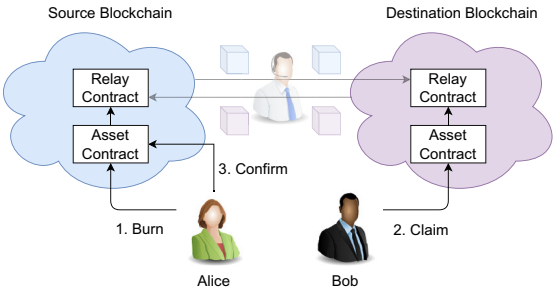

Fig. 3 Protocol extension to provide transfer confirmations leveraging blockchain relays

Protocol 1 provides finality by incentivizing users to submit tx_{CLAIM} for each tx_{BURN}. However, only the destination blockchain *dest* knows whether a cross-blockchain asset transfer has been completed successfully. Source blockchain *src* never actually learns about the finalization of the transfer. To circumvent this, we can augment Protocol 1 by adding a third kind of transaction (CONFIRM) to confirm the successful cross-blockchain asset transfer on *src*, as displayed in Fig. 3.

The extension of the protocol is specified in Protocol 2. At first, transaction tx_{BURN} is submitted to the source blockchain *src*. The payload of tx_{BURN} is the same as presented in Protocol 1—except for one additional field defined as Y. Y is a subset of asset A and represents a certain amount of stake that will act as a reward for users submitting a CONFIRM transaction to *src*.

When user *sender* submits tx_{BURN} to *src* (Step 1a), c_{src} first performs the same verifications as in Protocol 1 (Step 1(b)i). Additionally, a couple of checks concerning the stake Y are performed. First, it is verified that user *sender* is the owner of Y on blockchain *src* (Step 1(b)ii). Second, it is verified that the sets X and Y are disjoint (Step 1(b)iii). That is, the asset entities intended to be transferred should not be used as a stake and vice versa. If all verifications are successful, X is burned on *src*, and Y is locked (Step 1(b)iv).

Once tx_{BURN} is included in *src*, any user can create and submit transaction tx_{CLAIM} containing tx_{BURN} and a corresponding proof data $proof_{tx_{\text{BURN}}}$ to the destination blockchain *dest*. The steps performed when the corresponding smart contract c_{dest} is invoked are identical to those in Protocol 1 (Step 2).

With tx_{CLAIM} being successfully executed and included in blockchain *dest*, the successful transfer is reported back to source blockchain *src*. For that, any user can submit a CONFIRM transaction tx_{CONFIRM} to *src* (Step 3a).

When being invoked by tx_{CONFIRM}, contract c_{src} verifies that tx_{CLAIM} is included in *dest* using transaction inclusion verification (Step 3b). If the verifier contract does not confirm the inclusion of tx_{CLAIM}, the confirm request is rejected. Otherwise, contract c_{src} performs the following steps. First, it verifies that tx_{CLAIM} has not already been used for confirming transfer finalization (Step 3(c)i). Second, it checks that contract c_{src} is the contract that has burned X, i.e., whether c_{src} has been invoked by tx_{BURN} (Step 3(c)ii). This check ensures that contract c_{src} is the intended receiver of the confirmation. Third, c_{src} verifies that the contract invoked by tx_{CLAIM} is the correct contract managing asset A on the intended destination blockchain *dest* (Step 3(c)iii). *dest* can be extracted from tx_{BURN} included in the payload of the provided tx_{CLAIM}. Fourth, contract c_{src} checks that tx_{CLAIM} was successful, i.e., the execution of contract c_{dest} has been completed without any error (Step 3(c)vi).

If all checks are successful, contract c_{src} not only has the information about the successful execution of tx_{BURN} but also that the corresponding tx_{CLAIM} finalized the transfer on blockchain *dest*. To reward user u for submitting tx_{CONFIRM} to *src*, the locked stake Y from tx_{BURN} is assigned to u (Step 3(c)v). This provides an incentive for users to submit tx_{CONFIRM} to *src* for finalized cross-blockchain asset transfers.

The protocol uses a similar approach for the locked stake as for the transfer fee. The initiator of the transfer can withdraw its locked stake with certain conditions. A timeout determines who is eligible to retrieve the locked stake Y when tx_{CONFIRM} is submitted. Before the timeout, only *sender* is eligible for the stake. After the timeout, anyone submitting tx_{CONFIRM} earns Y as a reward. As contract c_{src} executes both the burning of entities and confirmations, it can track the time difference between the two events.

Finally, to ensure that tx_{CLAIM} cannot be used several times, tx_{CLAIM} is added to the set of already used CLAIM transactions T_{CLAIM} (Step 3(c)vi).

4 Evaluation

In this section, we evaluate the proposed cross-blockchain asset transfer protocol and its extension with regard to the defined requirements and conduct a quantitative analysis evaluating transfer costs and duration. For the evaluation, we provide a proof-of-concept implementation for EVM-based blockchains such as Ethereum and Ethereum Classic. The prototype, as well as the evaluation scripts used for obtaining the results presented in Sect. 4.3, are available as an open-source project on GitHub[1].

[1] https://github.com/soberm/x-chain-protocols/

4.1 Prototype

As mentioned above, the prototype is implemented for EVM-based blockchains. The advantages of targeting EVM-based blockchains in a first proof-of-concept are twofold. First, EVM-based blockchains such as Ethereum are among the most popular blockchains concerning decentralized applications (DApps) and digital assets [11, 29]. Cross-blockchain transfer capabilities for EVM-based blockchains can thus enhance the utility of a majority of available assets. The second reason is rather practical. As quite a few EVM-based blockchains exist, multiple blockchains can be targeted with a single implementation. However, as long as a blockchain provides sufficient scripting capabilities to implement the concepts of the protocol, as well as some means of transaction inclusion verification (e.g., via oracles or relays), the solution can be adopted beyond EVM-based blockchains.

For our analysis, we use the ERC20 token standard as asset representation. For transaction inclusion verification, we use two different means of cross-blockchain communication. Initially, we implemented the prototype to work with ETH Relay, a blockchain relay specifically targeting EVM-based blockchains [28]. ETH Relay follows a validation-on-demand pattern where relayers can issue a dispute during a certain lock time to trigger the validation of a block header. This approach saves costs since not every block has to be validated. For simplicity, we set the lock time to 0 as it would only lengthen the experiments and add a constant value to the transfer duration.

Additionally, we apply the oracle solution proposed in [25], which uses an off-chain aggregation mechanism based on threshold signatures. We extended the oracle to return block headers instead of single transactions such that it implements the same interface as ETH Relay. We run three oracle nodes, the majority of which have to agree on the same outcome to construct the threshold signature. With the application of both mechanisms, we can also observe how the asset transfer protocol works with different approaches to cross-blockchain communication.

A transaction in Ethereum consists of the fields *nonce*, *gasPrice*, *gasLimit*, *to*, *value*, *data*, and a signature (v, r, s) [30]. The field *data* contains the payload (e.g., the parameters for a smart contract invocation) of the transaction. The field *to* contains the address of the smart contract that has been invoked by the transaction (i.e., function $calledContract(tx)$ in our protocol). The submitter $submitter(tx)$ of the transaction can be calculated out of the signature fields v, r, and s.

It should be noted that the transaction data does not contain information about the transaction status, i.e., whether the execution succeeded or failed. In Ethereum, this information is stored in another data structure, the so-called transaction receipt. For each transaction, there exists a corresponding receipt. The receipt contains all events that were emitted during the execution of the transaction. Further, a status flag indicates the successful execution of the transaction. Thus, when evaluating the function $isSuccessful(tx)$ in our protocol, the asset contract must have access to the receipt of tx as well.

In Ethereum, a block's transactions and receipts are kept in separate Merkle trees, which do not contain references to each other. However, transaction and receipt are logically linked together as their position in their respective trees is identical. Both the inclusion of the transaction and the receipt can be verified via Merkle proofs of membership using the respective Merkle trees.

Both Merkle proofs must be evaluated along the same path to ensure that a transaction and receipt belong together. Hence, in our prototype, the proof data certifying the inclusion of a transaction (e.g., $proof_{tx_{\text{BURN}}}$) contains a Merkle proof for the transaction and the transaction receipt. Further, it includes the path for the evaluation of the Merkle proofs.

4.2 Requirements analysis

This section evaluates the protocol with regard to the requirements defined in Sect. 3.1.

4.2.1 Requirement 1 (ownership)

When user *sender* submits a BURN transaction tx_{BURN} invoking contract c_{src}, the contract verifies that $X \subseteq A^{src}_{sender}$ (see Step 1(b)ii of Protocol 1), thus making sure that user *sender* is the owner of X on *src*. Hence, we consider Requirement 1 as fulfilled.

4.2.2 Requirement 2 (no claim without burn)

To claim X on *dest*, a user u submits a CLAIM transaction tx_{CLAIM}. As defined by the protocol, the user provides tx_{BURN} as well as some proof data in the payload of tx_{CLAIM}. Before recreating X, the asset contract c_{dest} on blockchain *dest* performs several checks.

First, it is verified that tx_{BURN} is included in the source blockchain *src* and confirmed by enough blocks (Step 2b). Second, the protocol checks that tx_{BURN} indeed invoked asset contract c_{src} on the source blockchain *src* (Step 2(c)iii). Third, contract c_{dest} verifies that the execution of tx_{BURN} was successful (Step 2(c)iv).

These three checks ensure that assets are created on the destination blockchain *dest* if they have been successfully burned by the contract c_{src} on the source blockchain *src*.

Notably, the fulfillment of this requirement strongly depends on the security of the used transaction inclusion verification mechanism. A security analysis of the blockchain relay (ETH Relay), as well as the oracle used in our proof-of-concept implementation, can be found in [25, 28], respectively.

4.2.3 Requirement 3 (double spend prevention)

To ensure that burned assets can not be claimed multiple times, all BURN transactions that have already been used within CLAIM transactions are stored in a set T_{BURN} within asset contract c_{dest}. When c_{dest} is invoked by a new CLAIM transaction tx_{CLAIM}, it only executes the claim if the provided BURN transaction tx_{BURN} is not yet included in T_{BURN} (Step 2(c)ii).

Further, by encoding an identifier of the desired destination blockchain *dest* within BURN transactions, a burned asset can not be claimed on multiple different blockchains. When an asset contract c_b on some blockchain b is invoked by a CLAIM transaction containing tx_{BURN}, c_b can verify whether it is the intended destination contract by comparing $b = dest$ (Step 2(c)i). If not, the claim is rejected. Therefore, Requirement 3 can be considered fulfilled as well.

4.2.4 Requirement 4 (decentralized finality)

For the analysis of finality, we make use of the BAR (Byzantine, Altruistic, Rational) model [31], which has found application in security analysis for blockchain protocols and extensions before, e.g., [28, 32, 33]. Under this model, *Byzantine* users may depart arbitrarily from the protocol for any reason; *altruistic* users always adhere to the protocol rules, and *rational* users will deviate from the protocol to maximize their profit.

The protocol in this paper offers a reward to users submitting the CLAIM transaction. As the reward is at least as high as the submission costs of CLAIM transactions (see Sect. 4.3), rational users have an economic incentive to finalize transfers. However, in any protocol, rational users according to the BAR model, cannot be fully trusted to act rationally in the sense of the protocol's incentive structure since seemingly irrational behavior might be perfectly rational in the context of a larger ecosystem with the protocol being part of it [34]. For instance, rational users might aim at yielding profit in the larger ecosystem by finding ways to bet against the protocol or the value of the asset.

Therefore, rational users are not guaranteed to comply with the protocol rules even with perfectly aligned incentives. Building an open and permissionless system that withstands all participants potentially deviating from the protocol rules appears fundamentally impossible [34]. Thus, in our protocol, not only the users directly involved in a transfer are allowed to post the CLAIM transaction, but rather any user of the system can do it. This provides stronger finalization guarantees as finalization does not depend on a single user acting honestly. It is sufficient if one user out of all users is altruistic to ensure finalization. Notably, the protocol only relies on an altruistic user in case rational users see an incentive in deviating from the proposed protocol.

4.2.5 Requirement 5 (transfer confirmation)

With Requirements 1to 4 fulfilled, cross-blockchain asset transfers with decentralized finality can be realized. However, in Protocol 1, only the destination blockchain *dest* knows when a transfer is finalized. The source blockchain *src* has no way of knowing about the finalization. We have thus proposed Protocol 2 as an extension of Protocol 1 to fulfill the additional requirement of providing *src* with a finalization confirmation. The confirmation takes place in the form of an additional transaction tx_{CONFIRM} being submitted to *src*.

The CONFIRM transaction also has to comply with Requirements 2 to 4. The execution of tx_{CONFIRM} should only be successful if there exists a corresponding CLAIM transaction tx_{CLAIM} successfully executed on *dest* (Requirement 2). Further, it should not be possible to confirm tx_{CLAIM} multiple times (Requirement 3). Additionally, after the successful execution of a CLAIM transaction tx_{CLAIM} on dest, the corresponding CONFIRM transaction tx_{CONFIRM} is eventually submitted to *src* (Requirement 4).

The requirements are fulfilled by Protocol 2 using the same approach that is used for CLAIM transactions. That is, similar to how a CLAIM transaction's payload contains a BURN transaction and corresponding proof data, tx_{CONFIRM} consists of a CLAIM transaction tx_{CLAIM} and corresponding proof data $proof_{tx_{\text{CLAIM}}}$.

The asset contract c_{src} performs the following checks when being invoked by a CONFIRM transaction tx_{CONFIRM} to fulfill Requirement 2: First, c_{src} verifies that tx_{CLAIM} is included in blockchain *dest* and confirmed by enough blocks (see Step 3b of Protocol 2). Second, the contract verifies that the provided CLAIM transaction tx_{CLAIM} has invoked asset contract c_{dest} on blockchain *dest* (see Step 3(c)iii of Protocol 2). Third, c_{src} checks that tx_{CLAIM} has been successfully executed on blockchain *dest* (see Step 3(c)iv of Protocol 2). With these checks in place, Requirement 2 also holds for CONFIRM transactions.

It should not be possible to reuse the same CLAIM transaction tx_{CLAIM} on blockchain *src* to fulfill Requirement 3. Further, it should not be possible to use tx_{CLAIM} on any

blockchain other than *src* for confirming transfer finalization. The first condition is ensured by adding each confirmed CLAIM transaction tx_{CLAIM} to a set T_{CLAIM} (see Step 3(c)vi of Protocol 2). Whenever c_{src} is invoked, it checks whether tx_{CLAIM} provided within tx_{CONFIRM} is included in T_{CLAIM} (see Step 3(c)i of Protocol 2). If so, the request is rejected. To ensure the second condition, the asset contract that has been invoked by the BURN transaction contained within tx_{CLAIM} is compared to c_{src} (see Step 3(c)ii of Protocol 2). If they match, c_{src} is the intended receiver of the confirmation.

The applied incentive structure ensures the confirmation of each transfer finalization on *src* (Requirement 4). Essentially, whenever some user *sender* submits a BURN transaction tx_{BURN} to blockchain *src*, it has to provide some stake *Y*. If user *sender* does not submit tx_{CONFIRM} before a timeout, any user can redeem *Y* by submitting tx_{CONFIRM} to *src*. Hence, there is an economic incentive to confirm transfer finalizations on *src*. Applying the same chain of reasoning for analyzing eventual finality reveals that as long as at least one altruistic user is participating, each transfer finalization on blockchain *dest* is eventually confirmed on blockchain *src*.

As the CONFIRM transaction tx_{CONFIRM} complies with Requirements 2 to 4, we consider the requirement for transfer confirmations fulfilled by Protocol 2.

4.3 Quantitative analysis

This section analyzes transfer costs and duration using the proof-of-concept implementation. We conduct cross-blockchain asset transfers between the public Ethereum test networks Rinkeby and Ropsten. Rinkeby and Ropsten are identical concerning the underlying EVM and the applied inter-block time (about 15 seconds). The main difference is the used consensus algorithm. Rinkeby uses Proof of Authority (PoA), while Ropsten uses Proof of Work (PoW). However, the execution of smart contracts is independent of the consensus algorithm. Therefore, evaluating transfers in one direction provides a sufficient estimation of transfer costs and duration.

Our evaluation consists of conducting 100 transfers of 1 ERC20 token from Rinkeby to Ropsten, i.e., BURN (CONFIRM) transactions are submitted to Rinkeby, whereas CLAIM transactions are executed on Ropsten. We repeat the experiment for both means of transaction inclusion verification, i.e., the oracle and the blockchain relay. Additionally, we also repeat the experiment without transaction inclusion verification to measure the overhead of the respective mechanisms. These mechanisms usually charge a fee to reward the relayers or the oracle nodes. For simplicity, we set the fee to zero. Further, as the protocol extension also includes the steps of Protocol 1, we only use the implementation of Protocol 2 for our experiment.

4.3.1 Transfer costs

For every performed transfer, we measure the gas consumption of all three transaction types, i.e., BURN, CONFIRM, and CLAIM. The obtained results (see Table 1) are outlined in Figs. 4 and 5. Note that the respective figures contain the gas consumption for the protocol and transaction inclusion verification.

First, we examine the gas consumption of the protocol while using the blockchain relay. Here, we get a total gas consumption of 450.02 kGas (standard deviation of 17.16 kGas) for Protocol 1, calculated as the sum of tx_{BURN} and tx_{CLAIM}. The total gas consumption of Protocol 2 is about 1018.09 kGas (standard deviation of 34.8 kGas) and includes the gas consumption of tx_{CONFIRM}, which accounts for the higher overall costs. With an exemplary exchange rate of about 3030.68 EUR per ETH (as of March 2022) and a gas price of 10 GWei, the transfer costs amount to 13.64 EUR for Protocol 1 and 30.86 EUR for Protocol 2.

After examining the gas consumption with the blockchain relay, we look at the gas consumption with the oracle. Here, we get a total gas consumption of 427.31 kGas (standard deviation of 8.75 kGas) for Protocol 1 and 976.35 kGas (standard deviation of 21.21 kGas) for Protocol 2. Using the same exchange rate and gas price leads to transfer costs of 12.95 EUR for Protocol 1 and 29.6 EUR for Protocol 2.

The execution of tx_{BURN} is the cheapest, followed by tx_{CLAIM} and by tx_{CONFIRM}. The differences can be explained by the different payloads of each transaction type. As the payload of tx_{BURN} does not consist of any other transaction and proof data, less data needs to be passed to and processed by the asset contract leading to lower gas consumption. On the contrary, tx_{CLAIM} contains tx_{BURN}, as well as proof data for tx_{BURN}, and tx_{CONFIRM} contains tx_{CLAIM} together with the corresponding proof data, which leads to higher gas consumption.

Table 1 Average gas consumption of the different operations in kGas

	Burn	Claim	Confirm	Protocol 1	Protocol 2
Relay	39.39 ± 0.00	410.62 ± 17.16	568.07 ± 28.52	450.02 ± 17.16	$1,018.09 \pm 34.80$
Oracle	39.39 ± 0.00	364.19 ± 8.75	525.30 ± 14.86	427.31 ± 8.75	976.35 ± 21.21

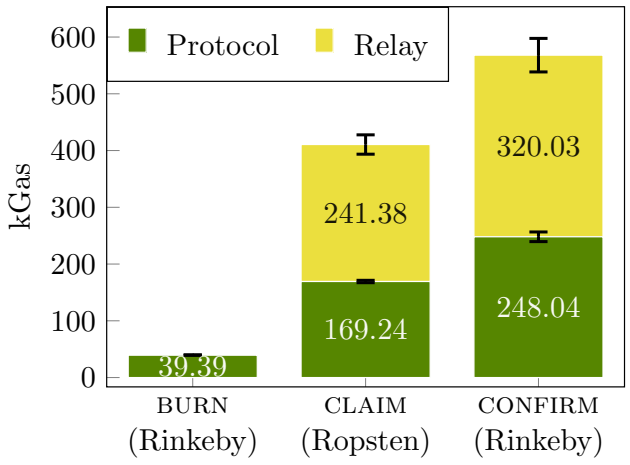

Fig. 4 Average transaction gas consumption with the blockchain relay

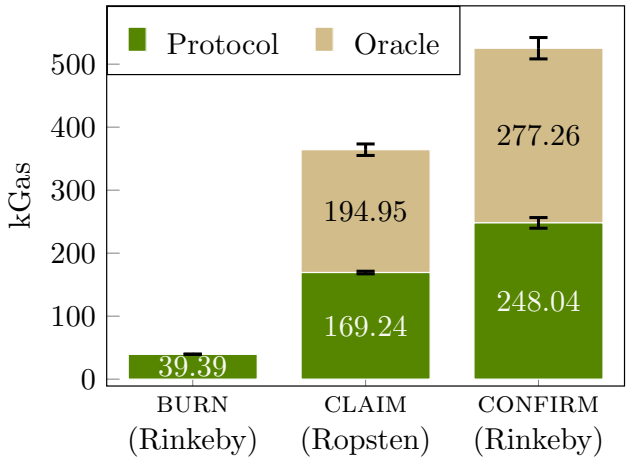

Fig. 5 Average transaction gas consumption with the oracle

The gas consumption of transaction inclusion verification depends on the concrete means of cross-blockchain communication. The oracle checks off-chain if the block header is part of the main chain, thus resulting in lower gas costs than the blockchain relay. Nevertheless, the client has to issue the requests to the oracle separately to provide the hashes of the block headers needed for the proofs. These requests additionally consume 23.4 kGas per request as there are no independent relayers that continuously provide the data. However, an oracle is much cheaper to operate than a blockchain relay since it does not cause any ongoing costs [25]. These operating costs heavily influence the fees charged by the relay or the oracle.

4.3.2 Transfer duration

In this subsection, we analyze the duration of asset transfers, which depends on a few core factors. The current network status and the selected gas price greatly influence the inclusion and confirmation time of a transaction. Therefore, we conduct the experiments at different times over a period of 3 days, while the gas price is estimated automatically to ensure the inclusion of a transaction with a high probability in the next block.

Further, after submitting a BURN transaction, the submitter may wait before posting the corresponding CLAIM transaction. The same applies to a CLAIM transaction, after which the submitter can wait to submit the following CONFIRM transaction. Hence, the overall transfer duration depends on the user executing the transfer. However, despite this uncertainty, we can measure the lowest possible transfer duration by submitting each transaction at the earliest possible time, i.e., as soon as the previous transaction is included in the blockchain and confirmed by enough blocks. In our experiment, we require each transaction on Rinkeby as well as on Ropsten to be confirmed by at least five succeeding blocks.

Finally, the transfer duration also depends on the used transaction inclusion verification mechanism. Therefore, we examine the influence of different transaction inclusion verification mechanisms by measuring the transfer duration using a blockchain relay and an oracle.

