TU WIEN Informatics

Blockchain Interoperability: A Cross-Chain Token Transfer Protocol

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Dragan Tosic, BSc
Matrikelnummer 00927956

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr.-Ing. Stefan Schulte
Mitwirkung: Dr. techn. Michael Borkowski

Wien, 18. April 2024 [redacted] [redacted]

Dragan Tosic Stefan Schulte

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Informatics

Blockchain Interoperability: A Cross-Chain Token Transfer Protocol

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Dragan Tosic, BSc
Registration Number 00927956

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr.-Ing. Stefan Schulte
Assistance: Dr. techn. Michael Borkowski

Vienna, 18th April, 2024

[redacted] [redacted]

Dragan Tosic Stefan Schulte

Erklärung zur Verfassung der Arbeit

Dragan Tosic, BSc

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

Wien, 18. April 2024

[redacted]

Dragan Tosic

Acknowledgements

I would like to thank my advisor Prof. Dr.-Ing. Stefan Schulte who gave me the opportunity to write this thesis at the Distributed Systems Group. He always provided very constructive and fast feedback, and continued supervising me even after he left TU Wien. Furthermore, I would like to express my greatest gratitude to my co-advisor Dr. techn. Michael Borkowski for his guidance and valuable feedback. Last but not least, I would like to thank my parents, sister, and girlfriend for their unlimited support during the years of my study.

Kurzfassung

Blockchains, die wichtigste zugrundeliegende Technologie hinter Kryptowährungen, haben in der Informatik, im Finanzwesen, in der Wirtschaft und in der Forschung großes Interesse geweckt. Blockchains bieten die Möglichkeit, Daten zu speichern und Datenintegrität in einer vertrauenswürdigen Umgebung zu gewährleisten, jedoch laufen sie grundsätzlich in einer isolierten Umgebung. Eingeschränkte Interoperabilität zwischen Blockchains führt zu Kompromissen, die Eigentümer digitaler Assets und Entwickler verteilter Anwendungen machen müssen. Eine erhöhte Interoperabilität würde es Benutzern ermöglichen, ihre Assets und Daten über mehrere Blockchains zu verteilen, und würde es Entwicklern ermöglichen, Blockchain-unabhängige verteilte Anwendungen zu entwickeln. Im Zuge der Erreichung der Interoperabilität zwischen Blockchains stehen wir vor dem Problem, dass es nicht möglich ist, innerhalb einer Blockchain zu überprüfen, ob bestimmte Daten auf einer anderen Blockchain aufgezeichnet wurden.

In dieser Arbeit stellen wir das Konzept von teilweise dezentralisierten Validation Parties vor und argumentieren, wie dieses Konzept Blockchain-übergreifende Token-Transfers auf überprüfbare Weise ermöglichen kann, ohne die Integrität der übertragenen Tokens zu beeinträchtigen.

Im theoretischen Teil dieser Arbeit untersuchen wir die Literatur im Blockchain-Bereich, wobei der Schwerpunkt auf dem State of the Art im Bereich Blockchain-Interoperabilität liegt. Ziel ist es, die größten Herausforderungen für die Interoperabilität von Blockchains zu identifizieren und Anforderungen für ein Blockchain-übergreifende Token-Transfers Protokoll auf Basis einer teilweise dezentralisierten Architektur zur Verifizierung von Blockchain-übergreifenden Beweisen zu definieren und zu spezifizieren. Der praktische Teil dieser Arbeit zielt darauf ab, einen Prototyp des definierten Protokolls als Smart Contract für Ethereum-basierte Blockchains zu implementieren. Um die Machbarkeit des implementierten Protokolls zu zeigen, wird ein Proof of Concept einer prototypisch verteilten Anwendung entwickelt, die in der Lage ist, Blockchain-übergreifende Token-Transfers durchzuführen.

Letztendlich werden das Blockchain-übergreifende Token-Transfer Protokoll und der Prototyp in einer aus zwei Ethereum-basierten Blockchains bestehenden Testumgebung bereitgestellt und evaluiert. Wir diskutieren und bewerten verschiedene Anwendungsfälle sowie gesammelte Metriken wie Transaktionskosten und Übertragungszeiten.

Abstract

Blockchains, the main underlying technology behind cryptocurrencies, have gained significant interest in computer science, finance, economics, and research. Blockchains provide the means to store data and guarantee its integrity in a trustless environment, but notably they run in an isolated setting. Limited interoperability between blockchains leads to compromises that owners of digital assets and developers of distributed applications have to make. Increased interoperability would allow users to distribute their assets and data across multiple blockchains, and would enable developers to design blockchain-agnostic distributed applications. In the course of achieving interoperability between blockchains, we face the problem that it is not possible to verify that specific data has been recorded on one blockchain from within another blockchain.