As described above, in our experiment, assets are transferred from Rinkeby to Ropsten. Therefore, BURN and CONFIRM transactions are submitted to Rinkeby while CLAIM transactions are submitted to Ropsten. Hence, durations for BURN and CONFIRM transactions were measured on Rinkeby, whereas for CLAIM transactions on Ropsten.

Essentially, CLAIM (CONFIRM) transactions are submitted to Ropsten (Rinkeby) as soon as the corresponding BURN (CLAIM) transactions are included and confirmed on Rinkeby (Ropsten). However, users need to wait until the relay running on Ropsten (Rinkeby) has been brought up to date or the oracle has submitted the data before they can submit the corresponding CLAIM (CONFIRM) transactions. Otherwise, the transactions would not be successful as the transaction inclusion verifier does not have enough information to verify the inclusion of transactions.

To this end, $\Delta_{\text{Inclusion}}$ denotes the duration from the moment a transaction is submitted to Rinkeby (Ropsten) until it is included in some block. $\Delta_{\text{Confirmation}}$ specifies the time it takes for an already included transaction to be confirmed by enough succeeding blocks. Further, Δ_{Relay} denotes the time it takes for the relay to collect enough information to verify the inclusion of transactions.

Alternatively, Δ_{Oracle} specifies the time the oracle takes to query the other blockchain and submit the result.

Figure 6 shows the average transfer duration for each transaction type for both protocols while leveraging a blockchain relay. With an average duration of 55 seconds (standard deviation of 19 seconds), BURN transactions achieve the lowest duration, followed by CONFIRM transactions (average duration of 134 seconds, standard deviation of 49 seconds), and CLAIM transactions (average duration of 210 seconds, standard deviation of 138 seconds). The total duration of Protocol 1 is calculated by summing up the duration of BURN and CLAIM transactions, while the total duration of Protocol 2 also contains the duration of CONFIRM transactions. This calculation leads to an average transfer duration of 265 seconds (standard deviation of 145 seconds) for Protocol 1 and an average duration of 400 seconds (standard deviation of 160 seconds) for Protocol 2. Transfers with Protocol 2 take longer since an additional transaction is required.

The results depicted in Fig. 7 also show the average duration for each transaction type for both protocols. However, an oracle instead of a blockchain relay is used for transaction inclusion verification. Here also, BURN transactions show the lowest transfer duration with 89 seconds (standard deviation of 3 seconds). CLAIM transactions consume on average 248 seconds (standard deviation of 137 seconds), and the execution of CONFIRM transactions averages 117 seconds (standard deviation of 12 seconds). The total duration for Protocol 1 averages 337 seconds (standard deviation of 138 seconds) and 454 seconds (standard deviation of 137 seconds) for Protocol 2.

The results (see Table 2) initially indicate that using a relay leads to a shorter transfer duration. However, looking at the different durations for the inclusion and confirmation of the transactions and the time it takes for the relay and the oracle to include the data, we can discover different results. Further, we also have to consider the lock time, which was set to 0 for the experiments.

Although the inclusion and confirmation times are higher due to the network status, the time it takes for the oracle to provide the necessary data is less than the time it takes for the blockchain relay. This difference is due to the fact that the relay has to be kept up-to-date through the continuous submission of new block headers, which can lead to delays. With the oracle, on the other hand, only two transactions are needed to get the necessary data into the blockchain.

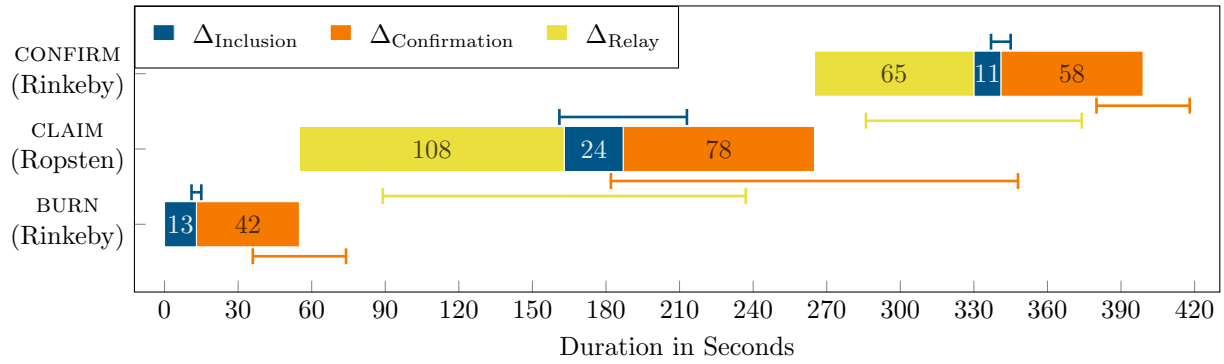

Fig. 6 Average transaction durations with the blockchain relay

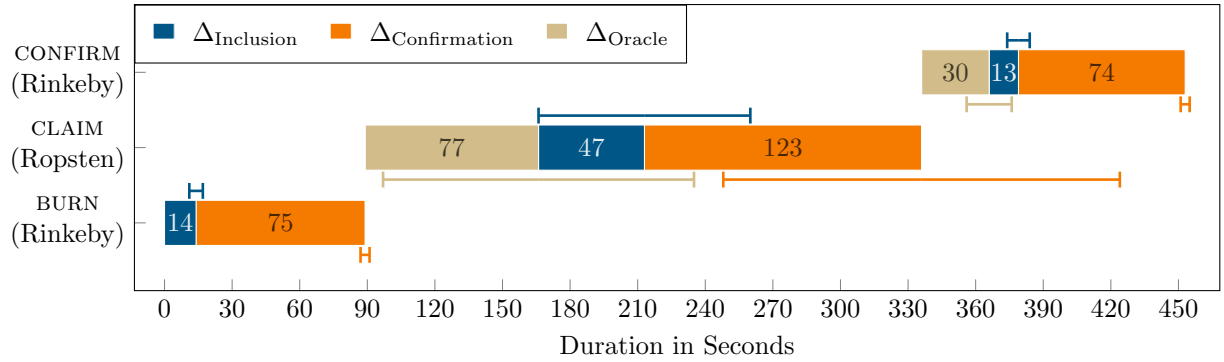

Fig. 7 Average transaction durations with the oracle

Table 2 Average durations for the different operations in seconds

		Relay	Oracle
Inclusion time	Burn	12.84 ± 2.25	14.27 ± 2.83
	Claim	24.26 ± 25.72	47.06 ± 46.87
	Confirm	11.36 ± 4.02	12.57 ± 4.84
Confirmation time	Burn	42.35 ± 19.20	75.16 ± 2.0
	Claim	77.68 ± 83.40	123.18 ± 87.68
	Confirm	57.84 ± 19.37	73.68 ± 2.42
Verification time	Burn	108.04 ± 73.5	77.71 ± 68.98
	Claim	65.14 ± 44.31	30.27 ± 10.30
Protocol 1		265.16 ± 145.05	337.37 ± 137.69
Protocol 2		399.49 ± 160.24	453.88 ± 136.87

The original implementation of ETH Relay foresees a lock time of 5 minutes to allow relayers to dispute invalid block headers. This mechanism causes another delay of at least 5 minutes for Protocol 1 and 10 minutes for Protocol 2. Therefore, using an oracle results in a considerably shorter transfer duration. Overall, the oracle is cheaper and faster, but the blockchain relay provides a higher degree of decentralization. Therefore, it depends on the use case which solution is more viable. In any case, the protocol can support both types of cross-blockchain communication.

5 Related work

Several solutions for cross-blockchain asset transfers have been proposed in the literature [12]. Evaluating the most important existing solutions against the requirements defined in Sect. 3.1 reveals that solutions generally fulfill Requirements 1 to 3. However, they lack with regards to decentralized finality (Requirement 4) or do not provide implementations of the proposed protocols. A summary of the different cross-blockchain asset transfer solutions is provided in Table 3.

In XClaim [35], cross-blockchain asset transfers are realized by first locking assets with a client called "vault" on a "backing" blockchain and reissuing the assets on another "issuing" blockchain. Locking of the assets on the backing blockchain is verified on the issuing blockchain via blockchain relays. However, the locked assets remain with the vault on the backing blockchain. While malicious vaults are penalized, transfer finality still depends on this single actor. In contrast, our protocol enables any client—whether directly involved in the transfer or not—to finalize transfers. Further, transfer confirmations are not foreseen by the XClaim protocol.

Metronome [36] implements cross-blockchain asset transfers for their MET token. Token holders can export MET from one blockchain and then import them on another blockchain via receipts where validators vote on the validity of receipts. While this can prevent illegal transfers, at the time of writing, Metronome cannot be considered decentralized since only authorized nodes can participate as validators. Further, Metronome does not provide concepts for transfer finality and confirmations.

In [37], the authors define a proof-of-burn protocol. The protocol consists of two functions: *GenBurnAddr* and *BurnVerify*. With the former, users can generate new burn addresses with an encoded tag, while the latter allows users to verify the tagged burn. For the verification of the proof-

Table 3 Comparison of different cross-blockchain asset transfer protocols

Reference	Ownership	No claim without burn	Double spend prevention	Decentralized finality	Transfer confirmation	Implementation
XCLAIM [35]	✔	✔	✔			✔
Metronome [36]	✔	✔	✔			✔
Karantias et al. [37]	✔	✔	✔			✔
Pillai et al. [38]	✔	✔	✔			✔
Liu et al. [39]	✔	✔	✔			✔
Kiayias and Zindros [40]	✔	✔				
Gazi et al. [41]	✔	✔				
Zendoo [43]	✔	✔	✔			
van Glabbeek et al. [44]	✔	✔		✔	✔	
DeXTT [46]	✔		✔	✔		✔
Protocol 1	✔	✔	✔	✔		✔
Protocol 2	✔	✔	✔	✔	✔	✔

of-burn, the protocol foresees the usage of Noninteractive Proofs of Proof of Work (NiPoPoWs), oracles, or direct observation of the source blockchain by the miners. In the course of this, the authors show how users can acquire target cryptocurrency (e.g., ERC-20 tokens) by burning the source cryptocurrency. However, the protocol does not provide decentralized finality and transfer confirmations.

Pillai et al. [38] propose a burn-to-claim asset transfer protocol. Also, similar to the protocol presented in Sect. 3, it comprises a burn operation on the source blockchain and a claim operation on the target blockchain. However, to access the transaction on the source blockchain to verify the correct execution of the burn transaction, the protocol requires nodes that can mine in multiple networks and act as gateways. Furthermore, the protocol does not take decentralized finality and transfer confirmations into account.

In [39], the authors present AucSwap, a cross-blockchain token transfer protocol modeled as an auction process. More specifically, the authors leverage the Vickery auction process and atomic swap technology to enable cross-blockchain asset transfers. The protocol consists of two parts: bidding collection and asset exchange. Initially, the seller broadcasts the bidding information and waits for the buyers to return their bids. After receiving all bids, the seller provides the buyers with a list of all bids. Having this information, the buyers can determine who won the auction. After that, the parties use Hash Time-Locked Contracts (HTLCs) to perform the actual exchange of the assets. Compared to our protocol, this solution has more similarities with asset exchanges than asset transfers.

The authors of [40, 41] propose approaches for realizing cross-blockchain transfers between PoW and Proof of Stake (PoS) blockchains, respectively. While [40] verifies transaction inclusions via NiPoPoWs [41, 42] enables transaction inclusion verifications via a novel cryptographic construction called ad-hoc threshold multisignatures (ATMS) ATMS. As such, NiPoPoW and ATMS are used to prove events (e.g., BURN transactions) that occurred on the source blockchain to the destination blockchain. While this satisfies Requirements 1 and 2, Requirements 3 and4 are generally not covered by the protocol. Further, NiPoPoWs currently cannot be implemented in existing PoW blockchains like Bitcoin or Ethereum without introducing a so-called velvet fork, which requires adoption from at least a subset of miners.

Similarly, Zendoo [43] provides a protocol for cross-blockchain asset transfers focussing on zero-knowledge proofs as a method for transaction inclusion verification. However, requirements such as transfer finality are not discussed. Further, the protocol relies on a special sidechain construction and can thus not be easily implemented on existing blockchains.

An approach that takes transfer finality into account is presented by van Glabbeck et al. [44]. The paper proposes a generic protocol for payments across blockchains similar to how multi-hop payment channels operate [45]. The work has a strong focus on finality. Requirements such as double spend prevention are mentioned though not further specified. Also, the protocol has not been implemented and evaluated yet. It does not become clear whether the protocol allows cross-blockchain asset transfers as defined in Sect. 3.1, or only value transfers similar to atomic swaps.

An alternative approach for cross-blockchain asset transfers is introduced in DeXTT [46]. DeXTT describes an asset that can exist on different blockchains at the same time. However, users cannot keep different denominations of the asset on each blockchain. Rather, balances are synchronized across all participating blockchains. While the synchronization process itself is decentralized, the protocol uses a concept called claim-first transactions, where assets are claimed on the other blockchains before being burned on the blockchain on which the transfer was initiated. This clearly violates Requirement 2. Transfer confirmations (Requirement 5) are also not foreseen.

Other works [15, 32, 47] focus on the transfer of value across different blockchains. However, these solutions rather focus on atomic swaps where two different assets are exchanged and do not constitute true cross-blockchain asset transfers as defined by our requirements in Sect. 3.1.

Finally, projects such as Polkadot [48] and Cosmos [49] also aim for generic cross-blockchain interactions. Polkadot implements a Cross-chain Message Passing (XCMP) protocol that enables two separate parachains to communicate with each other. To accomplish this, it makes use of a simple queuing mechanism based on Merkle trees. Cosmos implements the Interblockchain Communication (IBC) protocol proposed in [50]. The IBC protocol is inspired by the TCP/IP protocol and enables the communication between separate ledgers which implement the same interface. Multiple ledgers can establish a connection with each other to create channels over which packages can be transmitted to modules (e.g., smart contracts) on the other ledger. Both protocols have already been implemented by the respective projects. While cross-blockchain asset transfers are mentioned as example use cases, the documentation does not mention specifics on how these transfers are implemented. Further, these projects aim to provide interoperability primarily between specialized blockchains that adhere to certain structures and consensus protocols.

To the best of our knowledge, this work is the first to provide requirements, a specification, and a proof-of-concept implementation of a cross-blockchain asset transfer protocol that also takes transfer finality and transfer confirmations into account.

6 Conclusion

Decentralized cross-blockchain asset transfers are one way to provide interoperability between blockchains. In particular, they prevent vendor lock-ins by allowing blockchain assets to be moved away from the blockchains on which they were originally issued in a completely decentralized way. While a number of solutions for enabling cross-blockchain asset transfers have been proposed, these solutions often focus on specific assets and neglect the fundamental functionality that cross-blockchain asset transfers should offer. In this work, we defined general requirements and specifications for cross-blockchain asset transfer protocols. Providing a proof-of-concept implementation of the proposed protocol, we have shown that requirements such as decentralized finality and transfer confirmations can be fulfilled.

In future work, we will investigate how the concepts of this work can be extended to provide interoperability beyond cross-blockchain asset transfers, e.g., generic message passing between blockchains.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association is gratefully acknowledged.

Author contributions All authors contributed to the conception and design of the protocol. The original paper was written by MS and PF. The first draft of the manuscript was written by MS and is an extended and revised version of the original paper. Data collection and analysis were performed by MS and MK. SS provided supervision, as well as reviewed, and edited this work. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association.

Data availability The datasets generated during and/or analysed during the current work are available in the Zenodo repository, https://zenodo.org/record/6394523#.YkMj5H9BzJU

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. White Paper. Accessed 29 March 2022 (2008) https://bitcoin.org/bitcoin.pdf
2. Tian, F.: An agri-food supply chain traceability system for China based on RFID & blockchain technology. In: 2016 13th International Conference on Service Systems and Service Management (ICSSSM), pp. 1– 6 (2016). IEEE
3. Mettler, M.: Blockchain technology in healthcare: the revolution starts here. In: 2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 1– 3 (2016). IEEE
4. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. Future Gener. Comput. Syst. **107**, 816–831 (2020)
5. Makani, S., Pittala, R., Alsayed, E., Aloqaily, M., Jararweh, Y.: A survey of blockchain applications in sustainable and smart cities. Clust. Comput. **15**, 1–22 (2022)
6. Tseng, L., Yao, X., Otoum, S., Aloqaily, M., Jararweh, Y.: Blockchain-based database in an IoT environment: challenges, opportunities, and analysis. Clust. Comput. **23**(3), 2151–2165 (2020)
7. Al Ridhawi, I., Aloqaily, M., Karray, F.: Intelligent blockchain-enabled communication and services: solutions for moving internet of things devices. IEEE Robot. Automat. Mag. **29**, 10–20 (2022)
8. Berdik, D., Otoum, S., Schmidt, N., Porter, D., Jararweh, Y.: A survey on blockchain for information systems management and security. Inform. Process. Manage. **58**(1), 102329 (2021)
9. Schulte, S., Sigwart, M., Frauenthaler, P., Borkowski, M.: Towards blockchain interoperability. In: Business Process Management: Blockchain and Central and Eastern Europe Forum. LNBIP, vol. 361, pp. 1– 8 (2019)
10. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sánchez, P., Kiayias, A., Knottenbelt, W.J.: SoK: Communication across distributed ledgers. In: International Conference on Financial Cryptography and Data Security. LNCS, vol. 12675, pp. 3–36 (2021)
11. Cai, W., Wang, Z., Ernst, J.B., Hong, Z., Feng, C., Leung, V.C.M.: Decentralized applications: the blockchain-empowered software system. IEEE Access **6**, 53019–53033 (2018)
12. Belchior, R., Vasconcelos, A., Guerreiro, S., Correia, M.: A survey on blockchain interoperability: past, present, and future trends. ACM Comput. Surv. **54**(8), 1–41 (2021)
13. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains. http://kevinriggen.com/files/sidechains.pdf. White Paper. Accessed 29 March 2022 (2014)
14. Poon, J., Buterin, V.: Plasma: scalable autonomous smart contracts. https://www.plasma.io/plasma.pdf. White Paper. Accessed 29 March 2022 (2017)
15. Herlihy, M.: Atomic cross-chain swaps. In: 2018 ACM Symposium on Principles of Distributed Computing (PODC), pp. 245–254 (2018). ACM
16. Sigwart, M., Frauenthaler, P., Spanring, C., Sober, M., Schulte, S.: Decentralized cross-blockchain asset transfers. In: 2021 Third International Conference on Blockchain Computing and Applications (BCCA), pp. 34–41 (2021). IEEE

17. Pillai, B., Biswas, K., Muthukkumarasamy, V.: Blockchain interoperable digital objects. In: International Conference on Blockchain, pp. 80–94 (2019). Springer
18. Cuffe, P.: The role of the ERC-20 token standard in a financial revolution: the case of Initial Coin Offerings. In: IEC-IEEE-KATS Academic Challenge (2018)
19. Casale-Brunet, S., Ribeca, P., Doyle, P., Mattavelli, M.: Networks of Ethereum Non-Fungible Tokens: A graph-based analysis of the ERC-721 ecosystem. In: 2021 IEEE International Conference on Blockchain, pp. 188–195 (2021). IEEE
20. Heiss, J., Eberhardt, J., Tai, S.: From oracles to trustworthy data on-chaining systems. In: 2019 IEEE International Conference on Blockchain, pp. 496–503 (2019). IEEE
21. Peterson, J., Krug, J., Zoltu, M., Williams, A.K., Alexander, S.: Augur: a decentralized oracle and prediction market platform. arXiv:1501.01042 (2015)
22. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: next steps in the evolution of decentralized oracle networks. Accessed 29 March 2022 (2021) https://research.chain.link/whitepaper-v2.pdf?_ga=2.244768454.295607443.1648540372-1480369942.164639185
23. Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A.: Astraea: A decentralized blockchain oracle. In: 2018 IEEE International Conference on Blockchain, pp. 1145–1152 (2018). IEEE
24. Kamiya, R.: Shintaku: An end-to-end-decentralized general-purpose blockchain oracle system. https://gitlab.com/shintaku-group/paper/blob/master/shintaku.pdf. Accessed 29 March 2022 (2019)
25. Sober, M., Scaffino, G., Spanring, C., Schulte, S.: A voting-based blockchain interoperability oracle. In: 2021 IEEE International Conference on Blockchain, pp. 160–169 (2021). IEEE
26. Buterin, V.: Chain interoperability. https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf. Accessed 29 March 2022 (2016)
27. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, Princeton (2016)
28. Frauenthaler, P., Sigwart, M., Spanring, C., Sober, M., Schulte, S.: ETH relay: a cost-efficient relay for ethereum-based blockchains. In: 2020 IEEE International Conference on Blockchain, pp. 204–213 (2020). IEEE
29. Di Angelo, M., Salzer, G.: A survey of tools for analyzing ethereum smart contracts. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), pp. 69–78 (2019). IEEE
30. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Accessed 29 March 2022 (2014) https://ethereum.github.io/yellowpaper/paper.pdf
31. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., Porth, C.: BAR fault tolerance for cooperative services. In: 20th ACM Symposium on Operating Systems Principles (SOSP), pp. 45–58 (2005). ACM
32. Herlihy, M., Liskov, B., Shrira, L.: Cross-chain deals and adversarial commerce. VLDB J. **25**, 1–19 (2021)
33. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gazi, P., Meiklejohn, S., Weippl, E.R.: Pay-to-win: incentive attacks on proof-of-work cryptocurrencies. IACR Cryptology ePrint Archive (2019)
34. Ford, B., Böhme, R.: Rationality is self-defeating in permissionless systems. (2019)arXiv:1908.03999
35. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.: XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 193–210 (2019). IEEE
36. Autonomous Software: Metronome: Owner's Manual. Version 0.99 (Last Updated 2019-08-15). Accessed 29 March 2022 (2018). https://github.com/autonomoussoftware/documentation/blob/master/owners_manual/owners_manual.md
37. Karantias, K., Kiayias, A., Zindros, D.: Proof-of-burn. In: International Conference on Financial Cryptography and Data Security, pp. 523–540 (2020). Springer
38. Pillai, B., Biswas, K., Hóu, Z., Muthukkumarasamy, V.: The burn-to-claim cross-blockchain asset transfer protocol. In: 2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 119–124 (2020). IEEE
39. Liu, W., Wu, H., Meng, T., Wang, R., Wang, Y., Xu, C.-Z.: AucSwap: a vickrey auction modeled decentralized cross-blockchain asset transfer protocol. J. Syst. Archit. **117**, 102102 (2021)
40. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: International Conference on Financial Cryptography and Data Security, pp. 21–34 (2019). Springer
41. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 139–156 (2019). IEEE
42. Kiayias, A., Miller, A., Zindros, D.: Non-interactive Proofs of Proof-of-Work. In: 24th International Conference on Financial Cryptography and Data Security—Revised Selected Papers. LNCS, vol. 12059, pp. 505–522 (2020). Springer
43. Garoffolo, A., Kaidalov, D., Oliynykov, R.: Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), pp. 1257–1262 (2020). IEEE
44. van Glabbeek, R., Gramoli, V., Tholoniat, P.: Cross-chain payment protocols with success guarantees. arXiv preprint arXiv:1912.04513 (2019)
45. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 455–471 (2017). ACM
46. Borkowski, M., Sigwart, M., Frauenthaler, P., Hukkinen, T., Schulte, S.: DeXTT: deterministic cross-blockchain Token transfers. IEEE Access **7**, 111030–111042 (2019)
47. Thomas, S., Schwartz, E.: A protocol for interledger payments. https://interledger.org/interledger.pdf. Accessed 29 March 2022 (2015)
48. Wood, G.: Polkadot: vision for a heterogeneous multi-chain framework. https://polkadot.network/PolkaDotPaper.pdf. Accessed 29 March 2022 (2016)
49. Kwon, J., Buchman, E.: Cosmos whitepaper: a network of distributed ledgers. https://cosmos.network/resources/whitepaper. Accessed 29 March 2022 (2020)
50. Goes, C.: The interblockchain communication protocol: an overview. arXiv preprint arXiv:2006.15918 (2020)

Michael Sober received his master's degree in Software Engineering & Internet Computing from TU Wien in 2020. He is a research assistant and Ph.D. student at the Institute for Data Engineering at TU Hamburg and working on blockchain interoperability solutions in the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Marten Sigwart received his master's degree in Computer Science from TU Berlin, Germany, in 2018. He was a project assistant in the TAST Research Project at TU Wien, working on blockchain interoperability solutions, and is now a full-stack software engineer at Datawrapper, Berlin.

Philipp Frauenthaler received his master's degree in Software Engineering & Internet Computing from TU Wien in 2018, where he was also a project assistant in the Distributed Systems Group. Before joining the Distributed Systems Group in February 2019, he worked for five years as a software engineer, developing enterprise software for insurances. He is now a senior software engineer at the Austrian Federal Computing Center in Vienna, Austria.

Christof Spanring is currently the Chief Technology Officer of Bonfire Group BV. He was the lead blockchain researcher at Pantos GmbH and worked on blockchain interoperability solutions, contributing to the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Max Kobelt is currently pursuing his bachelor's degree at TU Hamburg. He is also a student assistant in the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Stefan Schulte heads the Institute for Data Engineering at Hamburg University of Technology, Germany, and leads the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT). His research interests include the areas of data stream processing, the Internet of Things, and distributed ledger technologies. Findings from his research have been published in more than 100 refereed scholarly publications.

12

Efficient Cross-Blockchain Token Transfers with Rollback Support

DOI: 10.1109/DAPPS61106.2024.00009

Keywords: *blockchain interoperability, decentralized asset transfers, cross-blockchain communication, digital assets, rollback, rollups*

Abstract. Blockchain interoperability still poses a significant challenge to the practical applicability of Distributed Ledger Technologies (DLTs). However, several cross-blockchain protocols already exist, allowing, e.g., the transfer of tokens between heterogenous blockchains. Unfortunately, executing these protocols is expensive and often does not consider eventual inconsistencies, which can occur if the involved blockchains fork.

Therefore, we propose a cross-blockchain token transfer protocol operating on Layer 2 to decrease transaction costs and enable the ability to revert transactions. The protocol uses the approach of zk-rollups to move storage and computation off-chain and only stores state commitments on Layer 1. Hence, the states of the involved token contracts can quickly revert to an older state root to account for possible forks of the involved blockchains and ensure consistency.

Efficient Cross-Blockchain Token Transfers with Rollback Support

Michael Sober*†, Markus Levonyak‡, Stefan Schulte*†
*Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things
†Hamburg University of Technology, Hamburg, Germany, {michael.sober, stefan.schulte}@tuhh.de
‡Pantos GmbH, Vienna, Austria, markus.levonyak@bitpanda.com

***Abstract*—Blockchain interoperability still poses a significant challenge to the practical applicability of Distributed Ledger Technologies (DLTs). However, several cross-blockchain protocols already exist, allowing, e.g., the transfer of tokens between heterogenous blockchains. Unfortunately, executing these protocols is expensive and often does not consider eventual inconsistencies, which can occur if the involved blockchains fork.**

Therefore, we propose a cross-blockchain token transfer protocol operating on Layer 2 to decrease transaction costs and enable the ability to revert transactions. The protocol uses the approach of zk rollups to move storage and computation off-chain and only stores state commitments on Layer 1. Hence, the states of the involved token contracts can quickly revert to an older state root to account for possible forks of the involved blockchains and ensure consistency.

***Index Terms*—blockchain interoperability, decentralized asset transfers, cross-blockchain communication, digital assets, rollback, rollups**

I. Introduction

Blockchains, or more generally Distributed Ledger Technologies (DLTs), enable a peer-to-peer network of nodes to manage a distributed ledger of transactions without a trusted third party. The network of nodes follows specific rules to reach a consensus about the state of the distributed ledger and ensure its integrity, consistency, and security. Initially, DLTs only formed the backbone of cryptocurrencies like Bitcoin [1] and Ethereum [2]. However, the rapid development of DLTs over the past years led to major improvements and the application of DLTs in other areas, such as supply chain management [3], healthcare [4], the Internet of Things [5], and others. These improvements include, amongst others, the ability to create decentralized applications with smart contracts [2], better scalability through Layer 2 protocols [6], and privacy-preserving mechanisms to ensure user privacy [7]. However, there are still open problems that hinder the widespread adoption and applicability of DLTs in real-world use cases.

Blockchain interoperability is one of the major issues that hinder the adoption of blockchain technology [8]. In the past years, many different blockchains emerged due to the evolving needs of various stakeholders. These blockchains are tailored to specific use cases or provide unique features, as a single general-purpose blockchain can not fulfill all requirements imposed in different application areas. Unfortunately, the interaction between these blockchains is usually not foreseen, through which blockchains become isolated systems that cannot interact with other blockchains or external systems [9]. Different accounts cannot inflict state changes across multiple blockchains or retrieve data from external systems. These limitations result in users being bound to specific blockchains without the ability to migrate to novel blockchains easily. However, blockchain interoperability would enable users to use their tokens on different blockchains, utilizing blockchain-specific applications, features, or security guarantees. Therefore, there is a strong need for solutions that help to establish interoperability between different blockchains and allow, e.g., the transfer of tokens between different blockchains.

Transferring value in the form of tokens between different blockchains is a desired mechanism that allows users to freely decide which blockchain they want to transfer and use their funds. The transfer usually occurs by burning (destroying) the tokens on the source blockchain and minting (recreating) the tokens on the target blockchain [10]. However, this requires a mechanism for cross-blockchain communication as the target blockchain must be able to verify that the tokens got destroyed on the source blockchain before the recreation on the target blockchain. A problem that many of these protocols have is the high cost of transferring a token directly on Layer 1 of the involved blockchains. Further, current mechanisms do not support rollbacks in the case of forks, which can lead to inconsistencies in the state of the token contracts.

Therefore, we propose a cross-blockchain token transfer protocol operating on Layer 2. By moving storage and computation of the token contract off-chain, we can considerably reduce the transaction costs since the blockchain only needs to verify the correctness of the computation through Zero-Knowledge Proofs (ZKPs). Further, since the blockchain only needs to store the state root of the token contract, rollbacks are easy to execute since they only require reverting to an older state root instead of reverting each account individually. The protocol applies a cross-blockchain communication mechanism, such as oracles and blockchain relays, to transfer the state of the rollup contract on the source blockchain to the rollup contract on the target blockchain. Further, we provide a prototypical implementation and conduct several experiments to evaluate the costs, performance, and memory usage.

The remainder of this paper is structured as follows: Section II introduces background information. Then, we propose the protocol for cross-blockchain token transfers in Section III. We evaluate the protocol in Section IV and discuss related work in Section V. Finally, Section VI concludes the paper.

II. Background

In this section, we provide the necessary background information about the applied concepts. For that, we describe cross-blockchain communication and rollups.

A. Cross-Blockchain Communication

Cross-blockchain communication is a fundamental building block for interoperable blockchain systems [8]. A cross-blockchain communication mechanism allows two blockchains to exchange arbitrary data and synchronize transaction execution to reach a consistent state. With the ability to transfer arbitrary data between different blockchains, it is possible to build cross-blockchain protocols to transfer assets or conduct smart contract calls across the boundaries of blockchains. Unfortunately, cross-blockchain communication always requires a trusted third party [11] since most blockchains cannot directly interact with each other, as these blockchains have their specific consensus mechanism, governance, and tokenomics. Many different cross-blockchain communication mechanisms exist with different advantages and trade-offs. In the following, we describe the three major mechanisms for cross-blockchain communication: notary schemes, blockchain relays, and blockchains of blockchains.

The simplest form of cross-blockchain communication is the usage of notary schemes. A notary is a trusted entity or group that acts as an oracle [12], i.e., a system that bridges blockchains with external systems. The oracle retrieves data from the source blockchain to transfer the data to the target blockchain. Further, the notary ensures the correctness and integrity of the transmitted data. The oracle usually consists of a smart contract on the target blockchain, which receives data from an off-chain component composed of several parties that vote on the correctness of the data [13]. The oracle contract only has to verify the validity of the attestation from the notary. Therefore, a notary scheme is a flexible and lightweight solution since the on-chain component does not require a tailored solution for different blockchains. Unfortunately, the degree of decentralization is lower than other solutions for cross-blockchain communication since it depends on a trusted group. Notary schemes, therefore, present a balance between ease of implementation and the degree of trust required in the involved parties.

Blockchain relays constitute another mechanism for cross-blockchain communication [14], [15]. A blockchain relay is a smart contract deployed on the target blockchain, which acts as a light client for the source blockchain. For that, a third party must usually transfer information from the source blockchain to the target blockchain, e.g., relay block headers or update the current validator set. The relay contract verifies the consensus of the source blockchain and ensures the correctness of transferred data. Therefore, the third party does not need to be trusted. A single honest relayer is sufficient for the system's operation and security. The major advantage of blockchain relays is the high degree of decentralization compared to other cross-blockchain communication mechanisms. Unfortunately, the operation of blockchain relays is very resource-intensive, depending on the source blockchain's consensus mechanism.

Another major mechanism for cross-blockchain communication is the blockchains of blockchains approach applied in Cosmos [16] and Polkadot [17]. Blockchains of blockchains form a network of blockchains using a central blockchain for communication between different blockchains. The blockchains include block headers in the central relay chain to communicate state information to the other chains. A major disadvantage is the big overhead of operating an additional blockchain.

B. Rollups

Blockchains suffer from a significant scalability issue that limits the number of Transactions per Second (TPS) a blockchain can process [18], e.g., Bitcoin can only process 7 TPS [19]. Ethereum experiences the same problem and can only process around 20 TPS [20], while modern payment systems like Visa can handle almost 6000 TPS [21]. Further, the storage space required to store the blockchain grows steadily over time, which makes it harder for nodes to store an entire copy of the distributed ledger. Hence, there is a strong need for scalability solutions.

Currently, there are two basic approaches to improve the scalability of blockchains [18]. Layer 1 solutions strive to improve the core protocol of the blockchain directly, and Layer 2 solutions create additional solutions on top of Layer 1, which increase the scalability of a blockchain without making any changes to the core protocol. Since Layer 2 solutions do not require any changes to Layer 1 and allow for customization of specific applications, they constitute a potential candidate to improve current cross-blockchain token transfer protocols.

Rollups, which move state storage and computation off-chain [22], are especially interesting. A rollup usually requires a rollup contract managing the state of the rollup stored as a Merkle tree. Further, a rollup requires off-chain components called operators, which advance the state of the rollup. These operators are responsible for bundling and executing transactions off-chain and computing the new state of the rollup. Instead of executing each transaction on Layer 1, an operator only submits the updated state root and compressed transaction data to the rollup contract.

Rollups must ensure data availability by submitting transaction data to Layer 1. The availability of transaction data on Layer 1 ensures that anyone can recompute and verify the state of the rollup and act as an operator. Further, users must be able to query their account data for withdrawals and deposits. Other solutions exist that store transaction data off-chain [23] using a data availability committee [24]. However, these solutions do not achieve the same security as Layer 1, as the committee needs to be trusted.

Further, rollups need to ensure the correctness of the state changes for which two major approaches exist: optimistic rollups and zk rollups [25]. Optimistic rollups assume that the state changes by an operator are correct and only check their correctness if there is an invalid state change [26]. Therefore,

users must monitor state changes and submit fraud proofs to trigger dispute resolutions. The rollup gives users a specific time window to submit fraud proofs, meaning that transactions are only final once this time window passes. Zk rollups, on the other hand, require an operator to submit a validity proof, e.g., a Zero-Knowledge Succinct Non-interactive Argument of Knowledge (zk-SNARK) [27], [28]. These proofs enable verifying arbitrary computations represented as algebraic circuits. An operator submits the validity proof, the compressed transaction data, and the updated state root to the rollup contract. The rollup contract immediately verifies the proof and only accepts the new state root if the proof is valid. The immediate finality of zk rollups makes them an excellent candidate to realize cross-blockchain token transfers through rollups.

III. Cross-Blockchain Token Transfers

In this section, we define the requirements of a cross-blockchain token transfer protocol. Further, we provide an overview of the protocol and a detailed description. Finally, we discuss the applied incentive mechanism and rollback support.

A. Requirements

Many works have already defined the requirements for cross-blockchain token transfer (see Section V). Hence, we base the requirements on previous work [29] and modify or add requirements to mitigate the drawbacks of existing protocols, which operate mostly on Layer 1.

Requirement 1 - Ownership: Only the owner of a token can transfer its tokens to another user.

Requirement 2 - No Claim without Burn: The claim of tokens on the target blockchain requires a valid burn with the same amount on the source blockchain.

Requirement 3 - Double Spend Prevention: The cross-blockchain token transfer protocol should prevent double spending by only allowing the claim of tokens on one target blockchain.

Requirement 4 - Decentralized Finality: The burn of tokens for a cross-blockchain token transfer always follows a claim.

Requirement 5 - Rollback Support: The blockchains involved in a cross-blockchain token transfer can revert to an older state due to forks if they do not provide immediate finality. In this case, the protocol must ensure the token contracts' consistency and revert the state of all token contracts that depend on the reverted blockchain.

Requirement 6 - Cost Efficiency: A cross-blockchain token transfer is more expensive than a regular token transfer since it incurs additional costs using a cross-blockchain communication mechanism and multiple blockchains. Hence, a cross-blockchain token transfer protocol should strive to keep the costs as low as possible.

B. Overview

The protocol requires several on-chain and off-chain components (see Fig. 1). The on-chain components include the smart contracts for the rollup and cross-blockchain communication mechanism. The off-chain components comprise the operators of the rollups and components for cross-blockchain communication, such as relayers, oracles, or blockchains of blockchains.

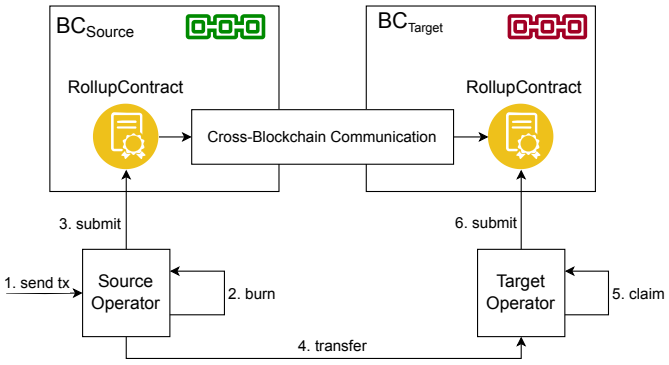

Fig. 1. Abstract overview of the protocol's execution steps.

Initially, users create transactions and send them to an operator of the source rollup (Step 1). The system allows two different transaction types: regular transactions and cross-blockchain transactions. A regular transaction is a normal value transfer between two accounts on the same blockchain, and cross-blockchain transactions allow burning tokens on the source blockchain and recreating tokens on the target blockchain.

An operator of the source rollup collects cross-blockchain transactions in its mempool until it has enough transactions with high enough transaction fees. Afterward, the source operator bundles and executes the transactions to update the rollup contract on the source blockchain. For that, it executes the cross-blockchain transactions locally and computes the new state root. During the execution of the transactions, the source operator burns the tokens of the involved accounts (Step 2). Further, the source operator generates a zk-SNARK to prove the correctness of the state transition. Finally, the source operator submits the transaction data, new state root, and proof to the rollup contract on the source blockchain (Step 3).

After that, the source operator uses a cross-blockchain communication mechanism to transfer the state information of the source rollup contract to the target blockchain and provides the state information and cross-blockchain transaction data to an operator of the target blockchain (Step 4). Then, the target operator follows the same steps as the source operator and executes the same cross-blockchain transactions, which, however, recreates the tokens for the involved accounts and distributes the transaction fees (Step 5). Finally, the target operator submits the new state root and the proof to the target rollup contract (Step 6). An incentive mechanism ensures that the involved operators execute the protocol and receive compensation for the resources they use. In the following section, we explain each protocol step in more detail.

C. Protocol

Initially, a user creates a cross-blockchain transaction $tx_x = \langle nonce, amount, sender, receiver, fee, dest, sig \rangle$. These pa-

Algorithm 1 *Burn* circuit

```
1:  function UPDATESTATE(preStateRoot, postStateRoot,
    transactionsRoot, transactions, senders,
    senderMerkleProofs, blockchains)
2:      leaves ← []
3:      intermediateRoot ← preStateRoot
4:      for i = 0 to len(transactions) do
5:          leaves[i] ← hash(tx[i])
6:          containsBlockchain(blockchains, tx[i].dest)
7:          verifySignature(tx[i].sig, senders[i], tx[i])
8:          intermediateRoot ← burn(intermediateRoot,
                tx[i], senders[i], senderMerkleProofs[i])
9:      end for
10:     actTransactionsRoot ← computeRoot(leaves)
11:     assert(transactionsRoot = actTransactionsRoot)
12:     assert(intermediateRoot = postStateRoot)
13: end function
14: function BURN(root, tx, sender, merkleProof)
15:     assert(merkleProof.RootHash = root)
16:     assert(merkleProof.Path[0] = hash(sender))
17:     verifyMerkleProof(merkleProof, sender.index)
18:     assert(sender.nonce = tx.nonce)
19:     assert(sender.pubKey = tx.sender)
20:     assert(sender.balance ≥ tx.amount + tx.fee)
21:     sender.nonce ← sender.nonce + 1
22:     sender.balance ← sender.balance − tx.amount
            −tx.fee
23:     merkleProof.Path[0] ← hash(sender)
24:     return ComputeRootFromPath(merkleProof)
25: end function
```

rameters are already known and do not differ from regular blockchain transactions except $dest$ where $dest \in B$ and B is the set of possible destination blockchains. Upon signing tx_x, a user sends tx_x to an operator of the rollup contract of the source blockchain.

1) Burn: An operator of a source rollup contract receives transactions from users and stores these transactions in the operator's mempool until enough transactions are available for execution. The source operator verifies each transaction by checking the following: the nonce is equal to the sender's nonce, the sender is part of the state tree of the rollup on the source blockchain, the sender owns enough funds to cover the amount and the transaction fee, the destination blockchain is part of the supported blockchains, and that the signature is valid. After receiving enough valid transactions, a source operator creates a bundle of valid transactions. Then, the source operator applies each transaction locally to compute the new state root of the source rollup contract to compute the witness for the *Burn* circuit, which proves the correct execution of the bundle of transactions.

The *Burn* circuit (see Alg. 1) takes the *preStateRoot*, *postStateRoot*, *transactionsRoot*, *transactions*, *senders*, *senderMerkleProofs*, and *blockchains* as input. The *preStateRoot* is the current state root stored by the rollup contract, while the *postStateRoot* is the new state root after applying the provided transactions. Further, the circuit requires the transactions with the corresponding root of the transaction Merkle tree. The *Burn* circuit also needs the account data of the senders with the respective Merkle proofs. Finally, the *Burn* circuit requires a set of supported blockchains.

The *Burn* circuit verifies and applies all transactions in a loop (Lines 4 to 9). Initially, the circuit adds the transaction's hash to the leaves (Line 5) and verifies if the set of supported blockchains contains the transaction's destination (Line 6) to ensure that a transfer to the defined blockchain is possible. After that, it verifies the transaction's signature (Line 7) and invokes the *Burn* function (Line 8) to update the state.

The *Burn* function (Line 14) updates the senders' account and computes an intermediary state root. Initially, the circuit verifies the inclusion of the sender in the state tree by verifying the Merkle proof (Lines 15 to 17). After that, the circuit verifies if the nonce and public key of the sender match the transaction data and if the sender has enough funds by checking that it is higher than the transaction amount and fees (Lines 18 to 20). If these checks are successful, the circuit increments the nonce (Line 21) and updates the balance by deducting the transaction amount and fees (Line 22). This account update also requires an update of the Merkle proof to reflect the state changes (Line 23). The last step requires computing the new intermediate root from the sender's updated Merkle path (Line 24).

After processing all transactions, the circuit computes the transaction root (Line 10). Finally, the circuit verifies that the actual computed transactions root equals the *transactionsRoot* (Line 11) and the final *intermediaryRoot* equals the *postStateRoot* (Line 12).

After generating the proof, the source operator submits the proof, the updated state root, and the transactions to the rollup contract on the source blockchain. The rollup contract verifies the proof and, if the proof is valid, stores the new state root.

2) Information Transfer: After executing the burn on the source blockchain, the information is still not readily available on the target blockchain. For that, the protocol applies a cross-blockchain communication mechanism such as oracles or blockchain relays. The information required on the target blockchain is the transaction root of the executed transactions on the source blockchain, which burned the tokens. Further, the target blockchain requires information about the source operator that submitted the state update, which is required for the incentive mechanism. In the following, we describe the previously mentioned possibilities to transfer this information.

A source operator uses a cross-blockchain oracle to request information about the latest state change. Therefore, a source operator has to issue and pay for the request to a cross-blockchain oracle. Upon receiving the request, the cross-blockchain oracle queries the source blockchain for the specified data and, after reaching an agreement about the validity of the data, attests to the correctness of the data and submits it to the target blockchain. Here, the cross-blockchain oracle can directly submit the required data or the transaction

corresponding to the state change of the rollup contract on the source blockchain.