In this thesis, we present the concept of semi-decentralized validation parties, and argue how this concept can enable cross-chain token transfers in a verifiable manner without compromising the integrity of the transferred tokens.

In the theoretical part of this thesis, we survey literature in the blockchain field, with focus on the current state-of-the-art in blockchain interoperability. The aim is to identify the main challenges to achieve blockchains interoperability, and to define and specify requirements for a cross-chain token transfer protocol based on a semi-decentralized architecture for the verification of cross-chain proofs. The practical part of this thesis aims to implement a prototype of the defined protocol as a smart contract for Ethereum-based blockchains. To show the feasibility of the implemented protocol, a proof of concept of a prototypical distributed application able to perform cross-chain token transfers is developed.

Finally, the cross-chain token transfer protocol and the prototype are deployed and evaluated in a test environment consisting of two Ethereum-based blockchains. We discuss and evaluate various use cases, as well as collected metrics data such as transaction cost and transfer times.

Contents

CHAPTER 1

Introduction

This chapter provides an overview of the problem addressed by this thesis, including motivation, problem statement, aim of the work, and methodological approach used to solve the problem.

1.1 Motivation

Since the introduction of Bitcoin in 2009, cryptocurrencies and the main underlying technology, blockchains, have gained significant interest in various fields, such as computer science, research, finance, economics, society, etc. A blockchain is a data structure consisting of blocks that are linked to each other by hash values forming a linked list (chain). Each block contains data, the exact form of which is defined by each blockchain (e.g., Bitcoin transactions), a timestamp, a hash of that block and a hash of the previous block. This structure makes data in the blockchain tamper-resistant, because any change of a single block would void the block's hash and all subsequent block hashes. Blockchains are distributed over a peer-to-peer network and they represent a distributed ledger. Adding new blocks to a chain requires a majority consensus as defined by the consensus algorithm of the particular blockchain (e.g., the majority of computing power decides on the canonical blockchain) [22].

Bitcoin has demonstrated how distributed ledgers can be used to provide a completely decentralized digital currency, which does not require a centralized verification entity, and which cannot be controlled by a single entity (e.g., governments) [41]. Since the advent of Bitcoin, a lot of research and development has been invested into new blockchain technologies that can now be applied for storing any type of tamper-sensitive data, or for building serverless distributed applications that are running on top of blockchains. For instance, blockchains can be applied for runtime verification of business processes [19], or for storing highly sensitive medical records in healthcare [42].

Blockchains provide the means to store and guarantee integrity of data in a trustless environment, but notably they run in an isolated setting, and there is no interconnection between different blockchains foreseen. Therefore, users are forced to store their data (including digital currencies or other domain-specific data) on the blockchain they have most trust in, and developers must choose a single blockchain for developing their distributed applications. Since various blockchains might offer different features and benefits, this means that choosing one blockchain might force the developer to a compromise in the design process. Finally, in case of a security vulnerability or cryptographic break in a blockchain's design or implementation, all digital assets on the chain selected by the user or developer may be compromised. This increases the risk when utilizing blockchain technology [33].

1.2 Problem Statement

To overcome limitations of relying on a single blockchain, it is necessary to enable interoperability between blockchains. First, this allows users to distribute their assets and data across multiple blockchains and move the assets from one blockchain to another at their will. Second, with increased interoperability, developers can design applications that are mostly blockchain-agnostic, and possibly benefit from multiple blockchains' benefits. Finally, the risk of asset loss is minimized since data, including digital assets, can be freely moved between blockchains [16, 33].

While solutions like Atomic Swaps [27] only partly enable interoperability by guaranteeing the atomicity of cross-chain assets exchange, current state-of-the-art solutions do not support the actual transfer of assets between blockchains, where some data is destroyed or reduced on one blockchain and created on another.

In the course of achieving interoperability between blockchains, we face the cross-blockchain proof problem [31, 32]: If on a Blockchain A we want to prove that some data D has been recorded on that blockchain, we can easily traverse over all blocks on the Blockchain A until we find a block containing D. However, transferring D to another Blockchain B poses a challenge, since there are no practical means, for two independent Blockchains A and B, to verify on the Blockchain B whether the data D has been recorded on the Blockchain A, as argued and formalized in [32].