Another approach is to use a blockchain relay, i.e., a light client deployed as a smart contract on the target blockchain. In this case, a target operator must provide the block which includes the transaction and the respective Merkle proof. Hence, the rollup contract can call the relay contract to check the validity and inclusion of the block in the source blockchain. After that, the rollup contract can execute Simplified Payment Verification (SPV) to prove the inclusion of the transferred data. After the transfer, the data is available for the rollup contract.

In addition to the information transfer between the source blockchain and the target blockchain, the operators also need to exchange the same information off-chain. While the target operator could also read the source blockchain, it is more convenient if the source operator directly sends the data to the target operator.

3) Claim: After the information transfer, an operator of the target rollup can query the transactions from the source blockchain or receive it from the source operator. Initially, the target operator checks whether the receivers are included in the state tree of the rollup on the destination blockchain and verifies that the transactions' destination is indeed the destination blockchain. After that, the target operator applies each transaction off-chain to compute the new state root of the rollup contract on the destination blockchain and the witness for the *Claim* circuit, which proves the correct execution of the transactions.

The *Claim* circuit, as outlined in Alg. 2, necessitates the input of the *preStateRoot*, *postStateRoot*, *transactionsRoot*, *transactions*, *receivers*, *srcOp*, *targetOp*, *receiverMerkleProofs*, *sourceOpMerkleProof*, *targetOpMerkleProof*, and *blockchain*. Compared to the *Burn* circuit, the *Claim* circuit requires receivers instead of senders and the operators to transfer the reward. Additionally, the blockchain identifier is required to verify the destination of the transactions.

Similar to the *Burn* circuit, the *Claim* circuit verifies and applies all transactions in a loop (Lines 5 to 10). Initially, the *Claim* circuit also adds the transaction's hash to the leaves (Line 6) and checks if the transaction's destination is the target blockchain (Line 7). This check ensures that the claim is only possible on the specified blockchain. After that, analog to the *Burn* circuit, the *Claim* circuit invokes the *Claim* function to update the state (Line 8) and adds half of the transaction fee to the current operator reward (Line 9).

The *Claim* function (Line 17) checks if the state tree includes the receiver by checking the Merkle proof (Lines 18 to 20). In the next step, the *Claim* circuit verifies that the provided receiver matches the transaction's receiver, adds the number of tokens to the receiver's balance, and updates the corresponding Merkle proof (Lines 21 to 23). After that, the circuit computes an *intermediateRoot* from the receiver's updated Merkle proof (Line 24).

After processing the claims, the circuit uses the *reward* function (Line 26) to distribute the rewards to the operators (Lines 11 and 12). Initially, it checks the operator's inclusion in the state tree (Lines 27 to 29). Then, the circuit updates the operator's balance by adding half of the overall transaction fees, and updates the respective Merkle proof (Lines 30 and 31). After this update, the circuit recomputes the *intermediateRoot* again (Line 32). Finally, after processing all transactions, the *Claim* circuit computes and verifies the *transactionsRoot* (Lines 13 and 14) and checks that the final *intermediateRoot* matches the *postStateRoot* (Line 15).

Algorithm 2 *Claim* circuit

```
1:  function UPDATESTATE(preStateRoot, postStateRoot,
    transactionsRoot, transactions, receivers,
    srcOp, targetOp, receiverMerkleProofs,
    srcOpMerkleProof, targetOpMerkleProof,
    blockchain)
2:      leaves ← []
3:      operatorReward ← 0
4:      intermediateRoot ← preStateRoot
5:      for i = 0 to len(transactions) do
6:          leaves[i] ← hash(tx[i])
7:          assert(blockchain, tx[i].dest)
8:          intermediateRoot ← claim(intermediateRoot,
                tx[i], receivers[i],
                receiverMerkleProofs[i])
9:          opReward ← opReward + (tx[i].Fee/2)
10:     end for
11:     intermediateRoot ← reward(intermediateRoot,
            operatorReward, srcOp,
            srcOpMerkleProof)
12:     intermediateRoot ← reward(intermediateRoot,
            operatorReward, targetOp,
            targetOpMerkleProof)
13:     aTransactionsRoot ← computeRoot(leaves)
14:     assert(transactionsRoot = aTransactionsRoot)
15:     assert(intermediateRoot = postStateRoot)
16: end function
17: function CLAIM(root, tx, receiver, merkleProof)
18:     assert(merkleProof.RootHash = root)
19:     assert(merkleProof.Path[0] = hash(receiver))
20:     verifyMerkleProof(merkleProof, receiver.index)
21:     assert(receiver.pubKey = tx.receiver)
22:     receiver.balance ← receiver.balance + tx.amount
23:     merkleProof.Path[0] ← hash(receiver)
24:     return ComputeRootFromPath(merkleProof)
25: end function
26: function REWARD(root, reward, operator,
    merkleProof)
27:     assert(merkleProof.RootHash = root)
28:     assert(merkleProof.Path[0] = hash(operator))
29:     verifyMerkleProof(merkleProof, operator.index)
30:     operator.balance ← operator.balance + reward
31:     merkleProof.Path[0] ← hash(operator)
32:     return ComputeRootFromPath(merkleProof)
33: end function
```

After successfully generating the proof, the operator submits the proof, transaction data, and the *postStateRoot* to the rollup contract on the target blockchain. Upon submission, the rollup contract again verifies the proof and, if the proof is valid, stores the new state root. Further, the rollup contract marks the *transactionsRoot* as already used to prevent double-spending.

D. Incentive Mechanism

The alignment of incentives is essential to decentralized cross-blockchain token transfer protocols. An incentive mechanism encourages participants to use their resources to execute the protocol, encourages honesty, and discourages malicious behavior. Without correctly aligned incentives, nodes may not use the system or deviate from the protocol. Hence, the proposed solution uses crypto-economic incentives to compensate and motivate operators to enforce the protocol's rules.

An operator must spend many resources executing state changes, generating the zk-SNARK, perform cross-blockchain communication, and submitting the data to the blockchain. The former incurs costs for the necessary infrastructure, while the latter requires the operator to pay fees for cross-blockchain communication and transaction fees. Therefore, each user has to pay a transaction fee for executing a cross-blockchain token transfer, as is already done by most blockchains and rollups. For that, a user specifies the fee in the transaction, which needs to be greater than zero. An operator can then decide whether to process the transaction or not.

Further, an additional incentive is necessary to ensure the complete execution of a cross-blockchain token transfer. A cross-blockchain token transfer should ensure that for each execution of a cross-blockchain transaction on the source blockchain, there is a corresponding execution of this transaction on the target blockchain. Other protocols offer a subset of the transferred tokens as a fee if the original user does not issue the transaction to the target blockchain within a specified time window [10]. However, this incentive mechanism only works if the transferred value is high enough, which would require a minimal transaction value to ensure that users finalize cross-blockchain token transfers. Another possibility requires users to lock a high collateral for the compensation of other users finalizing the transfer. Unfortunately, this solution resembles a higher entry barrier and makes the users' funds unusable for extended periods.

Therefore, we propose to use the transaction fees already applied for the operators' compensation to incentivize the operators to execute cross-blockchain transactions and submit the state updates to the involved blockchains. For that, the protocol does not reward the source operator with the transaction fees of cross-blockchain transactions on the source blockchain. Instead, the source operator can only receive the transaction fees of cross-blockchain transactions on the target blockchain. This approach ensures dynamic transaction fees that can adapt to the costs of using the involved blockchains and do not require users to lock collateral. Further, the source operator only includes cross-blockchain transactions if the transaction fees are high enough to cover the costs for cross-blockchain communication and execution on both blockchains. Otherwise, the source operator can not ensure to get a reward. Both operators receive their share of the transaction fees on the target blockchain during the claim step.

E. Rollback Support

The design of previous cross-blockchain token transfer protocols always required storage and computation on Layer 1 (see Section V), making state changes in the token contract especially expensive. This issue is problematic in the context of possible inconsistencies in a cross-blockchain ecosystem where some blockchains only provide probabilistic finality, and forks can sometimes occur. In a cross-blockchain protocol, the state changes must be consistent over all involved blockchains, which is hard to ensure. Therefore, a mechanism that allows for reverting the state of the token contract in case of forks on other blockchains is necessary. Thus, the design of a cross-blockchain asset transfer protocol must allow for easy rollbacks.

The proposed protocol uses rollups to move state storage and computation for the token contracts off-chain. As previously mentioned, the rollup contract manages the state of the rollup and only stores the corresponding state root. This approach makes reverting to older states easy since it only requires reverting a single variable in the rollup contract. Hence, if there is a fork on another blockchain, the rollup contract allows a reset of the state root with proof that there was a fork on another blockchain. Several possibilities exist for generating such proof: a light client of the target blockchain as a smart contract, an oracle monitoring the other blockchain for forks, etc. Upon receiving such proof, the rollup contract reverts to a state that does not involve any transaction on blocks part of this fork.

IV. Evaluation

In this section, we evaluate the protocol regarding different aspects. Initially, we describe the experimental setup. Afterward, we discuss the results of our experiments, where we examine the costs, performance, and memory usage.

A. Experimental Setup

For our experiments, we implement the smart contracts in Solidity and the necessary circuits for burning and claiming tokens in Go using the gnark zk-SNARK library [30]. We execute our experiments on a machine equipped with an Intel Core i7-10510U CPU running Ubuntu 20.04.6 using 8 cores with a clock speed of 1.80 GHz. The machine has 16GB LPDDR3 RAM with a frequency of 2133 MHz and a 2 TB SSD. Further, we deploy the smart contracts on a local blockchain using Ganache 2.7.1. We provide the source code of the implemented smart contracts and circuits on GitHub[1].

We simulate an operator executing transactions to measure the proof generation time and memory usage. We fix the number of accounts in the rollup to 65,536 and, therefore, the depth of the state tree to 17 due to the machine's resource

[1] https://github.com/soberm/l2_x_chain_transfer

constraints. We execute batches of transactions with different sizes and repeat the experiment for each batch size ten times. Further, we submit the generated proofs and transaction data to the rollup contract to measure the gas consumption.

B. Costs

Initially, we examine the costs of burning and claiming tokens. The Ethereum network uses Gas as a unit of measurement for computational work to prevent network abuse and as an economic incentive. Each transaction requires a specific amount of Gas, which the transaction sender pays for in the blockchain's native currency. If the Gas costs are excessive, utilizing the protocol may not be economically viable. Therefore, reducing Gas consumption correlates with lowering the associated costs and improving the protocol's applicability in real-world scenarios. In our experiments, we measure the Gas consumption of submitting a batch of transactions along with the validity proof to update the current state root of the rollup contract and verify that the state changes are correct concerning the submitted proof.

The results (see Fig. 2) show that submitting an update for batch size 4 consumes 269,992 Gas on the source blockchain and 300,911 Gas on the target blockchain. Increasing the batch size to 128 results in a Gas consumption of 589,615 Gas on the source blockchain and 620,582 Gas on the target blockchain. Therefore, the total Gas consumption amounts to 1,210,197 Gas without cross-blockchain communication. The Gas consumption for verifying the proof is constant, while the increasing batch size increases the Gas consumption linearly for storing the transaction data. Further, we compare our protocol to an already existing solution using the protocol described in [10], which serves as an effective benchmark for our comparison. It requires 39,394 Gas for burning the tokens and 169,242 Gas for claiming the tokens, excluding the costs associated with cross-blockchain communication. In comparison, our protocol requires 4,606 Gas for burning tokens and 4,848 Gas for claiming tokens if the batch size is 128. These results represent a reduction in Gas consumption of 88% for burning tokens and 97% for claiming tokens. Further reductions are possible by increasing the batch size and compressing the transaction data. However, this leads to higher resource consumption by the operator when carrying out state changes and creating proof.

C. Performance

The proving time is essential as it influences the system's throughput. The primary factor that impacts the proving time is the complexity of the circuits. The complexity of a circuit depends on the underlying program that it represents. The more complex the calculations are, the larger the circuit. Hence, we examine the proving time for different batch sizes.

The results (see Fig. 3) show that generating a proof for the *Burn* circuit with batch size 4 takes, on average, 2076 ± 22 ms, while for the *Claim* circuit, it takes 2246 ± 7 ms. For batch size 128, the proof generation time for the *Burn* circuit is already $57,318 \pm 50$ ms and $38,898 \pm 31$ ms for the *Claim* circuit.

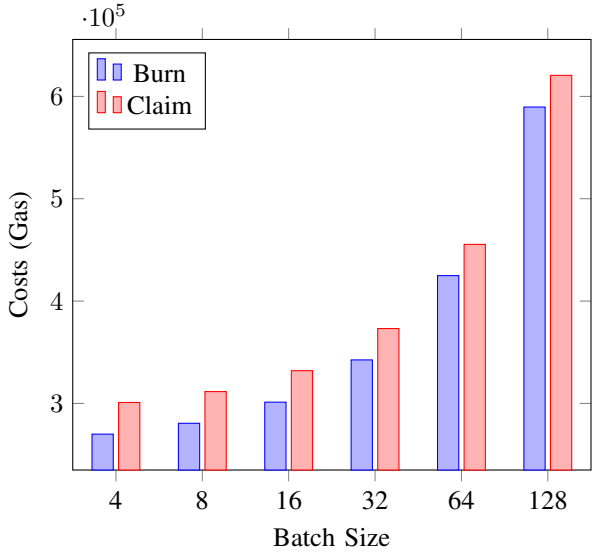

Fig. 2. Costs of submitting a batch.

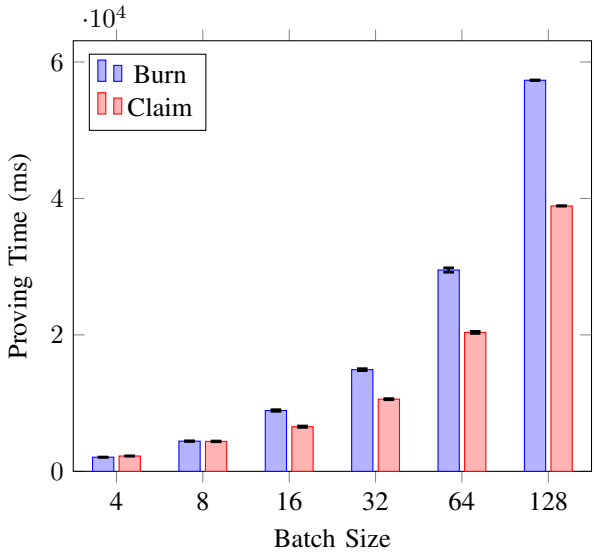

Fig. 3. Proving time for both circuits.

As expected, the proof generation time increases linearly with the batch size, and the *Burn* circuit has a higher proving time than the *Claim* circuit due to the increased circuit size from verifying the transaction signatures.

D. Memory

Memory usage is another critical factor when relying on off-chain computations. Generating zk-SNARKs requires many resources from the operator. As with the proving time, the memory consumption also depends on the complexity of the circuits. For that, we include the total allocated memory for the constraint system, proving and verification keys, witness generation, and generating the proof.

The results (see Fig. 4) show that generating a proof for the *Burn* circuit requires 413 MB and 465 MB for the *Claim*

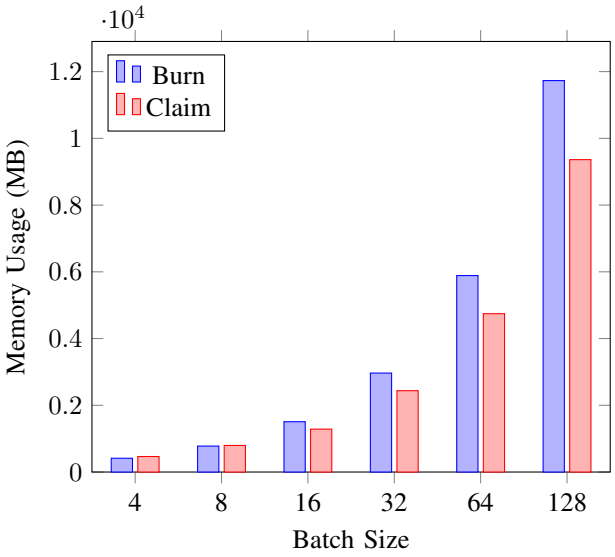

Fig. 4. Memory usage during proof generation for both circuits.

circuit, considering a batch size of 4 and increasing the batch size to 128 results in memory usage of $11,731$ MB for the *Burn* circuit and $9,362$ MB for the *Claim* circuit. These results are analog to the results of measuring the proving time. Therefore, it is also expected that generating the proof for the *Burn* circuit requires more memory due to the circuit's increased complexity compared to the *Claim* circuit. Further, the results show that operators need many resources to execute transactions and generate the proofs. Hence, resource usage constitutes a high entry barrier to running an operator, which results in lower decentralization as many rollups only use a single operator.

V. Related Work

In this section, we discuss other cross-blockchain token transfer protocols together with the differences to the proposed protocol.

The authors of [29] propose a decentralized cross-blockchain asset transfer protocol for arbitrary assets that guarantees finality. An asset transfer requires a burn step on the source blockchain and a claim step on the target blockchain. A user issues a burn transaction, including the recipient, the target blockchain, and the amount to the asset contract on the source blockchain, which burns the token. The protocol relies on a blockchain relay for cross-blockchain communication to enable the asset contract on the target blockchain to verify the burn transaction. After submitting a claim transaction to the target blockchain, the asset contract verifies the burn transaction and recreates the tokens. Additionally, the protocol applies an incentive mechanism to reward users for completing transfers to ensure finality. Sober et al. [10] extend the protocol with transfer confirmations on the source blockchain and apply notary schemes for cross-blockchain communication. Our novel protocol presented in the work at hand follows a similar mechanism but on Layer 2 instead of Layer 1, which reduces the Gas consumption at the cost of additional computational resources.

Similar to Sober et al. [10], the authors of [31] propose a cross-blockchain asset transfer protocol that follows a burn-to-claim approach. The protocol consists of an *exitTransaction*, which locks and destroys the tokens on the source blockchain, and an *entryTransaction*, which verifies the burn and recreates the tokens. However, blockchains must incorporate the protocol directly and operate gateway nodes to verify state proofs of the source blockchain. This characteristic requires blockchains to adopt their consensus mechanism and lowers the degree of decentralization since only a few nodes act as gateways. In comparison, our presented protocol does not require changes to the core protocol.

Karantias et al. [32] propose *Proof-of-Burn* as a primitive to allow the verification of destroying cryptocurrency. This primitive allows the transfer of tokens from a source blockchain to a target blockchain. In contrast to similar works [10], [29], [31], the authors provide a cryptographic definition and prove the scheme in the random oracle model. The protocol consists of two algorithms: *GenBurnAddr* and *BurnVerify*. The first creates a burn address with a specific tag, while the latter can verify if a given burn address encodes a particular tag. The tag usually encodes the target address of the token transfer. To verify the burn transaction, the authors suggest that miners monitor the source blockchain, use Non-Interactive Proofs-of-Proof-of-Work (NIPoPoWs), or apply notary schemes. Compared to our work, the authors focus more on defining a primitive and proving its security, not a full-fledged protocol.

Zendoo [33] is a sidechain construction that allows the creation of different sidechains to communicate with each other through a verifiable transfer mechanism based on zk-SNARKs. The system consists of a mainchain receiving cryptographic proofs from sidechains that are directly following the mainchain. A cross-chain transfer protocol enables transferring coins over multiple chains with two basic operations: forward and backward transfer. A forward transfer destroys tokens on the mainchain to send the tokens to a sidechain. Backward transfers allow batched transfers of tokens from a sidechain to the main chain by providing a withdrawal certificate using zk-SNARKs. Unfortunately, this solution only works for Bitcoin-like blockchains.

The authors of [34] propose a framework named *XClaim* for trustless cross-blockchain asset transfers. *XClaim* uses four different protocols for an asset transfer: Issue, Transfer, Swap, and Redeem. During the execution of these protocols, the assets are managed by a smart contract and locked by a vault on a backing blockchain to transfer and recreate the tokens on an issuing blockchain. Finally, the smart contract destroys the tokens on the backing blockchains after the receiver redeems the tokens. The framework applies blockchain relays for cross-chain state verification to provide data from the backing blockchain to the issuing blockchain. Further, *XClaim* uses collateral to incentivize honest behavior and punish misbehaving participants. Compared to our proposed protocol, *XClaim* operates on Layer 1.

In [35], the authors propose a cross-chain payment protocol with finality guarantees. The cross-chain payment protocol is formalized using Asynchronous Networks of Timed Automata (ANTA). Further, the authors show that it is impossible to solve this problem without synchrony. The authors demonstrate a solution to this problem in a partially synchronous setting, even with Byzantine failures. However, this protocol does not ensure atomicity by providing the ability to complete or roll back transactions in the case of misbehaving parties.

Niu et al. [36] propose a cross-blockchain Non-Fungible Token (NFT) transfer protocol based on a notary scheme. Initially, a sender sends an NFT to a notary smart contract. After that, the sender triggers the execution of a cross-blockchain transfer by calling the notary smart contract, which notifies the notary group. The notary group elects a leader who submits a transaction to the target blockchain to mint the NFT. After successfully minting the NFT on the target blockchain, the notary group calls the smart contract on the source blockchain to burn the locked NFT. Additionally, the notary group uses a reputation mechanism to ensure the election of a reliable leader and ensure the security of the approach. Compared to our protocol, this approach works on Layer 1 and focuses on NFTs.

The work by Borkowski et al. [37] proposes a deterministic cross-blockchain asset transfer protocol with eventual consistency. The sender and receiver of a token transfer need to create a proof of intent, which contains the necessary information to prove to the involved blockchains that the transfer is valid. The receiver issues a claim transaction containing the proof to publish the proof. Other participants can enter a witness contest to propagate this information to the remaining blockchains to earn a reward. The witness with the lowest signature value is considered the winner and receives the reward for propagating the information. Further, participants also need to veto conflicting proofs to prevent double-spending. Compared to our solution, this work requires replicating the same state across multiple blockchains.

The authors of [38] propose *CrossLedger*, a cross-blockchain asset transfer protocol for heterogenous blockchains. For that, the protocol selects an asset forwarder based on the blockchain's consensus mechanism, e.g., by selecting the previous miner. The asset forwarders are responsible for the execution and confirmation of the asset transfer. Each node of the respective blockchain network must verify the asset forwarder by creating a ring signature. Finally, the asset forwarders transfer and include the assets in the latest block of the target blockchain. Again, this approach requires changes to already existing blockchains.

In [39], the authors present *AucSwap*, a cross-blockchain asset transfer protocol based on the Vickrey auction model. The protocol consists of two parts: *Bidding Collection* and *Asset Exchange*. Initially, a sender wishing to transfer tokens publishes bidding information. After that, the sender collects from and distributes the bids to all participants to determine the receiver. After executing the auction, the sender and receiver execute an atomic swap to exchange the assets. While this approach allows for swapping assets, it does not enable transfers without a counterparty.