In order to solve the problem, we propose the concept of semi-decentralized validation parties that we are going to refer to as "validators". In this thesis, we argue how such validators can enable cross-chain token transfers in a verifiable manner without compromising the integrity of the transferred tokens.

1.3 Aim of the Work

The aim of the theoretical part of this thesis is to survey literature in the blockchain field, with special focus on the current state-of-the-art in cross-chain interoperability technologies. Since this is a very recent topic, scientific papers are emerging, e.g., [16, 27, 31]. Similar approaches shall be analyzed, and their advantages and drawbacks compared, to get more insight into the field.

The goal is to define a cross-chain token transfer protocol by introducing a semi-decentralized architecture for the verification of cross-chain proofs. While the proof of destroying the tokens on the source blockchain must be provided by a party trusted by the network, i.e., a centralized model is used, the validity of this proof can in turn be verified by anyone else. Thus, we call the approach semi-decentralized. This means that while a central party is required to execute the cross-chain transfers, anyone can verify that the central party is working correctly. The central party has disincentive to give untrue confirmations, since doing so would destroy trust into the entire process.

The practical part of this thesis aims to implement a prototype based on the defined protocol and architecture. The prototype shall be implemented using the Solidity contract-oriented programming language for Ethereum-based blockchains. To demonstrate the feasibility of the proposed solution, a proof of concept shall be applied on two Ethereum-based test networks that performs a transfer of an ERC20 token between the two blockchains.

1.4 Methodological Approach

The methodological approach of this thesis is divided into five phases.

Literature Research In the first phase, literature and the current state-of-the-art in the blockchain field are reviewed and studied, with special focus on cross-chain interoperability technologies. The goal is to identify the main challenges to achieve interoperability between multiple blockchains, and to propose solutions and requirements to overcome these challenges. The research is not only focused on cross-chain token transfer techniques, but also includes general approaches to achieve an interconnection between different blockchains.

Protocol Definition Based on the results and identified requirements from the first phase, a protocol that allows cross-chain token transfers shall be defined and specified. The protocol shall assure by its design that the integrity of tokens on all involved blockchains is not compromised during a cross-chain transfer.

Protocol Implementation The protocol defined in Phase 2 shall be implemented as a smart contract for Ethereum-based blockchains. Since smart contracts may be deployed on public blockchains, and anybody could attempt potentially malicious function calls, the security hardening of the smart contract will play an important role during the protocol implementation phase.

Proof of Concept Implementation To demonstrate the feasibility of the protocol defined and implemented in the previous two phases, it is planned to provide a proof of concept by implementing a prototypical distributed application able to perform cross-chain token transfers by interacting with the smart contract implemented in Phase 3.

Evaluation In the final phase, the implementation of the cross-chain token transfer protocol and the prototype are evaluated, and possible enhancements are identified. The evaluation requires setting up a test environment consisting of two Ethereum-based blockchains. The test environment will be used for performing and analyzing various use cases, as well as for collecting metrics data such as transaction cost and transfer times. Furthermore, results of a comparison with similar approaches are shown, and limitations are presented.

1.5 Structure of the Work

The remainder of this thesis is structured as follows:

- Chapter 2 provides background information for the succeeding chapters. It provides a short introduction to cryptocurrencies, the main technology behind them — blockchains, and an overview of smart contract technologies that we will encounter in the practical part of this thesis.
- Chapter 3 entails a survey of related work. In our research, we present and discuss approaches that are related to the interoperability between multiple blockchains.
- Chapter 4 begins with the definition of the cross-blockchain proof problem, and continues with motivating scenarios and the requirements specification.
- Chapter 5 describes the architecture of a solution for a cross-chain token transfer based on the research of the cross-blockchain proof problem.
- Chapter 6 provides implementation details of a smart contract, which enforces a cross-chain token transfers protocol. Implementation details of components that interact with the smart contract are discussed in this Chapter as well.
- Chapter 7 presents evaluation of transfer metrics. It starts with a decription of a test lab environment used to perform transfers, and continues with the discussion of the results. This Chapter also comprises a discussion of semi-decentralized validation parties.
- Chapter 8 concludes the work, and presents future work.

CHAPTER 2
Background

In this chapter, we introduce basic concepts that are used in this thesis. We start by introducing cryptocurrencies in Section 2.1, followed by an introduction of the main technology behind cryptocurrencies — blockchains (Section 2.2). In Section 2.3, we discuss special programs that can be deployed and executed on blockchains called smart contracts.

2.1 Cryptocurrencies

The invention of the Internet has changed the way people communicate, exchange information, buy and pay for goods and services, etc. When it comes to online payments, they are usually based on Web services of centralized financial institutions like banks and credit card corporations, and prices are expressed in one of the fiat currencies that are backed by governments and central banks. There have been many attempts to develop alternative solutions for traditional e-commerce that cannot be monitored and controlled by centralized authorities, and thus provide more safety and privacy for end users.

Even in the early nineties, solutions such as DigiCash have tried to tackle these problems by utilizing cryptographic protocols to create payments based on a blind signature. The main characteristic of such payments is the inability of a third-party to determine payee, time or amount of payments, but still giving users an option to provide a proof of payment [20]. The problem with this or similar solutions is that they either rely on services of commercial banks, or they require some type of a central validation entity as well. PayPal is a successful example of an alternative payment system that has grown to a multi-billion business. However, it only serves as an intermediary processor between businesses and customers, and PayPal accounts are connected to credit cards of its users.

A central verification entity is required to store transaction records, and to resolve the main issue of digital money — the double spending problem. In a digital world, creating a copy of digital money is trivial, which is the reason why double spending is the main issue where one party could create two transactions for the same money units at the same time. A centralized entity that keeps records of all transactions can easily recognize and prevent double spending, but it must be trusted by all involved parties, and represents a potential single point of failure. On the other hand, it is much harder to develop a decentralized, Byzantine fault–tolerant [30] solution where components do not have to be trusted, and can join and leave a network at their will without affecting integrity of the network. The first completely decentralized solution that is not controlled by a single entity offering at the same time anonymity and integrity of transactions was Bitcoin.

2.1.1 Bitcoin

Bitcoin was made public as an open-source project by a person, or an organization known under the pseudonym Satoshi Nakamoto in 2009, based on the research that was published in a white paper in 2008 [41]. It was proposed as a solution to the double spending problem that is based on a peer-to-peer network consisting of different types of nodes, among which the so-called miner nodes play the crucial role in the verification of transactions. In a traditional payment system, a centralized party must be trusted and verified, and any compromise of this party would lead to a disruption of the entire system. Although decentralized systems with an increased number of controlling entities solve the single point of failure issue, it introduces the Byzantine Generals problem of making a mutual agreement in an environment where some participants might be malicious.

In order to overcome the problem, participants could vote upon a correct state of a system, which would keep the system reliable as long as the majority of entities are behaving honestly. However, a malicious party could run a Sybil attack [21] by setting up multiple nodes, giving it more votes in the voting process. Bitcoin overcomes all of these difficulties with a peer-to-peer network using proof-of-work to record a public history of transactions and a consensus mechanism instead of relying on trust [22, 41].

Similar to the previously mentioned DigiCash, Bitcoin is utilizing cryptography to make pseudo-anonymous transactions that cannot be linked to their issuers. Bitcoin is not only an online payment system, but it is also a digital currency that represents an alternative to traditional fiat currencies, and since it relies on cryptographic protocols, this new type of digital currencies is called cryptocurrencies. Currently, there are thousands of different cryptocurrencies, of which some are forks of Bitcoin called Altcoins (alternative coins) with changed economics or a different consensus mechanism [22].

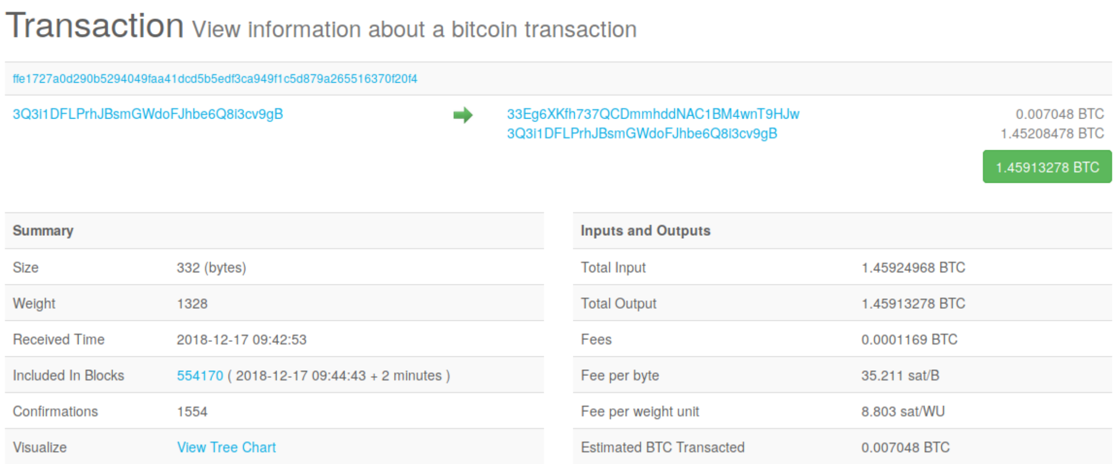
2.1.2 Transactions

Coins in cryptocurrencies like Bitcoin do not exist as such and they are not defined as some special data structure. The coins are only represented as linked transactions that show the ownership of cryptographically signed digital assets. Every transaction consists of an identifier (the hash value of the transaction), and a list of inputs and outputs. Inputs are represented as references to unspent outputs of previously executed transactions.