Further, there is a standardization effort from the Internet Engineering Task Force (IETF) creating the Secure Asset Transfer Protocol (SATP) for transferring digital assets between blockchains [40]. Marstein et al. [41] describe an implementation of SATP and the corresponding challenges. The protocol relies on gateway nodes that transfer assets between the different blockchains. An asset transfer consists of several steps: identity and asset verification, transfer initiation, lock assertion, and commitment preparation. The protocol also enables rollbacks in case of failures. However, the protocol mainly addresses private blockchains, while our proposed protocol targets arbitrary blockchains.

Additionally, blockchains of blockchains such as Cosmos [42] and Polkadot [43], designed with blockchain interoperability in mind, connect multiple blockchains through a central relay chain to ensure that different blockchains can communicate. Cosmos uses the Inter-Blockchain Communication Protocol (IBC) for cross-blockchain communication, while Polkadot uses the Cross-Consensus Message Format (XCM). These cross-blockchain communication protocols enable the implementation of cross-blockchain asset transfers in the respective networks.

As can be seen from the discussion, many different cross-blockchain asset transfer protocols exist. However, most protocols mentioned above heavily rely on Layer 1 of the involved blockchains, which incurs high costs. Further, these protocols usually neglect that the involved blockchains can fork, leading to possible inconsistencies they cannot resolve efficiently. The proposed solution, on the other hand, moves computation and state storage off-chain by using a rollup, which also allows for rollbacks to ensure consistency.

VI. Conclusion

While blockchain interoperability receives more attention from research and industry, it remains a significant issue in today's blockchain landscape. The efficient transfer of tokens from a source blockchain to a target blockchain can help bridge the gaps between different blockchain ecosystems. Unfortunately, current solutions still rely on Layer 1, which is not very cost-efficient.

Therefore, we propose a cross-blockchain token transfer protocol operating on Layer 2 based on rollups. By moving the burn and claim step of the cross-blockchain token transfer protocol off-chain, we can reduce the gas costs of executing transfers considerably while enabling the possibility of implementing rollbacks to account for possible forks. In future work, we want to adapt the protocol to include support for non-fungible tokens, explore different rollback mechanisms, and further optimize the costs.

Acknowledgment

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development as well as the Christian Doppler Research Association is gratefully acknowledged.

REFERENCES

[1] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] N. Vu, A. Ghadge, and M. Bourlakis, "Blockchain adoption in food supply chains: A review and implementation framework," *Production Planning & Control*, vol. 34, no. 6, pp. 506–523, 2023.

[4] E. R. D. Villarreal, J. García-Alonso, E. Moguel, and J. A. H. Alegría, "Blockchain for healthcare management systems: A survey on interoperability and security," *IEEE Access*, vol. 11, pp. 5629–5652, 2023.

[5] W. Issa, N. Moustafa, B. Turnbull, N. Sohrabi, and Z. Tari, "Blockchain-based federated learning for securing internet of things: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–43, 2023.

[6] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-two blockchain protocols," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 201–226.

[7] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. [Online]. Available: https://github.com/zcash/zips/blob/7292281012d505793fa7f171250dc062a9880d82/protocol/protocol.pdf

[8] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[9] S. Schulte, M. Sigwart, P. Frauenthaler, and M. Borkowski, "Towards blockchain interoperability," in *Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1–6, 2019, Proceedings 17*. Springer, 2019, pp. 3–10.

[10] M. Sober, M. Sigwart, P. Frauenthaler, C. Spanring, M. Kobelt, and S. Schulte, "Decentralized cross-blockchain asset transfers with transfer confirmation," *Cluster computing*, vol. 26, no. 4, pp. 2129–2146, 2023.

[11] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "SoK: Communication across distributed ledgers," in *25th International Conference on Financial Cryptography and Data Security, Revised Selected Papers, Part II*. Springer, 2021, pp. 3–36.

[12] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: review, comparison, and open research challenges," *IEEE Access*, vol. 8, pp. 85 675–85 685, 2020.

[13] M. Sober, G. Scaffino, C. Spanring, and S. Schulte, "A voting-based blockchain interoperability oracle," in *2021 IEEE International Conference on Blockchain*. IEEE, 2021, pp. 160–169.

[14] P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober, and S. Schulte, "ETH relay: A cost-efficient relay for ethereum-based blockchains," in *2020 IEEE International Conference on Blockchain*. IEEE, 2020, pp. 204–213.

[15] M. Westerkamp and M. Diez, "Verilay: A verifiable proof of stake chain relay," in *2022 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 2022, pp. 1–9.

[16] J. Kwon and E. Buchman. (2020) Cosmos whitepaper: A network of distributed ledgers. [Online]. Available: https://wikibitimg.fx994.com/attach/2020/12/16623142020/WBE16623142020_55300.pdf

[17] G. Wood. (2016) Polkadot: Vision for a heterogeneous multi-chain framework. [Online]. Available: https://www.win.tue.nl/~mholende/seminar/references/ethereum_polkadot.pdf

[18] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE access*, vol. 8, pp. 125 244–125 262, 2020.

[19] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer *et al.*, "On scaling decentralized blockchains: (a position paper)," in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[20] J. Ronis, "Understanding Ethereum's layer 1 and layer 2: Differences, adoption, and drawbacks," https://www.wilsoncenter.org/article/understanding-ethereums-layer-1-and-layer-2-differences-adoption-and-drawbacks, 2023, accessed 2024-01-10.

[21] Visa, "Annual report 2023," https://s29.q4cdn.com/385744025/files/doc_downloads/2023/Visa-Inc-Fiscal-2023-Annual-Report.pdf, 2023, accessed 2024-01-10.

[22] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.

[23] M. Labs, "zkporter: a breakthrough in l2 scaling," https://blog.matter-labs.io/zkporter-a-breakthrough-in-l2-scaling-ed5e48842fbf, 2021, accessed 2024-01-10.

[24] E. N. Tas and D. Boneh, "Cryptoeconomic security for data availability committees," in *International Conference on Financial Cryptography and Data Security*. Springer, 2023, pp. 310–326.

[25] V. Buterin, "An incomplete guide to rollups," https://vitalik.eth.limo/general/2021/01/05/rollup.html, 2021, accessed 2024-01-10.

[26] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1353–1370.

[27] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 781–796.

[28] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 305–326.

[29] M. Sigwart, P. Frauenthaler, C. Spanring, M. Sober, and S. Schulte, "Decentralized cross-blockchain asset transfers," in *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*. IEEE, 2021, pp. 34–41.

[30] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.9.0," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[31] B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "The burn-to-claim cross-blockchain asset transfer protocol," in *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2020, pp. 119–124.

[32] K. Karantias, A. Kiayias, and D. Zindros, "Proof-of-burn," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 2020, pp. 523–540.

[33] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: a zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains," in *40th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1257–1262.

[34] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. J. Knottenbelt, "XCLAIM: trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 193–210.

[35] R. van Glabbeek, V. Gramoli, and P. Tholoniat, "Cross-chain payment protocols with success guarantees," *Distributed Comput.*, vol. 36, no. 2, pp. 137–157, 2023.

[36] X. Niu, L. Kong, F. Jin, X. Song, X. Min, and Q. Li, "NFT cross-chain transfer method under the notary group scheme," in *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2023, pp. 996–1001.

[37] M. Borkowski, M. Sigwart, P. Frauenthaler, T. Hukkinen, and S. Schulte, "Dextt: Deterministic cross-blockchain token transfers," *IEEE Access*, vol. 7, pp. 111 030–111 042, 2019.

[38] L. Vishwakarma, A. Kumar, and D. Das, "Crossledger: A pioneer cross-chain asset transfer protocol," in *23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2023, Bangalore, India, May 1-4, 2023*. IEEE, 2023, pp. 568–578.

[39] W. Liu, H. Wu, T. Meng, R. Wang, Y. Wang, and C. Xu, "AucSwap: A vickrey auction modeled decentralized cross-blockchain asset transfer protocol," *J. Syst. Archit.*, vol. 117, p. 102102, 2021.

[40] T. Hardjono, M. Hargreaves, T. Hardjono, N. Smith, and V. Ramakrishna. (2023) Secure asset transfer protocol. draft specifications, Internet Engineering Task Force (IETF). [Online]. Available: https://datatracker.ietf.org/doc/pdf/draft-ietf-satp-architecture-01

[41] K. Marstein, A. Chiriac, L. Riley, T. Hardjono, and G. Verdian, "Implementing secure bridges: Learnings from the secure asset transfer protocol," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2023, Dubai, United Arab Emirates, May 1-5, 2023*. IEEE, 2023, pp. 1–9.

[42] J. Kwon and E. Buchman, "Cosmos a network of distributed ledgers," https://cosmos.network/resources/whitepaper, 2020, accessed 2023-12-12.

[43] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," https://assets.polkadot.network/Polkadot-whitepaper.pdf, 2016, accessed 2023-12-12.

13

Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs

Michael Sober, Giulia Scaffino, and Stefan Schulte: *Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs*. In *Distributed Ledger Technologies: Research and Practice*, volume 3 (4), 2024

DOI: 10.1145/3678187

Keywords: *blockchain, smart contracts, blockchain interoperability, blockchain oracles, cross-blockchain communication, zero-knowledge proofs*

Abstract. The closed architecture of prevailing blockchain systems renders the usage of this technology mostly infeasible for a wide range of real-world problems. Most blockchains trap users and applications in their isolated space without the possibility of cooperating or switching to other blockchains. Therefore, blockchains need additional mechanisms for seamless communication and arbitrary data exchange between each other and external systems. Unfortunately, current approaches for cross-blockchain communication are resource-intensive or require additional blockchains or tailored solutions depending on the applied consensus mechanisms of the connected blockchains. Therefore, we propose an oracle with an off-chain aggregation mechanism based on Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) to facilitate cross-blockchain communication. The oracle queries data from another blockchain and applies a rollup-like mechanism to move state and computation off-chain. The zkOracle contract only expects the transferred data, an updated state root, and proof of the correct execution of the aggregation mechanism. The proposed solution only requires constant 378 kgas to submit data on the Ethereum blockchain and is primarily independent of the underlying technology of the queried blockchains.

Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs

MICHAEL SOBER, Hamburg University of Technology, Hamburg, Germany and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Germany
GIULIA SCAFFINO, TU Wien, Vienna, Austria and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Austria
STEFAN SCHULTE, Hamburg University of Technology, Hamburg, Germany and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Germany

The closed architecture of prevailing blockchain systems renders the usage of this technology mostly infeasible for a wide range of real-world problems. Most blockchains trap users and applications in their isolated space without the possibility of cooperating or switching to other blockchains. Therefore, blockchains need additional mechanisms for seamless communication and arbitrary data exchange between each other and external systems. Unfortunately, current approaches for cross-blockchain communication are resource-intensive or require additional blockchains or tailored solutions depending on the applied consensus mechanisms of the connected blockchains. Therefore, we propose an oracle with an off-chain aggregation mechanism based on Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) to facilitate cross-blockchain communication. The oracle queries data from another blockchain and applies a rollup-like mechanism to move state and computation off-chain. The *zkOracle* contract only expects the transferred data, an updated state root, and proof of the correct execution of the aggregation mechanism. The proposed solution only requires constant 378 kgas to submit data on the Ethereum blockchain and is primarily independent of the underlying technology of the queried blockchains.

CCS Concepts: • **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Distributed algorithms**; • **Security and privacy** → **Cryptography**;

Additional Key Words and Phrases: Blockchain, smart contracts, blockchain interoperability, blockchain oracles, cross-blockchain communication, zero-knowledge proofs

ACM Reference format:
Michael Sober, Giulia Scaffino, and Stefan Schulte. 2024. Cross-Blockchain Communication Using Oracles With an Off-Chain Aggregation Mechanism Based on zk-SNARKs. *Distrib. Ledger Technol.* 3, 4, Article 27 (December 2024), 24 pages.
https://doi.org/10.1145/3678187

Authors' Contact Information: Michael Sober (corresponding author), Hamburg University of Technology, Hamburg, Germany and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg, Germany; e-mail: michael.sober@tuhh.de; Giulia Scaffino, TU Wien, Vienna, Austria and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Vienna, Austria; e-mail: giulia.scaffino@tuwien.ac.at; Stefan Schulte, Hamburg University of Technology, Hamburg, Germany and Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg, Germany; e-mail: stefan.schulte@tuhh.de.

ACM 2769-6480/2024/12-ART27
https://doi.org/10.1145/3678187

1 Introduction

Blockchains provide a distributed ledger of transactions while ensuring the integrity, consistency, and immutability of the transactions. Blockchain technology has been first introduced with Bitcoin [50] and is now considered the main backing technology of cryptocurrencies. However, it also found application in various other application areas like healthcare [2], supply chain management [55], and others [26, 27, 67]. The second generation of blockchains, like Ethereum [65], provides smart contracts, which are deterministic programs stored on the blockchain executed by every node of the network. The application of smart contracts allows the creation of **Decentralized Applications (dApps)** [23], opening an even wider range of possible application areas.

Different application domains of blockchain technology come with different challenges and requirements. A single blockchain can only fulfill some criteria, which results in a wide range of solutions tailored to particular application domains. These development efforts lead to a highly fragmented and heterogeneous blockchain space [58], as different blockchains often have unique consensus mechanisms, token standards, or smart contract capabilities. These broad differences make it difficult for different blockchains to work together and require special attention during development to ensure an interoperable blockchain landscape. Unfortunately, blockchain interoperability is a major issue often overseen and hinders blockchain technology from developing its full potential. Blockchains are usually considered closed systems that cannot interact with external systems. While exceptions exist, e.g., Hyperledger Cacti [30], Cosmos [45], Polkadot [66], and others [29], missing interoperability is still a widespread problem. Amongst other things, missing interoperability between blockchains leads to vendor lock-in, i.e., users cannot easily migrate to other blockchains [7].

Further, smart contracts can only manipulate their state on the hosting blockchain and cannot interact with smart contracts deployed on other blockchains or external systems. Therefore, there is a strong need for blockchain interoperability solutions to break the barrier between heterogeneous blockchain systems and enable cross-blockchain communication to securely transfer arbitrary state information between blockchains in a decentralized way. The possibility to transfer arbitrary information between blockchains enables the creation of decentralized cross-blockchain applications and protocols [51, 54, 57].

There is an ongoing effort in research and industry to create blockchain interoperability solutions [8, 62]. However, current solutions, like blockchain relays, are often very complex to implement since they depend on the source blockchains' consensus mechanism. A blockchain of blockchains solution has even more technical complexity, requiring an additional blockchain for cross-blockchain communication. On the other hand, notary schemes offer great flexibility and are mostly independent of the connected blockchains. Unfortunately, notary schemes exhibit a lower degree of decentralization [62]. Many notary schemes follow a decentralized approach where n out m parties must collaborate to distribute trust among multiple parties and reach an agreement, e.g., by applying a voting mechanism. These voting mechanisms require costly on-chain interactions since the smart contract has to process many votes to determine the outcome. More cost-efficient solutions use complex off-chain voting mechanisms, e.g., by applying threshold signatures, to submit an aggregated result [60]. However, these solutions require the additional execution of **Distributed Key Generation (DKG)** protocols, which introduce additional complexity and security risks [59].

Therefore, we propose a solution for cross-blockchain communication by using a decentralized oracle, i.e., a bridge to an external data source, with an off-chain aggregation mechanism based on **Zero-Knowledge Succinct Non-interactive Arguments of Knowledges (zk-SNARKs)**. The oracle allows its clients to query arbitrary information from other blockchains independent of the inner workings of the source blockchain. A committee of oracle nodes, divided into a single aggregator and multiple validators, executes an off-chain aggregation mechanism to attest to the correctness of the retrieved information. The validators query the source blockchain for the requested information and send the signed result to the current aggregator. The aggregator combines the results by generating a zk-SNARK to verify the signatures and distribute the rewards off-chain. Afterward, the aggregator submits the proof, the updated state, and the queried information to the smart contract. The smart

contract must only verify the proof, update the state, and store the retrieved data. Further, we provide a prototypical implementation for Ethereum-based blockchains and evaluate it concerning costs, memory, and performance. The results show that our oracle only requires 378 kgas to submit data on Ethereum-based blockchains. Furthermore, oracle nodes without many resources can also use the system and efficiently support a committee size of 256.

The remainder of this paper is structured as follows: Section 2 provides general information about the applied concepts. Afterward, Section 3 introduces the design of the oracle with its off-chain aggregation mechanism, and Section 4 follows with a security analysis. Section 5 provides implementation details, and Section 6 evaluates the proposed solution concerning costs, memory, and performance. After that, Section 7 follows with a discussion of related work. Finally, Section 8 concludes the paper.

2 Background

This section discusses the underlying concepts of the proposed solution. We provide the basics of cross-blockchain communication and follow with an explanation of **Zero-Knowledge Proofs (ZKPs)** and rollups.

2.1 Cross-Blockchain Communication

Cross-blockchain communication refers to the ability of two blockchains to reach a consistent state across both blockchains by synchronizing transaction execution [62]. Hence, cross-blockchain communication allows distinct blockchains to transfer arbitrary information, query the state of another blockchain [60] or allow more specific use cases such as atomic swaps [42] and cross-blockchain smart contract calls [51]. Unfortunately, cross-blockchain communication requires a trusted third party [70] since it is otherwise infeasible for two blockchains to verify each other's state. However, the ability to verify the state of other blockchains is a requirement to allow for interoperable blockchains to exist [46]. The trusted third party must attest to the correct execution of a transaction to another blockchain, allowing the interdependent execution of transactions on multiple blockchains. These trusted third parties can both be centralized or decentralized and most prominently include blockchain relays, notary schemes, and blockchains of blockchains [8].

A blockchain relay is a smart contract deployed on a target blockchain to replicate the source blockchain [8]. Off-chain clients continuously relay block headers from the source blockchain to the target blockchain while the smart contract enforces the consensus rules of the source blockchain to rebuild the chain. The off-chain clients relaying the block headers do not need to be trusted since the smart contract enforces the correct behavior. Further, the smart contract offers light client functionalities on the target blockchain, allowing smart contracts to query the state of the source blockchain. Typically, blockchains store a Merkle root of a block's transactions in the block header. Hence, the smart contract allows the execution of **Simplified Payment Verification (SPV)**, which allows checking if a particular transaction is part of a block while only having the block header. For that, clients submit a Merkle proof of inclusion to the relay contract to show that a transaction is part of a specific block. Further, the relay contract checks that the block containing the transaction is included in the source blockchain and considered final. Blockchain relays have a high degree of decentralization but are costly since on-chain block header verification and storage consume many resources. Additionally, off-chain clients must always keep the relay in sync with the source blockchain, which requires considerable maintenance effort. However, novel relay solutions already apply techniques to move computation off-chain by either optimistically accepting new block headers [31] or using ZKPs [64, 68] to achieve better cost efficiency.

Another technique for cross-blockchain communication is through the application of notary schemes [21]. A notary is a trusted entity or group responsible for executing certain operations across the boundaries of blockchains. The notary can monitor the involved blockchains, read the blockchains' state, and publish new transactions. Therefore, the notary can function as an oracle to query state information or publish new transactions on behalf of another blockchain. The blockchains interacting with the notary must only authenticate the notary to verify the integrity and authenticity of the executed operations. Notary schemes are a relatively simple solution

for cross-blockchain communication since they are independent of the underlying blockchains and offer great flexibility. However, notary schemes exhibit a lower degree of decentralization. Trusting only a single entity leads to a single point of failure, which can compromise the whole system. Therefore, many solutions require that n out of m entities agree by executing a voting mechanism, either on-chain or off-chain, to distribute the trust among multiple entities [60]. Further, these approaches require additional mechanisms to form a committee of trustworthy entities and reach an agreement.

Multiple blockchains can also communicate by following a blockchain of blockchains approach [62]. A blockchain of blockchains provides a platform of independent subchains to communicate with each other through a central relay chain. The main chain is responsible for recording the connected subchains' actions. Every subchain can access the main chain to verify the activities of other subchains to implement cross-blockchain communication. The subchains can work independently of the main chain [45] or adopt a shared security approach where validators of the main chain also validate the subchains [66]. The main chain is, to a certain extent, a notary scheme with the properties of a blockchain, as it attests to the actions of the subchains. However, in many cases, it also fulfills other purposes besides cross-blockchain communication. Unfortunately, there is also a considerable overhead to using an additional blockchain to keep track of all connected chains.

2.2 Zero-Knowledge Proofs

ZKPs were first introduced by Goldwasser et al. [37] to enable a prover to convince a verifier that the prover knows a witness to a particular statement without revealing any additional knowledge. For that, a ZKP system has to satisfy the following three properties [39]:

(1) *Completeness:* An honest prover can always convince an honest verifier that the prover knows a witness to a true statement.
(2) *Soundness:* A malicious prover cannot convince an honest verifier about the knowledge of a witness to a false statement.
(3) *Zero-knowledge:* A verifier does not learn additional information besides the statement's validity.

However, these proof systems require multiple interactions between the prover and the verifier to communicate the proof and convince the verifier. Therefore, Blum et al. [15] introduced non-interactive ZKP systems as an alternative to interactive proof systems. These non-interactive proof systems require an initial setup phase in which a prover and a verifier generate and share a CRS that replaces the previously necessary interactions. After that, a prover can generate a Non-interactive Zero-Knowledge Proof to convince the verifier, while the verifier does not have to send any messages to the prover. Further, Goldreich et al. [36] showed that all statements in $\mathcal{NP}$ have ZKPs, enabling the possibility of verifying arbitrary computations. Verifying computational integrity allows a verifier to check whether the output of a given program, given its inputs, is correct without re-executing the program itself [34, 53]. These properties are particularly advantageous in decentralized systems (e.g., blockchains) where provers cannot directly communicate with verifiers (e.g., smart contracts) or want to convince several verifiers.

Nowadays, ZKPs are heavily applied in blockchain technology to create scalable [61] and privacy-aware solutions [6]. Blockchains require every node in the network to execute and verify transactions which constitutes a major scalability issue. However, applying ZKPs makes it possible to move computations off-chain such that nodes only need to verify small proofs, reducing the overall network load and increasing throughput (see Section 2.3). Further, most blockchains only provide pseudonymity instead of anonymity, allowing anyone to link multiple addresses to the same identity by applying heuristics for address clustering [71]. With the application of ZKPs, it is possible to break the link between multiple addresses and create privacy-preserving blockchains [12, 43].