The Bitcoin network needs to actively keep track only of unspent transaction outputs (UTXO), since only these outputs are relevant for future transactions, whereas spent transaction outputs (STXO) can be archived [22].

Outputs are identified with an index within a specific transaction, and they show a distribution of input values to new owners. All values of outputs referenced by inputs of a transaction are always spent entirely, and they do not determine an amount spent by an initiator of the transaction. If the referenced values are higher than the amount that the initiator wanted to spend, then he can always specify an additional output that will serve as a change. A sum of all input values of a single transaction must always be equal or higher than a sum of all output values of the same transaction. The only exception from this rule is a special coinbase transaction that only has an output without any inputs [22].

Coinbase transactions can be created by validators of transactions as a reward for their effort, but they serve as a mechanism to increase coin supply over time as well. Furthermore, a difference between a sum of all input values and a sum of all output values is called a transaction fee, and it can be collected by a transaction validator [22]. More detailed descriptions of a validation process and technical details of a transaction execution are provided in subsequent sections. Figure 2.1 shows a Bitcoin transaction with the total input and output sums as well as the included transaction fee.


Transaction View information about a bitcoin transaction

ffe1727a0d290b5294049faa41dcd5b5edf3ca949f1c5d879a265516370f20f4

3Q3i1DFLPrhJBsmGWdoFJhbe6Q8i3cv9gB → 33Eg6XKfh737QCDmmhddNAC1BM4wnT9HJw 0.007048 BTC
3Q3i1DFLPrhJBsmGWdoFJhbe6Q8i3cv9gB 1.45208478 BTC

1.45913278 BTC

Summary	
Size	332 (bytes)
Weight	1328
Received Time	2018-12-17 09:42:53
Included In Blocks	554170 (2018-12-17 09:44:43 + 2 minutes)
Confirmations	1554
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	1.45924968 BTC
Total Output	1.45913278 BTC
Fees	0.0001169 BTC
Fee per byte	35.211 sat/B
Fee per weight unit	8.803 sat/WU
Estimated BTC Transacted	0.007048 BTC

Figure 2.1: Bitcoin Transaction[1]

2.1.3 Wallets and Exchanges

Users can access and spend their coins using a wallet, which stores public and private keys for cryptographic signatures of assets on a blockchain. There are various types of wallets such as software wallets, online wallets, hardware wallets, etc.

[1]https://www.blockchain.com/btc/tx/ffe1727a0d290b5294049faa41dcd5b5edf3ca949f1c5d879a265516370f20f4

Software wallets are applications that are installed directly on a user's computer, whereas online wallets are provided as Web applications by third-party entities. Since the owner of private keys controls all coins signed by these keys, it is very important how users handle their keys. If a provider of online wallets is compromised, then an attacker might gain access to coins of all users of that provider. There have been various news reports about compromised online wallet providers in which users have lost their assets (cryptocurrencies). Some online wallet providers do not store public and private keys on their servers, and the keys are only loaded on the client's side in a Web browser. Hardware wallets are special devices that are usually not always connected to the Internet, and they represent a more secure solution [22].

Online exchanges allow users to trade cryptocurrencies for fiat money or other cryptocurrencies. Similar to online wallets, online exchanges represent a high security risk as providers of these wallets might get compromised. But since cryptocurrencies are based on decentralized, peer-to-peer networks, users can use other mechanisms such as Atomic Swaps [3] to safely exchange cryptocurrencies directly between two parties in a trustless environment.

2.2 Blockchains

The core technology behind cryptocurrencies are blockchains. A blockchain is a data structure consisting of blocks that are linked to each other by hash values forming a linked list (chain). Every block contains at least, but not limited to, some data (e.g., transactions), timestamp, hash of that block and hash of the previous block [22]. Figure 2.2 shows an example of a blockchain containing some transactions. This structure of a blockchain makes it tamper-resistant, because any change of a single block would void hash values of that, and all subsequent blocks. So, in order to make changes on a blockchain, one would have to recalculate hash values of all blocks starting from the modified block. If there is only one copy of a blockchain, changing blocks and recalculating hash values would not be that hard. However, if we distribute a blockchain over a peer-to-peer network, modifying the blockchain would require a majority consensus of all involved parties. In this case, the blockchain represents a distributed ledger that is suitable for storing tamper-sensitive data such as financial transactions [41], medical records [42], etc.

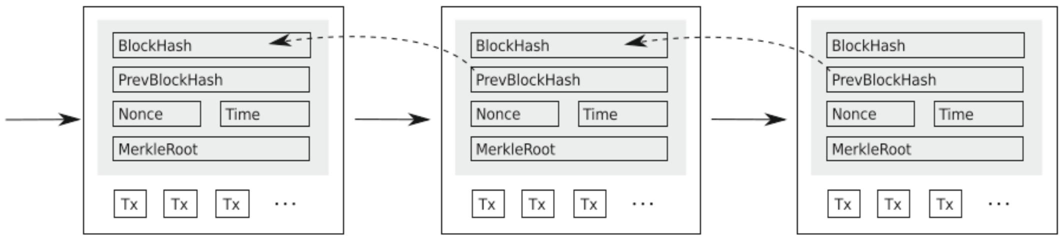


Figure 2.2: Blockchain Consisting of Blocks Linked by Hash Values [22]

2.2.1 Blocks

Technologies and concepts used for implementations of blockchains are not new, but it is the way how these technologies are combined together that makes the blockchains technology unique. For instance, hash functions are an important part of blockchains, and they have been used for decades in various fields of computer science. Hash functions are compression functions that take some data as input, and always produce an output of a fixed size. They are also called one-way functions, because it is infeasible to recalculate an input value from the respective output value. For the same input data, hash functions always return the same hash value. However, if we change only one bit of the input data, the calculated hash value will be entirely different. Although these two inputs differ by only one bit and thus can be associated with one another, their hash values will be completely different, and it is not possible to associate one output value with the other [18].

Hash functions have various applications, one of them being the usage of hash functions as a message digest. The property of hash functions that they always produce the same output if an input value is not modified, makes them suitable for validation of data integrity. For example, if we download some file over the Internet, very often the download does not take place from a single source, but various mirror sources are used. In order to validate the integrity of the file, a provider of the file often offers the hash value that users can use to make sure that the downloaded file has not been modified by a mirror source. Another usage of hash functions is to create a digital commitment. A digital commitment is the hash value resulting from applying a hash function on some commitment message and a random value that is usually called a nonce. The hash value is then published so that anybody with an access can read it. In case we want to reveal the original commitment, we can publish the message and the nonce, and anybody can now calculate the new hash value and compare it with the published one. This way the commitment that was defined in the message still holds, and it could not be modified over time.

Hashes are also used to create many data structures like hash pointers. A hash pointer is a pointer to some data that also includes the hash of the data, which can be used to check if data has been modified. Another useful data structure is a binary tree called a Merkle tree. If we have a set of data and we want to calculate a hash value of all data in the set, we can divide the data into two pairs, and then for each pair we create hash values of pair members and calculate the hash of the created hashes. If we continue to group hashes into two pairs and calculate their values, at one moment we will get only one hash value called a Merkle root. The process is depicted in Figure 2.3.

Many of the previously mentioned properties and application possibilities of hash functions are used in implementations of blockchains. As mentioned in the introduction part of this thesis, a blockchain is a linked list of blocks that is similar to a hash pointer data structure. The main difference is that a block does not only store some data and has a hash pointer, but it also contains a hash of a previous block. Since one of the main applications of a hash functions is to verify integrity of data, this structure of a blockchain makes it very hard for anyone to tamper with data stored on a blockchain, because if somebody modifies data of a block, then that block and all subsequent blocks would have different hash values. A block also contains a Merkle root of, e.g., transactions belonging to the block. Transactions can be validated without running a full network node of a blockchain. It it sufficient to keep a copy of block headers, which contain the hash of a previous block and a Merkle root. To verify that a transaction has existed at some point, we do not need all transactions, but only the transaction that we want to verify, and a list of intermediary hashes required to build a Merkle root. If the Merkle root is the same as the Merkle root saved in a block, then we can be sure that the transaction existed at that point. This verification process is known as Simplified Payment Verification (SPV) [22, 41].