The main ZKP proof systems applied are zk-SNARKs [13], **Zero-Knowledge Succinct (Scalable) Transparent Arguments of Knowledge (zk-STARKs)** [11], and Bulletproofs [20]. These proof systems differ in their

underlying cryptographic assumptions, complexity, and the necessity of a trusted setup. The basic cryptographic assumptions behind zk-SNARKs are the hardness assumption of the elliptic curve discrete logarithm problem and the existence of secure bi-linear pairings. While the proving complexity of zk-SNARKs is quasi-linear, the verification and communication complexity is constant. Unfortunately, zk-SNARKs require a trusted setup phase to generate a **Common Reference String (CRS)**. The construction of zk-STARKs, on the other hand, is based upon collision-resistant hash functions, requiring no trusted setup and having quasi-linear proving complexity but a poly-logarithmic verification and communication complexity. Finally, Bulletproofs rely on the discrete log assumption and require no trusted setup but have quasi-linear proving complexity, linear verification complexity, and logarithmic communication complexity. In the work at hand, we use zk-SNARKs to take advantage of the constant verification and communication complexity, which is particularly beneficial for verifying proofs with smart contracts.

The first protocol enabling efficient zk-SNARK constructions was the Pinocchio protocol [13, 52]. After that, protocols like Gepetto [25] and Groth16 [39] further improved the efficiency of zk-SNARKs, and recent works introduced zk-SNARKs with a universal setup [24, 32, 49]. However, Groth16 is still heavily used in blockchain technology due to its small proof size and low verification complexity. Hence, we apply Groth16 to create the off-chain aggregation mechanism. The protocol allows the creation of zk-SNARKs for arithmetic circuits. For that, a program is translated into an arithmetic circuit, which is converted to a **Rank-1 Constraint-System (R1CS)**. Finally, the R1CS is converted to a **Quadratic Arithmetic Program (QAP)**, representing the program in a different format to generate the proof. The input to the program also yields the corresponding witness to the QAP. After that, the setup produces a CRS and a simulation trapdoor. The simulation trapdoor needs to be forgotten because it allows the creation of fake proofs. However, by using Multiparty Computation, it is possible to distribute the trust among multiple parties [17]. Finally, the algorithm can use the CRS to create pairing-based zk-SNARKs for the witness to the QAP.

2.3 Rollups

Blockchain technology faces a considerable scalability issue since blockchains cannot cope with the growing amount of work [72]. Research has already come up with various solutions, including improvements to Layer-1, i.e., the base layer of the protocol, and Layer-2 protocols, which build on top of a blockchain [40]. Rollups are a Layer-2 blockchain scaling solution to increase the throughput of blockchains by moving transaction computation and state off-chain [61]. Aggregators execute multiple transactions off-chain and only submit state changes and a bundle of compressed transaction data to the blockchain, i.e., the nodes on Layer-1 solely need to execute a single transaction. A smart contract manages the state root of the rollup and allows anyone to submit a bundle of compressed transactions, the previous and the new state root. The corresponding Merkle tree to the state root contains the account data (e.g., balance, nonce) of the different accounts that are part of the rollup. Users can deposit and withdraw funds directly by interacting with the smart contract.

Rollups have to enact specific rules to determine the validity of the submitted transaction data and the according state changes. Currently, two main types of rollups exist: optimistic and zk-rollups [22]. Optimistic rollups [44] assume that transactions are valid and allow users to submit fraud proofs [3] during a dispute period to prove a transaction is invalid. The smart contract verifies the proof and reverts the rollup to its previous state. With zk-rollups, on the other hand, users submit validity proofs (e.g., zk-SNARKs) to prove the correctness of the new state root resulting from the execution of the transactions in the bundle. Therefore, zk-rollups do not require a dispute period and enable instant transaction finality.

The availability of the compressed transaction data on Layer-1 ensures that anyone can recompute and verify the state of the rollup. While zk-rollups only require minimal transaction data on Layer-1 to ensure that anyone can recompute and verify the state, optimistic rollups require additional data to enable the verification of fraud proofs. Therefore, even with malicious actors in the system, rollups can provide censorship resistance and liveness.

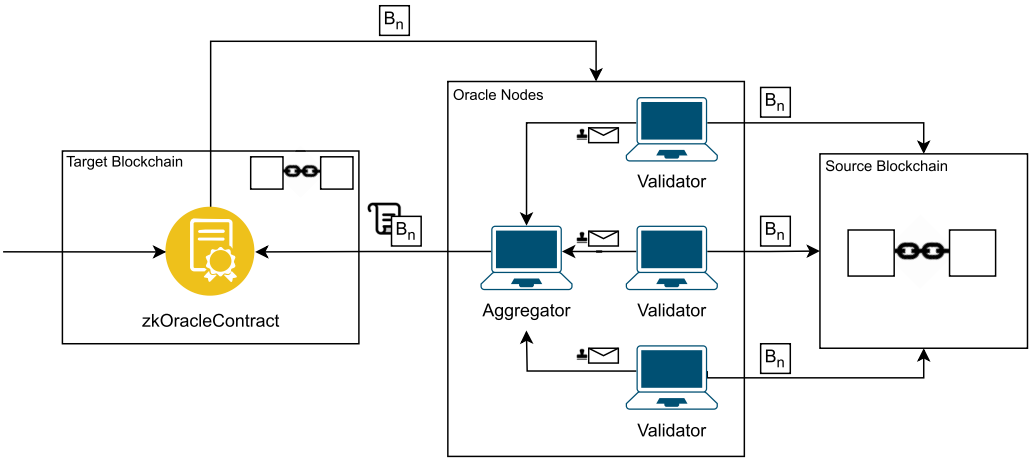

Fig. 1. Overview of the system.

Moving transaction computation and state off-chain is a promising approach to improve aggregation mechanisms that require many interactions with a smart contract or other tools (e.g., threshold signatures) to reduce on-chain computation. Hence, in the following, we apply the concept behind zk-rollups to create an off-chain aggregation mechanism.

3 System Design

In this section, we propose a cross-blockchain communication mechanism using oracles with an off-chain aggregation mechanism based on zk-SNARKs. Initially, we provide a short overview of the system. Afterward, we discuss dynamic participation and the applied incentive mechanism. Finally, we define the off-chain aggregation mechanism.

3.1 Overview

An oracle is a bridge between a blockchain and an external data source. The typical architecture for an oracle system consists of an on-chain component, e.g., a smart contract, and an off-chain component. The off-chain component retrieves data from external data sources, e.g., another blockchain, ensures the data's integrity, and submits the data to the oracle contract [41]. The oracle contract receives the data and serves it to data consumers. There are different types of oracles based on the data source, trust level, communication direction, and data type [4]. However, this work focuses on cross-chain oracles to enable blockchain interoperability and data transfer between distinct blockchains. These oracles represent a flexible and lightweight solution for cross-blockchain communication as they do not depend on the source blockchain's consensus mechanism and require no additional blockchain. Hence, the proposed oracle solution is compatible with different blockchains, assuming the connected blockchains support smart contracts.

The oracle (see Figure 1) allows arbitrary data transfer between two blockchains by enabling the oracle's clients to issue requests to query data from another blockchain. The system consists of the *zkOracle* contract deployed on a target blockchain and off-chain oracle nodes, which query data from a source blockchain and attest to the data's integrity. The *zkOracle* contract offers a mechanism for dynamic participation (see Section 3.2) that allows anyone to register and join a committee of oracle nodes. This committee collectively responds to requests by querying another blockchain and executing an off-chain aggregation mechanism (see Section 3.4) before returning the data. The oracle selects a single aggregator responsible for aggregating the results from the committee members. The

other committee members act as validators by querying the data from the source blockchain and attesting to the correctness. The aggregator collects a majority of answers with the same result and creates an aggregated result in a rollup-like fashion. For that, the aggregator generates a zk-SNARK proving that the majority voted for a specific answer and that the rewards are distributed correctly. By generating this proof, the oracle can execute the voting mechanism and reward distribution off-chain, while the smart contract only needs to verify the proof to ensure the correctness of these computations.

Additionally, the oracle applies an incentive mechanism (see Section 3.3) to reward honest behavior and punish misbehavior. While honest nodes participating in the aggregation protocol receive a constant reward to compensate them for providing resources, misbehaving nodes may get slashed and lose part or all of their deposited collateral. The aggregator slashes misbehaving oracle nodes by generating a zk-SNARK that the oracle node in question provided an answer that differs from the majority. Upon receiving the aggregated result, the *zkOracle* contract only verifies the zk-SNARK, stores the result, and updates the Merkle root containing the state of all oracle nodes. Afterward, any other smart contract can access the data retrieved from the source blockchain by calling the *zkOracle* contract.

3.2 Dynamic Participation

The oracle relies on a committee of oracle nodes with a maximum size of n nodes to collectively return the result to a query. However, the oracle can already operate with less than n nodes as long as $\frac{n}{2} + 1$ nodes are honest. For that, the system provides a mechanism for dynamic participation, allowing anyone to register oracle nodes and participate in transferring data between two blockchains. The mechanism enables any oracle node to join or leave the system by calling the *zkOracle* contract deployed on the target blockchain. Further, the mechanism provides high Sybil resistance by following a **Proof of Stake (PoS)** mechanism to select the committee. While other mechanisms like using a pre-defined set of participants or a first come, first served approach are also possible, they suffer from Sybil attacks [28] and a lower degree of decentralization.

Initially, the *zkOracle* contract has no registered oracle nodes. Anyone can call the *zkOracle* contract and register an oracle node providing a public key, IP address, and stake in the blockchain's native currency. The *zkOracle* contract uses a Merkle Tree to store the accounts of registered oracle nodes consisting of an address, index, public key, and the stake balance. Therefore, the *zkOracle* contract does not store raw account data, and the off-chain aggregation protocol can update the Merkle root to apply state changes, resulting in considerable cost savings. The *zkOracle* contract only allows up to n oracle nodes providing a minimum stake to register. After the registration of n oracle nodes, it is only possible for a new oracle node to join if it provides a higher stake than the oracle node it wishes to replace. The *zkOracle* contract enables anyone to provide a public key, a higher stake than a specific oracle node, and the respective Merkle proof to replace the node. With this approach, the *zkOracle* contract ensures that the committee of oracle nodes always consists of the n nodes with the highest deposited stake.

Additionally, registered oracle nodes can also leave the system. Leaving the system requires two steps: exit and withdraw. First, an oracle node calls the *zkOracle* contract to announce that it wants to exit the system by providing the necessary account details with the respective Merkle proof. The *zkOracle* contract ensures that the caller is the rightful owner of the account, the validity of the Merkle proof, and computes the time an oracle node has to wait until it is allowed to withdraw. The exit time ensures that users with a high enough stake cannot repeatedly remove other oracle nodes with a lower stake from the system by replacing the node and immediately leaving the system to replace other nodes. Without the exit time, anyone with a high enough stake could remove all nodes with a lower stake and replace these oracle nodes with new oracle nodes that do not require a high stake. After reaching the exit time, an oracle node can provide its account details with the corresponding Merkle proof to withdraw its stake from the *zkOracle* contract. Finally, the *zkOracle* contract transfers the stake and replaces the account in the Merkle tree with an empty account.

3.3 Incentive Mechanism

The proposed oracle solution is a public system allowing anyone to register oracle nodes cooperating to transfer data between blockchains. However, without an incentive, there is no reason for anyone to participate in the system since participation requires resources. Therefore, the oracle applies an incentive mechanism that encourages participation and honest behavior and discourages malicious behavior. For that, every oracle node, whether acting as an aggregator or a validator, receives a monetary reward for submitting a result to the *zkOracle* contract. On the other hand, the *zkOracle* contract punishes misbehaving nodes by slashing the deposited stake.

The aggregator and validators receive a monetary reward for each submission. This reward can be constant, or the *zkOracle* contract can dynamically define the amount. However, dynamically determining the reward requires additional public inputs to the aggregation circuit, which increases the verification costs. The oracle distributes these rewards during the off-chain aggregation mechanism. The aggregator increases its balance and the balance of every validator submitting a valid result and signature by the defined reward. The rewards of the aggregator need to be higher than the validator rewards since the aggregator has to provide a lot more computational power to generate the zk-SNARK during the execution of the off-chain aggregation mechanism and also has to pay the transaction fees for submitting the final result to the *zkOracle* contract.

The aggregator assigns the rewards during off-chain aggregation and proves that it assigned the correct rewards to the involved participants with the generated zk-SNARK. The *zkOracle* contract only needs to verify the proof and store the updated Merkle root, which contains the account information with the updated balances. Therefore, the system provides a fair and cost-efficient approach to reward every participant without relying on multiple on-chain interactions or relying on game theoretical approaches to distribute the rewards.

An aggregator selects only a majority of responses to create aggregated results. This selection process allows it to potentially exclude a minority of validators from receiving rewards. As a result, resources are wasted even though the validators' tasks are not very resource-intensive. Nevertheless, the oracle eventually appoints each validator as an aggregator, enabling them to earn rewards. This mechanism results in a tradeoff between the fees clients pay to use the oracle and the fair distribution of rewards among validators since clients must pay for each validator submitting an answer.

3.4 Off-Chain Aggregation

The off-chain aggregation mechanism (see Figure 2) allows the oracle nodes to return aggregated results to the *zkOracle* contract while reducing the number of on-chain operations to a minimum. During the execution of this mechanism, a majority of oracle nodes query data from another blockchain and have to agree on the result. The system follows the same principle as blockchain rollups by moving computation and state storage off-chain. However, instead of rolling up transactions, the system bundles votes on queried data from another blockchain. The aggregator verifies the votes and changes the balances of committee members based on their participation during the execution of the aggregation mechanism. The aggregator only submits a zk-SNARK, the query result, and the updated Merkle tree with the account information to the *zkOracle* contract. Hence, the aggregation mechanism requires only minimal computational and storage resources from the blockchain, reducing the overall costs while maintaining decentralization.

Initially, a client of the *zkOracle* contract has to issue a request by calling the *getBlockByNumber* function of the *zkOracle* contract (Step 1) providing the block number of a requested block and the necessary rewards for the oracle nodes executing this request. The *zkOracle* contract assigns the request a unique ID and stores it. Depending on the blockchain, the oracle nodes have to query the blockchain to check for any pending requests continuously, or the blockchain emits an event to notify the oracle nodes (Step 2).

After getting the information about a pending request, the oracle nodes use the *getAggregator* function of the *zkOracle* contract to determine their current role. The *zkOracle* contract supports different approaches for selecting the current aggregator. An efficient strategy is to adopt a round-robin method for organizing each

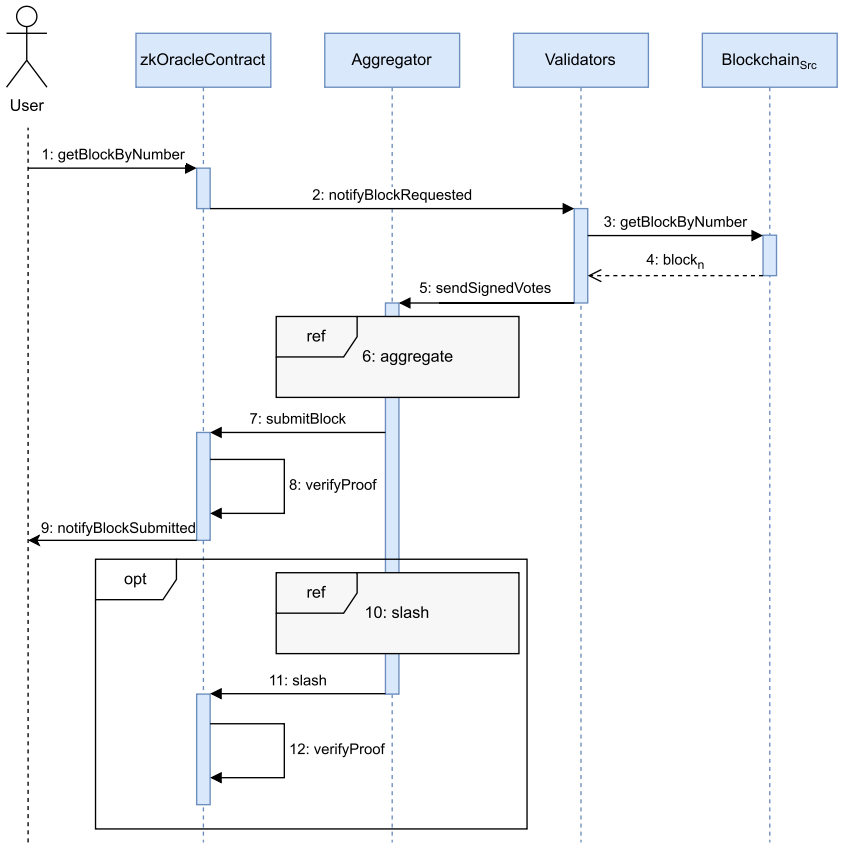

Fig. 2. Sequence of interactions for retrieving data.

oracle node to serve as an aggregator for a specific time interval until switching to the next aggregator. This approach prevents races between multiple aggregators for aggregating results, which wastes resources and incurs additional costs for interacting with the *zkOracle* contract since only one submission would be successful. Another possibility would be for the *zkOracle* contract to randomly choose the next aggregator, making it harder for adversaries to attack specific aggregators upfront. However, this approach requires the computation of a random seed using an additional secret input for the aggregation mechanism and a timeout to account for malicious aggregators not providing an answer. In this case, the aggregator rotates after every block submission or after reaching the timeout.

The validators query the source blockchain (Steps 3 and 4) to retrieve the block with the specified number and check if the block is final. Therefore, the validators need to check that such a block exists and that at least n blocks confirm it in the case of blockchains with probabilistic finality. For blockchains with immediate finality, waiting for n confirmations is unnecessary. Ignoring the finality of blocks can result in validators returning a wrong result due to possible forks in the source blockchain. If a validator determines that such a block does not exist or is not final, it responds with a zero value as the block hash. A validator creates a vote consisting of the validator's ID, the request number, and the hash of the retrieved block. After that, a validator signs the vote using its private key and retrieves the current aggregator's IP address from the *zkOracle* contract. Hence, an aggregator cannot forge an invalid vote. A validator establishes a direct channel with the aggregator and sends the vote with the according signature to the aggregator (Step 5).

Algorithm 1: Aggregation Circuit

```
1:  function AGGREGATE(preStateRoot, postStateRoot, blockHash, request, validatorBits, a, validators)
2:      for i < numAccounts do
3:          for j < numAccounts do
4:              if i = j then
5:                  continue
6:              end if
7:              assert(validators[i].index ≠ validators[j].index)
8:          end for
9:      end for
10:     verifyMerkleProof(preStateRoot, a.MerkleProof)
11:     a.balance ← a.balance + reward
12:     a.MerkleProof[0] ← hash(a.index, a.publicKeyX, a.publicKeyY, a.balance)
13:     intermediateRoot ← computeRootFromPath(a.MerkleProof)
14:     actualValidatorBits ← 0
15:     for all v ∈ validators do
16:         verifyMerkleProof(intermediateRoot, v.MerkleProof)
17:         msg ← hash(v.index, request, blockHash)
18:         verifySignature(v.signature, v.publicKey, msg)
19:         assert(v.blockHash == blockHash)
20:         v.balance ← v.balance + reward
21:         actualValidatorBits ← actualValidatorBits + 2^v.index
22:         v.MerkleProof[0] ← hash(v.index, v.publicKeyX, v.publicKeyY, v.balance)
23:         intermediateRoot ← computeRootFromPath(v.MerkleProof)
24:     end for
25:     assert(actualValidatorBits = validatorBits)
26:     assert(intermediateRoot = postStateRoot)
27: end function
```

Upon receiving a vote from a validator, the aggregator verifies the vote and stores it in the aggregator's mempool. The aggregator considers a vote to be valid if the signature is valid. After receiving a majority of valid votes with the same result for a specific request, the aggregator starts the aggregation process. During this process, the aggregator has to execute the aggregation mechanism locally and construct the witness for the aggregation circuit to generate the zk-SNARK, proving that the submitted result and applied state changes are correct (Step 6). The zk-SNARK ensures that an aggregator can only perform the correct aggregation algorithm based on the votes received from the validators and submit the majority answer.

The aggregator executes a program (see Algorithm 1) compiled into an arithmetic circuit for zk-SNARK construction to aggregate the votes. The aggregation circuit expects the previous state root, the resulting state root, the expected block hash, the request number, and the validator bit vector as public inputs. The aggregation circuit also requires private inputs, including the account information of the aggregator and validators with the respective Merkle proofs. Additionally, the private inputs contain the signatures and returned block hashes from the validators.

Initially, the aggregator must ensure no multiple votes from the same validator. For each request, a validator only has one vote. For that, the aggregator iterates over all submitted votes and checks that the validator index does not match another vote's validator index (Lines 2–9). After that, the aggregator checks that the Merkle tree includes the account of the aggregator (Line 10) and increases its balance by the specified reward (Line 11).

After increasing the balance, the aggregator computes an intermediate state root reflecting the change in the aggregator's balance (Lines 12–13). Subsequently, the aggregator processes the submitted votes from the validators. For each vote, the aggregator checks if the Merkle tree includes the validator's account (Line 16) and hashes the vote to construct the message (Line 17). Then, the aggregator verifies the signature provided by the validator by using the validator's public key and the message (Line 18). After verifying that the validator is part of the system and provided an authenticated vote, the aggregator checks if the block hash provided by the validator matches the expected block hash provided by the majority of validators (Line 19).

The aggregator also adds the reward to the validator's balance (Line 20) and updates a bit vector indicating which validators voted for the result to ensure data availability. A bit value of 1 at a specific position shows that the validator with this index contributed to the result. In contrast, a bit value of 0 shows that the respective validator did not participate. The aggregator flips the bits at the respective position by adding $2^{v.index}$ to the validatorBits (Line 21). Otherwise, the other oracle nodes cannot synchronize the state without knowing which state changes are necessary since the aggregator only submits the updated Merkle root. Then, the aggregator computes a new intermediate root reflecting the state changes for the validator's account (Lines 22–23). Finally, after repeating this process for each validator, the aggregator checks if the validatorBits match the provided validatorBits (Line 25) and if the intermediate root equals the provided post-state root (Line 26).

After creating the witness and the public inputs for the aggregation circuit, the aggregator generates the zk-SNARK to submit the aggregated result. The aggregator calls the *submitBlock* function of the *zkOracle* contract (Step 7), providing the request number, block hash, validators bit vector, updated state root, and zk-SNARK. Initially, the *zkOracle* contract verifies that the transaction sender is the current aggregator, and the request is still pending. After that, the *zkOracle* contract verifies the submitted zk-SNARK (Step 8), updates the state root, and stores the block hash. Finally, the *zkOracle* contract notifies the client about the submission (Step 9), and clients can query the *zkOracle* contract for the block hash to use it for SPV.

After successfully submitting an aggregated result to the *zkOracle* contract, the aggregator executes a program (see Algorithm 2) to process invalid votes and slash misbehaving validators (Step 10). The aggregator considers a vote invalid if the hash of the retrieved block differs from the majority answer to a certain request of the other validators. The slashing circuit expects the same inputs as the aggregation circuit, except that the slashing circuit only expects a single validator account with the submitted signature and block hash. Further, the indices of the aggregator and the validator must be public inputs to ensure data availability.