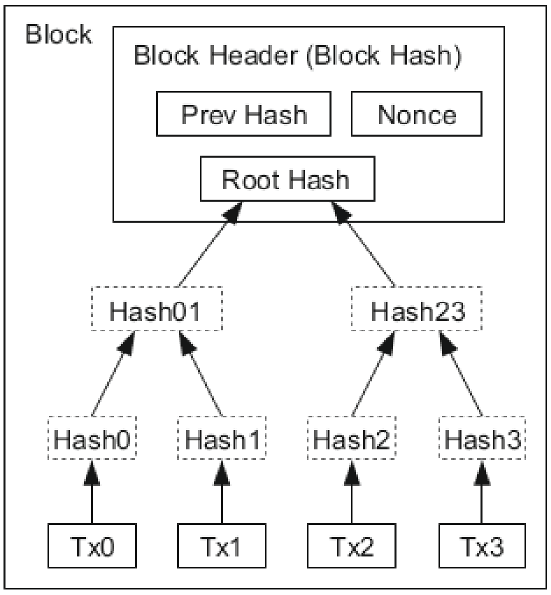


Figure 2.3: Transactions Hashed in a Merkle Tree [41]

2.2.2 Mining

In blockchains, new blocks are added to a chain through a process called mining, and nodes in a blockchain network involved in the mining process are called miners. Miners collect and validate transactions, and they need to provide some proof-of-work in order to add a new block to a blockchain. The reason for the proof-of-work is to make sure that miners are validating transactions honestly, because mining is a computational intensive task that requires a lot of computational power and thus consumes a lot of electricity. As an incentive for validating transactions and providing proof-of-work, miners get a reward (e.g., Bitcoins in case of Bitcoin mining). If a miner is not behaving rationally, he will not receive a reward, and his blocks will be rejected as invalid by other nodes on a network.

Proof-of-work is provided by solving a puzzle that is given as a search problem in which we need to find a hash value that suits some given criteria. Hash functions are puzzle friendly given that it is required to search a very large space in order to provide a puzzle solution. To provide a proof-of-work, one needs to calculate a hash value of a formed block and a nonce by modifying the nonce until the resulting value is lower or the same as some given target value. There is no other possibility to solve a mining puzzle, but to try many difference nonces making mining a very computational demanding task. By adjusting the difficulty of a puzzle, we can control how often new blocks will be added to a blockchain, and in the case of cryptocurrencies this way we also control minting of new coins and thus money supply [22].

2.2.3 Digital Signatures

Another security (cryptographic) concept used in implementations of blockchains are digital signatures. Like a handwritten signature, a digital signature ensures that only one person can sign a digital document, but anybody else is able to verify the signature of the document. Another important property of a signature is that it is always related to one specific document, meaning that we cannot take a signature of one document and apply it to another document. In computer science, signatures are realized using applied cryptography [22].

To sign a digital document, we first need to create a private and public key pair. The private key is used to sign a document resulting in a signature of the document, and only the party that is signing the document has access to the private key. The public key, on the other hand, is published and accessible to anybody who wants to verify the signature. In order to verify the signature of the document, one would need the public key, document itself and the signature. It is crucial to use a high level of randomness when generating key pairs and signing documents, otherwise somebody could be able to determine a private key from several different signatures signed by the same private key [22].

Private keys can sign only a limited amount of data (e.g., 256 bits), so they might not be convenient for signing large data. However, we can instead sign hash values of data, because as we have seen in the previous chapter, hash values always have the same output size. In blockchains, digital signatures are used as proof of ownership of digital assets stored on blockchains. Bitcoin for instance utilizes the elliptic curve digital signature algorithm (ECDSA) for transaction signatures. If one person can prove that he is the owner of a private key used to sign some transaction on a blockchain, then that transaction and respective coins belong to that person [22].

In cryptocurrencies, coins are only represented as linked transactions that show the ownership of signed digital assets. To issue a transaction that changes the ownership of one Bitcoin, we first need the Bitcoin address (e.g., public key) of the new owner. Then, the current owner needs to digitally sign the hash of a previous transaction proving that he is the current owner, and the new owner's public key. This way, we have linked transactions that we can traverse to determine owners of the respective coins. This is a very simplified description and in reality, for example, we would use obfuscated public keys for security reasons [22]. The process is illustrated in Figure 2.4.