Initially, the aggregator verifies the inclusion of the validator's account in the Merkle tree (Line 2). After that, the aggregator computes the message (Line 3) and verifies the signature provided by the validator (Line 4). In the next step, the aggregator sets the validator's balance to 0 (Line 5) and computes the new state root (Lines 6–7). The aggregator verifies the inclusion of the aggregator's account in the Merkle tree (Line 8), adds the deducted balance to its account (Line 9), and also computes the new state root (Lines 10–11). Further, the aggregator checks that the block hash provided by the validator is different from the block hash provided by the majority of validators (Line 12) and that the computed state root matches the provided post-state root (Line 13). Finally, the aggregator calls the *slash* function of the *zkOracle* contract (Step 11) and provides the proof, the aggregator and validator indices, the request number, and the post-state root. The *zkOracle* contract verifies that the request is not pending, verifies the proof, and stores the updated state root (Step 12).

4 Security Analysis

This section provides a security analysis of our oracle solution. Initially, we clarify assumptions and the adversarial model. Then, we define the required security properties and show that our solution fulfills all these properties. Finally, we discuss Sybil-resistance and the adversarial bound.

Cryptographic Assumptions. We assume that zk-SNARKs have perfect completeness, soundness, and zero-knowledge [39]. Further, we consider hash functions modeled as random oracles [9] and digital signature schemes having Existential Unforgeability under Chosen Message Attack security [38].

Algorithm 2: Slashing Circuit

```
1:  function SLASH(preStateRoot, postStateRoot, blockHash, request, a, v)
2:      verifyMerkleProof(preStateRoot, v.MerkleProof)
3:      msg ← hash(v.index, request, blockHash)
4:      verifySignature(v.signature, v.publicKey, msg)
5:      v.balance ← 0
6:      v.MerkleProof[0] ← hash(v.index, v.publicKeyX, v.publicKeyY, v.balance)
7:      intermediateRoot ← computeRootFromPath(v.MerkleProof)
8:      verifyMerkleProof(intermediateRoot, a.MerkleProof)
9:      a.balance ← a.balance + v.balance
10:     a.MerkleProof[0] ← hash(a.index, a.publicKeyX, a.publicKeyY, a.balance)
11:     intermediateRoot ← computeRootFromPath(a.MerkleProof)
12:     assert(blockHash ≠ v.blockHash)
13:     assert(postStateRoot = intermediateRoot)
14: end function
```

System Assumptions. We assume that, at any point in time, there is an honest majority of oracle nodes. Further, we assume that the communicating blockchains provide safety and liveness guarantees.

Adversarial Model and Security Properties. We assume a computationally-bound static adversary $\mathcal{A}$ that can corrupt a fixed number of oracle nodes prior to the execution of the protocol. However, depending on the committee's maximum size, $\mathcal{A}$ can only corrupt a minority of oracle nodes. The oracle possesses the following security properties:

Definition 1 (Oracle Safety). An oracle is safe if, for any request made on a destination chain bc_{dest} by a client c with respect to a source chain bc_{src}, it provides a correct answer to bc_{dest}. A correct answer to bc_{dest} includes the requested data included and confirmed with overwhelming probability in bc_{src}.

Definition 2 (Oracle Liveness). An oracle is live if, for any request made on a destination chain bc_{dest} by a client c with respect to a source chain bc_{src}, it eventually provides an answer to bc_{dest}.

THEOREM 4.1. *Our zkOracle is secure, i.e., it is safe and live.*

PROOF. Toward contradiction, suppose our zkOracle is not secure. For this to happen, it means that there was a point in time where (i) the oracle provided bc_{dest} with an incorrect answer, or (ii) the oracle did not provide an answer to a request.

In case (i), for the oracle to provide an incorrect answer, it means that either a majority of nodes were adversarial, i.e., they voted on an incorrect answer, or that the properties of the cryptographic primitives used by the oracle have been broken, i.e., the aggregator successfully forged an invalid proof or validator signatures, or that either bc_{src} or bc_{dest} are not secure. This contradicts our cryptographic and system assumptions.

In case (ii), for the oracle to not provide an answer, it means that either none of the aggregators ever submits a valid answer, not enough signatures are produced by the nodes in the committee, or that either bc_{src} or bc_{dest} are not secure. This contradicts our system assumptions. □

Sybil-Resistance and Adversarial Bound. We note that the staking requirement for nodes joining the oracle provides sybil-resistance as well as the lower bound on the stake for the adversary to pull an attack against the oracle successfully.

The oracle requires each oracle node o_i joining the committee C with a maximum size of n nodes to deposit a stake $s_i \geq s_{min}$ where s_{min} is the configurable minimum stake required to join C. The amount depends on

the number of registered oracle nodes $|C|$. Hence, if $|C| < n$, the committee is not full and $s_i \geq s_{min}$. However, if $|C| = n$, i.e., the committee is already full, o_i can only join C by replacing another node $o_j \in C$ if $s_i > s_j$. For the oracle to submit an answer, it requires a majority $t = \frac{n}{2} + 1$ of oracle nodes to attest to the correctness of the transferred data and generate a valid proof. Hence, an adversary $\mathcal{A}$ needs to have more stake than the majority of registered oracle nodes with the lowest deposited stake $C_P = min_t(C)$ which have a cumulative stake $s_P = \sum_{o_i \in C_P} s_i$. Therefore, s_P is a lower bound on the required adversary's stake $s_{\mathcal{A}}$, i.e., $s_{\mathcal{A}} > s_P$ to break the oracle's security successfully.

The requirement for oracle nodes to deposit a substantial stake not only increases their investment in the system's integrity but also elevates the economic cost of attempting to join C with multiple identities. This requirement ensures nodes are incentivized to maintain their position within the committee, as losing their place means forfeiting potential future rewards. Hence, this mechanism effectively raises the barriers against Sybil attacks, safeguarding the system's decentralization and security.

5 Implementation

In this section, we discuss the implementation details of the different smart contracts and the node software. Further, we provide the prototypical implementation as open-source software directly available on GitHub.[1]

5.1 Smart Contracts

We implemented the smart contracts for the Ethereum smart contract and dApp platform using Solidity. There are several reasons for using Ethereum to host the smart contracts of the proposed solution. Ethereum is currently the largest smart contract platform and, therefore, also comes with a rich ecosystem and huge community. Further, Ethereum provides the necessary smart contract capabilities and pre-compiled contracts, i.e., more efficient contracts, since they do not incur additional overhead using the **Ethereum Virtual Machine (EVM)**. These pre-compiled contracts are part of the Ethereum client software and allow, among other things, the efficient verification of zk-SNARKs using the *altbn128* curve, which is a necessary prerequisite to implement the proposed solution. Many blockchains also use the EVM and support the implemented smart contracts without requiring changes. These reasons render Ethereum the most viable solution for implementing the prototype. Further, porting or implementing the solution to other smart contract platforms that do not use the EVM is also possible. However, these platforms must bring the necessary smart contract capabilities and allow efficient elliptic curve pairing checks.

The *zkOracle* contract inherits an additional smart contract implementing a sparse Merkle tree to store the accounts since inserting new data are easier. The *zkOracle* contract applies MiMC [5] for hashing the data, an arithmetization-friendly hash function optimized for modular arithmetic in a finite field. Therefore, the oracle nodes can efficiently compute all operations regarding the Merkle tree inside an arithmetic circuit. Using arithmetization-friendly hash functions is necessary to minimize resource consumption during proof generation. Further, the oracle randomly chooses the next aggregator by adding a random elliptic curve point as an additional public input to the aggregation circuit. The aggregator computes the next seed during the aggregation mechanism by multiplying the seed with its private key. The *zkOracle* contract uses an exit time of seven days to ensure misbehaving clients cannot arbitrarily remove other oracle nodes from the committee. For that, the *zkOracle* contract retrieves the current block timestamp and adds a seven-day delay. For the verification of the zk-SNARKs, we use two additional verifier contracts, which are automatically generated with the help of the *gnark* zk-SNARK library [16], also used by the provided client software. These generated contracts use a pre-compiled contract by Ethereum to compute elliptic curve pairings to verify the submitted proofs.

[1] https://github.com/soberm/zkOracle

5.2 Client

We implemented the client software for the oracle nodes using the Go[2] programming language. The client software uses automatically generated contract bindings to interact with the smart contracts to avoid boilerplate code and to enable seamless interaction. For communication, the oracle nodes use gRPC[3] to establish point-to-point channels, with the current aggregator offering an RPC API for validators to submit their votes. The client software allows each oracle node to act as an aggregator and a validator.

Each oracle node has to store and continuously synchronize the state of the *zkOracle* contract. For that, an oracle node stores the raw state in memory and listens to all events that indicate state changes, i.e., changes to the accounts stored in the Merkle tree. These events include registering, replacing, and withdrawing oracle nodes and submitting results or validator slashing. The oracle node needs to execute the state changes in the correct order and to update its locally stored state. Newly joined oracle nodes synchronize their state by starting from the beginning, i.e., the deployment time of the *zkOracle* contract and executing all state changes based on the persisted events. Further, oracle nodes that went offline must synchronize their state from the last known state change. Optimizations are still possible, whereby oracle nodes download the state of already synchronized oracle nodes and verify that the root matches the root of the *zkOracle* contract.

The validators listen for *BlockRequested* events on the target blockchain to start the validation process. These events include a request number and the corresponding block number. After receiving a *BlockRequested* event, a validator uses the JSON-RPC client to connect to an Ethereum node and retrieve the block with the given block number. Then, a validator uses a predefined threshold to determine the finality of the retrieved block. Afterward, a validator creates the vote and signs it using **Edwards-curve Digital Signature Algorithm (EdDSA)**. The applied EdDSA does not use *Curve25519* [14] as the aggregator verifies the signature in an arithmetic circuit using modular arithmetic over a different field. Therefore, the validator uses EdDSA with a twisted Edwards curve defined over the same field as the arithmetic circuit [10]. Further, a validator uses the MiMC hash function to create the message to be signed. Finally, the validator calls the *zkOracle* contract to retrieve the current aggregator with its IP address and sends the signed vote to the current aggregator using gRPC. While this implementation only supports Ethereum-based source blockchains, it is easily possible to support other blockchains by simply switching the client of the source blockchain and defining a new strategy to determine the finality of a block.

The aggregator receives votes from the validators and stores the votes in its in-memory mempool until reaching a majority of votes with the same result. Afterward, the aggregator fetches the state and constructs the witness for the aggregation circuit created with the gnark zk-SNARKs library. To generate the proof, the aggregator uses the Groth16 proving scheme and transforms the proof into the respective format for the verifier contract.

6 Evaluation

After providing the implementation details, we use the prototypical implementation to evaluate the proposed solution. Initially, we conduct several experiments to measure the costs of using smart contracts. Afterward, we examine the performance and memory requirements for generating the proofs.

6.1 Experimental Setup

We perform the experiments on a machine equipped with an Intel Core i7-10510U CPU running Ubuntu 20.04.6 using 8 cores with a clock speed of 1.80 GHz. The machine has 16GB LPDDR3 RAM with a frequency of 2,133 MHz and a 2 TB SSD. We deploy the smart contracts on a local Ethereum blockchain using HardHat Network 2.11.1. After deploying the smart contracts, we execute each function of the *zkOracle* contract to measure the gas costs. To measure the proof generation time and memory consumption, we simulate a committee of oracle nodes responding to requests. After every 50th measurement, we double the committee size and repeat the experiment

[2] https://go.dev/
[3] https://grpc.io/

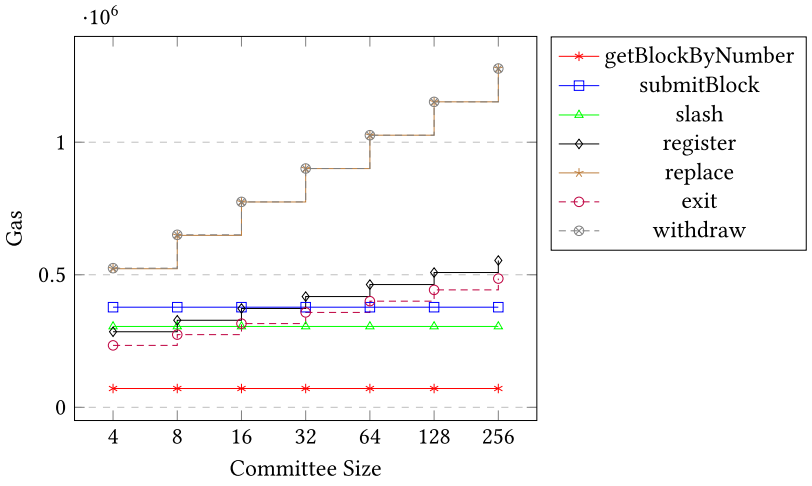

Fig. 3. Gas consumption of the *zkOracle* contract's functions.

until the committee size is 256. The program's source code for simulating the oracle nodes and measuring the proof generation time and memory consumption is again available on GitHub.

6.2 Costs

The Ethereum network uses gas as a unit of measurement to quantify the computational effort and storage space for executing transactions. Every user issuing a transaction has to pay transaction fees using **Ether (ETH)** based on the amount of gas spent to execute the transaction. This mechanism serves as the basis for incentivizing block producers and validators to spend resources on operating and securing the network. It also helps to manage and protect the network's resources from misuse. The costs are an essential aspect to pay attention to when designing and developing dApps since the costs determine a considerable part of the practical applicability of a solution since users have to pay for every interaction and reason whether the benefits outweigh the costs. Therefore, we examine the gas consumption of the *register*, *replace*, *exit*, *withdraw*, *getBlockByNumber*, *submitBlock*, and *slash* functions implemented by the *zkOracle* contract.

The results of our experiments (see Figure 3) show that the *getBlockByNumber*, *submitBlock*, and *slash* functions have a constant gas consumption. The *getBlockByNumber* function consumes 70,914 gas, the *submitBlock* function 377,673 gas, and the *slash* function 305,024 gas. The *submitBlock* function has low and constant gas consumption since its only task is to store the request and notify the oracle nodes. The *submitBlock* and *slash* functions consume more gas since these functions need to verify zk-SNARKs, including the computation of elliptic curve pairings. However, due to the application of zk-SNARKs, the gas consumption for verifying the correct execution of the off-chain aggregation and slashing mechanisms is low and independent of the committee size.

The remaining functions of the *zkOracle* contract to manage the committee of oracle nodes exhibit an increasing gas consumption depending on the height of the Merkle tree storing the accounts. The gas consumption for these functions remains nearly constant, with negligible variations due to the EVM's memory and storage management, until the height of the Merkle tree increases. The *register* function consumes 285,008 gas, the *replace* function 522,704 gas, the *exit* function 233,602 gas, and the *withdraw* function 524,883 gas considering a committee size of 4. Increasing the committee size to 256 leads to a gas consumption of 553,940 gas for the *register* function, 1,278,273 gas for the *replace* function, 485,572 gas for the *exit* function, and 1,277,974 gas for the *withdraw* function. The *replace* and *withdraw* functions have nearly the same gas consumption, as withdrawing is a replacement with an empty account and an additional transfer of funds to the account owner. The gas consumption of these operations

can get quite high since the number of expensive hash operations increases. However, these operations are rarely executed and only needed to change the committee of oracle nodes.

Compared to another oracle solution, e.g., the oracle in [60], which consumes 257,602 gas ± 21,671 gas for block submissions, the proposed solution consumes more gas. However, it does not require executing a DKG protocol and enables equal reward distribution. Further, the proposed solution incurs fewer costs than a relay solution, e.g., ETH relay [31], which consumes 284,041 gas for each block header submission. The submission costs of ETH relay may be lower, but the relay requires off-chain clients to relay each block of the source blockchain, while the proposed oracle solution only retrieves blocks on-demand. The **Off-Chain Reporting (OCR)** protocol [19] used by Chainlink's price feed oracles consumes around 291 kgas for the first submission using 31 oracle nodes. However, these costs increase with the number of oracle nodes, while our solution provides constant submission costs. Another comparison with zkBridge [68] shows that while zkBridge only consumes 220 kgas for on-chain verification, it requires tailored solutions based on the connected blockchains, while our solution is mostly blockchain agnostic. Further, there is still the potential to reduce gas consumption by changing the public inputs to private inputs and only providing a hash of the previously public inputs to the circuits to check if the inputs are correct. However, this comes at the cost of higher resource consumption for generating the proofs since most blockchains do not support arithmetization-friendly hash functions.

6.3 Performance

As has already been discussed before, the execution of smart contracts introduces additional costs for clients and oracle nodes. Therefore, the oracle nodes move as much computation and storage off-chain as possible to reduce costs. The aggregator executes the aggregation and slashing mechanisms locally and generates zk-SNARKs to allow the *zkOracle* contract to verify these computations on the blockchain. However, in exchange, the current aggregator must spend a lot of computational resources to generate a zk-SNARK. The aggregator's time to generate the proofs is a critical aspect affecting the system's throughput. The proof generation time depends mainly on the complexity of the computations, i.e., the aggregation and slashing algorithms. The more complex the calculation, the larger the arithmetic circuit and the longer it takes to generate the proof. Therefore, we examine the influence of the committee size on the proof generation time.

The results (see Figure 4) show that the proof generation time for the aggregation algorithm increases linearly with the committee size. The proof generation time for the slashing algorithm, on the other hand, is constant in intervals depending on the height of the Merkle tree. However, the proof generation time for the aggregation algorithm increases considerably faster since a bigger committee size requires more votes from the validators and, therefore, more Merkle proofs, signatures, and value transfers, leading to a bigger circuit. Generating the proof for the aggregation algorithm takes 787 ms ± 14 ms with a committee size of 4 up to 35.4 s ± 137 ms with a committee size of 256. The proof generation time for the slashing algorithm is 369 ms ± 12 ms for a committee of size 4 and 428 ms ± 2 ms for a committee of size 256. The measured proof generation time allows for the practical application of the proposed solution since blockchains exhibit block times in the range of multiple seconds, limiting the overall throughput.

6.4 Memory

Memory consumption is another critical aspect to consider for the applicability of the proposed solution in a real-world scenario. The reason for this is a higher entry barrier for oracle nodes and, consequently, a possibly lower degree of decentralization, as oracle nodes require more memory to execute the aggregation mechanism. Therefore, after examining the proof generation time, we also inspect the memory consumption. As with the proof generation time, the memory consumption also depends on the complexity of the algorithms due to the corresponding larger size of the arithmetic circuit. Therefore, we also inspect the memory consumption during

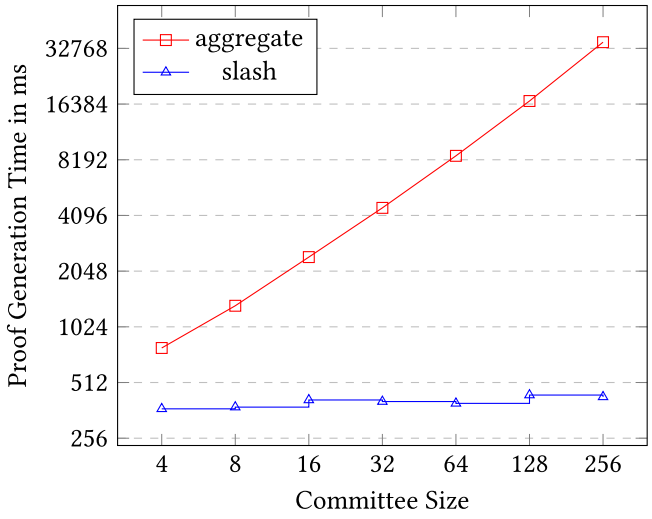

Fig. 4. Time to generate the proofs.

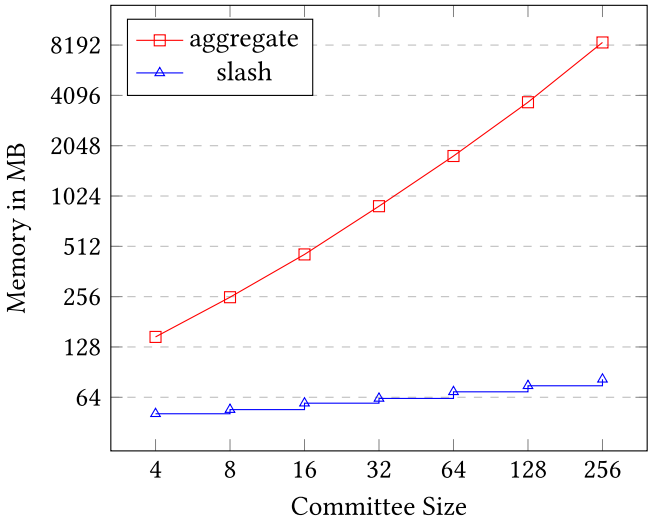

Fig. 5. Memory usage during proof generation.

proof generation for a varying committee size. For that, we consider the allocated memory for building the constraint system, the proving key, creating the witness, and generating the proof.

The results (see Figure 5) show that memory consumption follows the same behavior as the proof generation time. The memory consumption increases linearly for the aggregation algorithm and stepwise for the slashing algorithm. As with the proving time, the increasing complexity of the aggregation circuit makes the memory consumption higher and faster growing than for the slashing circuit. The generation of the aggregation proof requires 147 MB with a committee size of 4 and 8.5 GB with a committee size of 256. The generation of the slashing proof, on the other hand, only requires 51 MB for a committee size of 4 and 82 MB for a committee size of 256.

Therefore, oracle nodes, which do not have extensive resources, can still join the system and support fairly large committee sizes.

7 Related Work

In the course of this section, we discuss related work concerning cross-blockchain communication mechanisms and protocols. Among other things, we examine blockchain relays, notary schemes, blockchains of blockchains, and others.

7.1 Blockchain Relays

The authors of [31] propose ETH relay, a relay scheme for Ethereum-based blockchains which use the **Proof of Work (PoW)** consensus mechanism. ETH relay applies a validation-on-demand pattern where the relay contract optimistically accepts new block headers from the source blockchain. Off-chain clients continuously relay block headers from the source blockchain to the target blockchain but also monitor the relay contract to detect invalid block headers. Any off-chain client can trigger the validation of a submitted block header during a dispute period. If the block is invalid, the relay contract removes the invalid and all subsequent block headers. The relay considers a block valid if there is no dispute and the dispute period has ended. Additionally, the relay applies an incentive mechanism to encourage honest behavior and reward off-chain clients for block header submissions. While the solution achieves considerable improvements concerning cost efficiency, the cost overhead of relaying all block headers remains high.

Scaffino et al. [56] propose a cross-blockchain communication mechanism that retrieves on-demand state information about a PoW source blockchain on a target blockchain. A contract on the target blockchain can verify constant-sized proofs consisting of n confirmation block headers showing that certain transactions are part of the source blockchain. The proof construction is similar to stateless SPV but uses additional mechanisms to prevent upfront mining attacks and does not require Turing-complete smart contracts. Further, the authors prove the protocol's security and show how it enables several cross-chain applications such as lending, Proofs-of-Burn, and off-chain betting hubs.

Westerkamp and Eberhardt [64] propose zkRelay, another relay solution for PoW blockchains. The relay uses zk-SNARKs to validate block headers off-chain to reduce the costs of block header submissions. Off-chain clients generate a zk-SNARK for an off-chain validation program which takes a block header as input and returns a Boolean value indicating if the block is valid. Further, zkRelay supports batch verification of block headers, while the relay contract only needs to store the last block of a batch. The validation program builds a Merkle tree for the intermediary block headers and returns the root along with the validation result. After that, the relay allows the submission of intermediary block headers, which only requires a Merkle proof to show the inclusion of the block. The application of zk-SNARKs to verify the blocks off-chain leads to better cost-efficiency. However, off-chain clients need more computing power and memory to generate the proofs.

Another protocol for cross-blockchain communication based on zk-SNARKs is Zendoo [33]. Zendoo allows sidechain constructions for Bitcoin-like blockchains where sidechains directly monitor their respective mainchain. However, the mainchain only receives zk-SNARKs to verify the state of the sidechain. Sidechains can have different internal structures and follow distinct approaches to create certificates for communicating with the mainchain as long as they follow the basic verifier interface. Unfortunately, Zendoo only allows the transfer of assets and not arbitrary data.

Similar to [33] and [64], the authors of [68] propose a trustless blockchain bridge that uses zk-SNARKs to reduce on-chain costs. The generation of zk-SNARKs requires expressing the validation rules of the connected blockchains in an arithmetic circuit, which can be too inefficient for certain operations. Therefore, the authors propose a two-layer recursive proof system. The first layer applies the proposed proof system *deVirgo*, which allows for the parallel generation of the proof to achieve better performance. The second layer uses Groth16

to recursively verify proofs of the first layer and reduce the on-chain verification costs of the proof. The smart contract checks if the block is included and verifies the submitted proof against the current state and the block header given. Further, zkBridge also supports batch submissions of block headers.

Westerkamp and Diez [63] propose Verilay, a blockchain relay for PoS blockchains. The relay contract on the target blockchain starts from a trusted state containing the starting block and the corresponding validator set. After that, any user can update the relay state by retrieving a specific block, the current and next validator set, and submitting it to the relay contract on the target blockchain. The relay contract validates a block by checking the signatures and replaces the currently stored block if the signatures are valid. Further, the relay checks whether to transition to a new validator set. Verilay only requires a single update for each validator set period since every block contains the history of the whole chain. The authors implement Verilay for the Ethereum 2.0 beacon chain and show that Verilay is more cost-efficient than PoW relays. However, the relay only ensures accountable safety during a trusting period when the validators cannot withdraw the bonded stake. Otherwise, validators can misbehave without consequences. Further, the complexity of validator set changes can pose a challenge if they are not efficiently verifiable by the relay contract.

The discussed relay schemes enable cross-blockchain communication with a high degree of decentralization. However, these solutions heavily depend on the applied consensus mechanism of the source blockchain, as a smart contract or arithmetic circuit must be able to express the source blockchain's consensus mechanism. As a result, the solutions have a high complexity level and require many resources. On the other hand, the proposed oracle solution requires little to no changes depending on the connected blockchains. Therefore, the complexity is also correspondingly lower, leading to considerable resource savings. However, the degree of decentralization is lower as the oracle relies on a fixed-size committee.

7.2 Notary Schemes

In [60], the authors propose a voting-based oracle solution for cross-blockchain communication. The oracle applies an off-chain voting mechanism based on threshold signatures to allow a committee to collaboratively return information queried from another blockchain while keeping on-chain computation at a bare minimum. For that, the oracle divides its participants into an aggregator and multiple validators. The validators query the source blockchain for the requested data and sign the result with their private key shares. The aggregator collects results and the corresponding signature shares from the validators until the aggregator can create the threshold signature. After that, the aggregator returns the data and the threshold signature to an oracle contract, which only has to verify the threshold signature to verify the authenticity and integrity of the data. Further, the oracle applies a crypto-economic incentive mechanism to give aggregators a fixed reward and the chance to win an additional reward, increasing over time until an aggregator wins.

The decentralized oracle network Chainlink [18] has recently introduced the Cross-Chain Interoperability Protocol to allow cross-chain messages and token transfers. For that, Chainlink extends the application of its OCR protocol [19] to the domain of cross-blockchain communication. The OCR protocol allows multiple oracle nodes to aggregate their results in a single report and only to submit an aggregated transaction. The protocol's main purpose is to retrieve data from external systems. However, it also allows for efficient and secure off-chain computations, enabling cross-chain messaging capabilities. A smart contract can invoke a Chainlink message router on the source blockchain, which uses the decentralized oracle network to transport the message to a message router on the target blockchain. The message router on the target blockchain verifies the message and forwards the message to another smart contract.

The authors of [1] propose Chain-net, a blockchain interoperability framework for cross-blockchain communication. The solution uses additional gateway nodes for each blockchain network. The gateway nodes establish the connection between different blockchains and forward cross-chain transactions. Further, the gateway ensures the atomicity of cross-chain transactions by waiting for the necessary transaction confirmations of the involved

blockchains. The authors present a possible application in the healthcare domain and evaluate it against certain criteria. Unfortunately, the gateway nodes constitute a central entity in the system, contrary to blockchains' decentralized nature.

Madine et al. [48] propose appXchain, a blockchain interoperability solution based on a dApp that manages the execution of cross-blockchain transactions. The dApp communicates with the default APIs of the underlying blockchain networks and acts as a translation layer between blockchain networks. The system requires verifier nodes for each blockchain to retrieve and verify the state from other blockchains and applies a reputation system to encourage honest behavior. Further, the authors explain the proposed solution using an example in healthcare to manage patient data.

Robinson and Ramesh [54] propose a general-purpose atomic cross-chain transaction protocol. The protocol allows synchronous cross-blockchain smart contract calls for Ethereum-based blockchains. For executing a function call involving multiple blockchains, the application needs to simulate the execution of a call graph on all chains to determine the respective input parameters to the function calls. After that, a cross-chain control contract manages the execution on each blockchain. Further, each blockchain requires a registrar contract to determine a set of signers that attest to the emitted event's correctness during function calls.

In [51], the authors propose a framework for asynchronous cross-blockchain smart contract calls. The framework requires intermediaries that broker information between the source blockchain and the target blockchain, i.e., relay specific smart contract calls and return the results. For that, the intermediaries offer to execute cross-blockchain smart contracts calls for a reward. The framework applies crypto-economic incentives and relies on a trusted set of validators that vote on the validity of smart contract invocations. More specifically, the validators check that intermediaries execute the correct function, return the correct results, and use the right amount of computation steps.

The above-mentioned solutions allow for cross-blockchain smart contract calls and general cross-blockchain communication. Unfortunately, some of these solutions require many on-chain resources, are centralized, or need more details on the design and implementation. The approach presented in [60] comes closest to the work at hand, whereby our proposed solution does not require a DKG protocol or a lottery-like mechanism for reward distribution. Our solution does not require any additional protocols, probably jeopardizing security. Further, the proposed oracle allows for the equal splitting of rewards between the participating entities without incurring additional costs.

7.3 Blockchain of Blockchains

The Cosmos [45] network connects multiple independent blockchains, also called zones, which can seamlessly interoperate with each other. Cosmos uses an **Inter-blockchain Communication (IBC)** Protocol [35] to transfer arbitrary data between blockchains. The IBC protocol follows the basic concepts of TCP/IP. IBC allows modules on different blockchains to establish channels to transmit packets containing arbitrary data, e.g., information about a token transfer. The connected blockchains verify the transferred data packets by checking the inclusion of the data packet in the respective blockchain state. Further, off-chain clients with access to both blockchains handle the actual transmission of the packets between the ledgers.

Another blockchain network aiming for blockchain interoperability is Polkadot [66]. Polkadot connects multiple so-called parachains through a relay chain, providing shared security and the possibility of trustless communication between all connected parachains. The main actors of the system are nominators, validators, and collators. The validators are randomly divided into subsets for each parachain and add new blocks to the relay chain. Polkadot uses the Cross-Consensus Message Format for cross-blockchain communication, which defines a communication language for parachains. During cross-blockchain communication, nodes place cross-chain messages in message queues and continuously ping other nodes to retrieve new messages. When assembling a new block, nodes process cross-chain messages and transactions.

In [47], the authors propose HyperService, a platform for blockchain interoperability that allows the execution of cross-blockchain smart contracts. The platform introduces a new programming language to develop cross-blockchain applications under a universal state model that abstracts state transitions across multiple blockchains. HyperService uses a universal inter-blockchain protocol to ensure the atomic and trustless execution of the transactions submitted to the underlying blockchains. A blockchain of blockchains provides a unified view of cross-blockchain applications by recording the transaction's status. At the same time, insurance smart contracts determine the correctness of the execution and revert transactions in case of exceptions.

Yeh et al. [69] propose a cross-blockchain communication mechanism based on a relay chain and certificateless signatures. The relay chain connects two heterogeneous blockchains and applies a Delegated POS consensus mechanism independent of the consensus mechanism of the connected blockchains. During the execution of a cross-blockchain call, one party on each of the connected blockchains and a block producer collaborate to execute a cross-blockchain transaction and create the respective block in the relay chain. The authors implement the proposed solution for two private Ethereum-based blockchains and evaluate the solution's performance and security.

The systems based on the blockchain of blockchains approach enable cross-blockchain communication through a central relay chain, which records cross-blockchain transactions and attests to the correctness of the transferred data between the connected blockchains. However, this approach also entails considerable overhead, requiring an additional blockchain network to manage the relay chain with its consensus rules. Therefore, the proposed oracle solution represents a far more lightweight solution since the oracle builds upon already existing blockchains.

8 Conclusion

Cross-blockchain communication is one of the most pressing problems in enabling blockchain interoperability and connecting heterogenous blockchains to form a unified landscape of blockchain systems. Research has led to different solutions for achieving blockchain interoperability by proposing different cross-blockchain communication mechanisms. However, these solutions are primarily complex based on the connected blockchains, require additional blockchains, or are very resource-intensive.

Therefore, we propose an off-chain communication mechanism based on oracles. The oracle comprises a committee of oracle nodes that query the source blockchain for specific data. The *zkOracle* contract only stores the root of a Merkle tree, storing the account data of the oracle nodes to move state and computation off-chain. For each request, the oracle nodes need to execute an off-chain aggregation mechanism to aggregate the retrieved data from the oracle nodes and reach a consensus on the returned result. The oracle schedules one oracle node as an aggregator while the remaining nodes act as validators. The validators query the source blockchain to retrieve the data and send the signed result to the aggregator. Then, the aggregator verifies that the majority returned the same data and distributes the rewards. For this computation, the aggregator generates a zk-SNARK and only submits the new state root, result, and proof to the *zkOracle* contract. The contract must only verify the proof's correctness and store the updated state root and result. Additionally, the aggregator can slash misbehaving validators to ensure accountability and incentivize honest behavior. The proposed oracle solution requires minimal computation and storage on the blockchain and offers the necessary functionality to enable the decentralized transfer of arbitrary data between blockchains.

In future work, we want to explore possible optimizations through batch submissions to reduce the costs of the aggregator for submitting an aggregated result. Further, we want to explore possibilities to move the remaining state of the *zkOracle* contract off-chain.

Acknowledgements

The financial support from the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, and the Christian Doppler Research Association is gratefully acknowledged.

References

[1] Sidrah Abdullah, Junaid Arshad, and Muhammad Alsadi. 2022. Chain-Net: An internet-inspired framework for interoperable blockchains. *Distributed Ledger Technologies:* Research and Practice 1, 2 (2022), 1–20.

[2] Cornelius C. Agbo, Qusay H. Mahmoud, and J. Mikael Eklund. 2019. Blockchain technology in healthcare: A systematic review. In *Healthcare*, Vol. 7. 56.

[3] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. 2018. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. arXiv:1809.09044. Retrieved from https://arxiv.org/abs/1809.09044

[4] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic. 2020. Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access* 8 (2020), 85675–85685.

[5] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 191–219.

[6] Ghada Almashaqbeh and Ravital Solomon. 2022. SoK: privacy-preserving computing in the blockchain era. In *Proceedings of the IEEE 7th European Symposium on Security and Privacy (EuroS & P)*. IEEE, 124–139.

[7] Rafael Belchior, Luke Riley, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2023. Do you need a distributed ledger technology interoperability solution? *Distributed Ledger Technologies: Research and Practice* 2, 1 (2023), 1–37.

[8] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys* 54, 8 (2021), 1–41.

[9] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM, 62–73.

[10] Marta Bellés-Muñoz, Barry Whitehat, Jordi Baylina, Vanesa Daza, and Jose L. Muñoz-Tapia. 2021. Twisted Edwards elliptic curves for zero-knowledge circuits. *Mathematics* 9, 23 (2021), 3022.

[11] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable zero knowledge with no trusted setup. In *Proceedings of the 39th Annual International Cryptology Conference*. Springer, 701–732.

[12] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014a. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.

[13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014b. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium*. USENIX Association, 781–796.

[14] Daniel J Bernstein. 2006. Curve25519: New Diffie-Hellman speed records. In *Proceedings of the 9th International Conference on Theory and Practice in Public-Key Cryptography*. Springer, 207–228.

[15] Manuel Blum, Paul Feldman, and Silvio Micali. 2019. Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Oded Goldreich (Ed.), ACM, 329–349.

[16] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. 2023. ConsenSys/gnark: v0.8.0. DOI: https://doi.org/10.5281/zenodo.5819104

[17] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-Party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Paper 2017/1050. Retrieved from https://eprint.iacr.org/2017/1050

[18] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramèr, and Fan Zhang. 2021. Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. Retrieved March 16, from https://naorib.ir/white-paper/chinlink-whitepaper.pdf

[19] Lorenz Breidenbach, Christian Cachin, Alex Coventry, Ari Juels, and Andrew Miller. 2021. Chainlink Off-Chain Reporting Protocol. Retrieved March 07, from https://research.chain.link/ocr.pdf

[20] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE, 315–334.

[21] Vitalik Buterin. 2016. Chain interoperability. *R3 Research Paper* 9 (2016), 1–25.

[22] Vitalik Buterin. 2021. An Incomplete Guide to Rollups. Retrieved March 09, from https://vitalik.ca/general/2021/01/05/rollup.html

[23] Wei Cai, Zehua Wang, Jason B. Ernst, Zhen Hong, Chen Feng, and Victor C. M. Leung. 2018. Decentralized applications: The blockchain-empowered software system. *IEEE Access* 6 (2018), 53019–53033.

[24] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 738–768.

[25] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile verifiable computation. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. IEEE, 253–270.

[26] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. 2019. Blockchain for internet of things: A survey. *IEEE Internet of Things Journal* 6, 5 (2019), 8076–8094.

[27] Konstantinos Demestichas, Nikolaos Peppes, Theodoros Alexakis, and Evgenia Adamopoulou. 2020. Blockchain in agriculture traceability systems: A review. *Applied Sciences* 10, 12 (2020), 4113.
[28] John R Douceur. 2002. The sybil attack. In *Proceedings of the International Workshop on Peer-to-Peer Systems*. Springer, 251–260.
[29] Joshua Ellul and Gordon J. Pace. 2022. Verifiable external blockchain calls: Towards removing oracle input intermediaries. In *International Workshop on Data Privacy Management*. Springer, 317–324.
[30] Hyperledger Foundation. 2022. Introducing Hyperledger Cacti, a Multi-Faceted Pluggable Interoperability Framework. Retrieved March 07, from https://www.hyperledger.org/blog/2022/11/07/introducing-hyperledger-cacti-a-multi-faceted-pluggable-interoperability-framework
[31] Philipp Frauenthaler, Marten Sigwart, Christof Spanring, Michael Sober, and Stefan Schulte. 2020. ETH relay: A cost-efficient relay for ethereum-based blockchains. In *Proceedings of the 2020 IEEE International Conference on Blockchain*. IEEE, 204–213.
[32] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953. Retrieved from https://eprint.iacr.org/2019/953
[33] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. 2020. Zendoo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In *Proceedings of the IEEE 40th International Conference on Distributed Computing Systems (ICDCS '20)*. IEEE, 1257–1262.
[34] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Proceedings of the 30th Annual Cryptology Conference*. Springer, 465–482.
[35] Christopher Goes. 2020. The interblockchain communication protocol: An overview. arXiv:2006.15918. Retrieved from https://arxiv.org/abs/2006.15918
[36] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1991. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM* 38, 3 (1991), 690–728.
[37] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. 2019. The knowledge complexity of interactive proof-systems. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Oded Goldreich (Ed.), ACM, 203–225.
[38] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing* 17, 2 (1988), 281–308.
[39] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 305–326.
[40] Abdelatif Hafid, Abdelhakim S. Hafid, and Mustapha Samih. 2020. Scaling blockchains: A comprehensive survey. *IEEE Access* 8 (2020), 125244–125262.
[41] Jonathan Heiss, Jacob Eberhardt, and Stefan Tai. 2019. From oracles to trustworthy data on-chaining systems. In *Proceedings of the IEEE International Conference on Blockchain (Blockchain '19)*. IEEE, 496–503.
[42] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. ACM, 245–254.
[43] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash Protocol Specication. Retrieved June 24, from https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf
[44] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, 1353–1370.
[45] Jae Kwon and Ethan Buchman. 2020. Cosmos Whitepaper: A Network of Distributed Ledgers. Retrieved March 16, from https://wikibitimg.fx994.com/attach/2020/12/16623142020/WBE16623142020_55300.pdf
[46] Pascal Lafourcade and Marius Lombard-Platet. 2020. About blockchain interoperability. *Information Processing Letters* 161 (2020), 105976.
[47] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. 2019. Hyperservice: Interoperability and programmability across heterogeneous blockchains. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 549–566.
[48] Mohammad Madine, Khaled Salah, Raja Jayaraman, Yousof Al-Hammadi, Junaid Arshad, and Ibrar Yaqoob. 2021. appXchain: Application-level interoperability for blockchain networks. *IEEE Access* 9 (2021), 87777–87791.
[49] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. 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*. ACM, 2111–2128.
[50] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved July 02, from http://www.bitcoin.org/bitcoin.pdf
[51] Markus Nissl, Emanuel Sallinger, Stefan Schulte, and Michael Borkowski. 2021. Towards cross-blockchain smart contracts. In *Proceedings of the 2021 IEEE International Conference on Decentralized Applications and Infrastructures*. IEEE, 85–94.
[52] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM* 59, 2 (2016), 103–112.
[53] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. 2012. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Proceedings of the 9th Theory of Cryptography Conference*. Springer, 422–439.

[54] Peter Robinson and Raghavendra Ramesh. 2021. General purpose atomic crosschain transactions. In *Proceedings of the 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services*. IEEE, 61–68.
[55] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. 2019. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research* 57, 7 (2019), 2117–2135.
[56] Giulia Scaffino, Lukas Aumayr, Zeta Avarikioti, and Matteo Maffei. 2023. Glimpse: On-demand PoW light client with constant-size storage for DeFi. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security '23)*. USENIX Association, 733–750.
[57] Giulia Scaffino, Lukas Aumayr, Mahsa Bastankhah, Zeta Avarikioti, and Matteo Maffei. 2024. Alba: The Dawn of Scalable Bridges for Blockchains. Cryptology ePrint Archive, Paper 2024/197. Retrieved from https://eprint.iacr.org/2024/197
[58] Stefan Schulte, Marten Sigwart, Philipp Frauenthaler, and Michael Borkowski. 2019. Towards blockchain interoperability. In *Proceedings of the Business Process Management: Blockchain and Central and Eastern Europe Forum: BPM 2019 Blockchain and CEE Forum*. Springer, 3–10.
[59] Michael Sober, Max Kobelt, Giulia Scaffino, Dominik Kaaser, and Stefan Schulte. 2023. Distributed key generation with smart contracts using zk-SNARKs. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. ACM, 231–240.
[60] Michael Sober, Giulia Scaffino, Christof Spanring, and Stefan Schulte. 2021. A voting-based blockchain interoperability oracle. In *Proceedings of the 2021 IEEE International Conference on Blockchain*. IEEE, 160–169.
[61] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. 2022. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access* 10 (2022), 93039–93054.
[62] Gang Wang, Qin Wang, and Shiping Chen. 2023. Exploring blockchains interoperability: A systematic survey. *ACM Computing Surveys* 55, 13s (2023), 1–38.
[63] Martin Westerkamp and Maximilian Diez. 2022. Verilay: A verifiable proof of stake chain relay. In *Proceedings of the 2022 IEEE International Conference on Blockchain and Cryptocurrency*. IEEE, 1–9.
[64] Martin Westerkamp and Jacob Eberhardt. 2020. zkRelay: Facilitating sidechains using zkSNARK-based chain-relays. In *Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops*. IEEE, 378–386.
[65] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Retrieved June 13, from https://ethereum.github.io/yellowpaper/paper.pdf
[66] Gavin Wood. 2016. Polkadot: Vision for a Heterogeneous Multi-Chain Framework. Retrieved March 16, from https://www.win.tue.nl/mholende/seminar/references/ethereum_polkadot.pdf
[67] Junfeng Xie, Helen Tang, Tao Huang, F. Richard Yu, Renchao Xie, Jiang Liu, and Yunjie Liu. 2019. A survey of blockchain technology applied to smart cities: Research issues and challenges. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2794–2830.
[68] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. 2022. zkBridge: Trustless cross-chain bridges made practical. In *2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*. ACM, 3003–3017.
[69] Kuo-Hui Yeh, Guan-Yan Yang, Chanapha Butpheng, Lin-Fa Lee, and Ying-Ho Liu. 2022. A secure interoperability management scheme for cross-blockchain transactions. *Symmetry* 14, 12 (2022), 2473.
[70] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. 2021. SoK: Communication across distributed ledgers. In *Proceedings of the 25th International Conference on Financial Cryptography and Data Security, Revised Selected Papers, Part II*. Springer, 3–36.
[71] Yuhang Zhang, Jun Wang, and Jie Luo. 2020. Heuristic-based address clustering in bitcoin. *IEEE Access* 8 (2020), 210582–210591.
[72] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. 2020. Solutions to scalability of blockchain: A survey. *IEEE Access* 8 (2020), 16440–16455.

Received 16 June 2023; revised 26 April 2024; accepted 1 July 2024