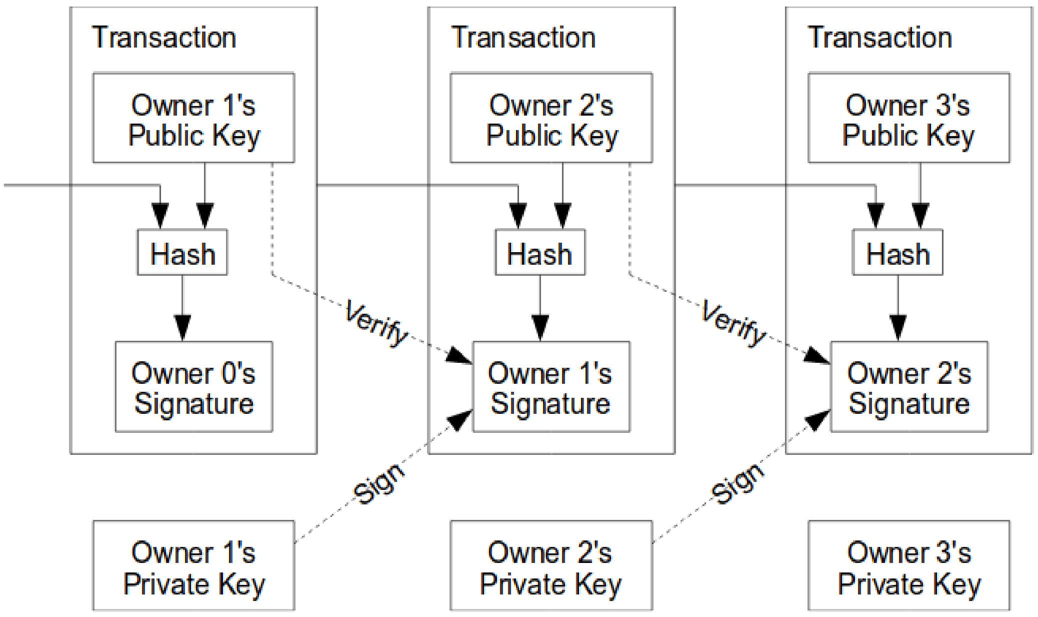


Figure 2.4: Coins: Chain of Digital Signatures [41]

2.2.4 Anonymity

As mentioned before, cryptocurrencies like Bitcoin allow making pseudo-anonymous transactions by utilizing the public key cryptography. Transactions cannot directly be linked to concrete users, but they can only be linked to addresses (hashed public keys). Thus, Bitcoin is considered to be pseudo-anonymous. However, there are various data mining techniques that can be used for a deanonymization of users. One could, for example, track the IP address of a transaction originator by controlling a hub connected to all peers, under the condition that a transaction does not come from an online wallet. In order to increase anonymity when interacting with a blockchain network, we can use anonymity services such as Tor, which utilizes an onion routing protocol to hide the IP address of a request sender. However, an attacker could disable Tor services by misusing the Bitcoin denial-of-service protection [22].

Online wallets and exchanges also reduce anonymity, because companies behind them are very often obligated to keep information about users [22].

2.3 Smart Contracts

The blockchain technology was introduced as the main technology behind cryptocurrencies, but there are other application possibilities of blockchains as well. Blockchains can be used as decentralized computer systems for running special programs for automated contractual enforcements called smart contracts [36].

The term smart contract was first coined by Nick Szabo. A smart contract is defined as a computerized transaction protocol that executes the terms of a contract. Digital cash protocols are an example of smart contracts. The general objectives of smart contract design are to satisfy common contractual conditions (e.g., payment terms), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. The economic goals of smart contracts include lowering fraud loss, arbitration and enforcement costs, and other transaction costs [39].

Bitcoin utilizes a simple stack-based scripting language that, although not Turing-complete, can be used for programming of more complex programs than only to manage balances of coins. Although programmability of Bitcoin scripts is very limited, some other blockchains like Ethereum provide a fully-fledged Turing-complete programming language that can be used to create highly sophisticated smart contracts [9, 22].

In this thesis, we will mainly refer to Ethereum-based blockchains and smart contracts for them.

2.3.1 Ethereum Smart Contracts

A smart contract runs on all nodes of a blockchain, which in the case of an Ethereum-based blockchain are provided by the Ethereum Virtual Machine (EVM). The nodes are able to execute compiled code written in the Solidity programming language that has a syntax similar to C++ and JavaScript. On all nodes, outputs and state of a smart contract must be the same. Smart contracts are isolated from the external world. They can only answer requests of distributed applications through a public interface, and communicate with other smart contracts on the same blockchain. Consequently, development of blockchains-based software systems is divided into two phases. The first phase is the specification and development of smart contracts. The second phase is the specification and development of distributed applications that communicate with users and smart contracts [36].

Smart contracts have a state that is permanently stored in a blockchain. Since blockchains are immutable, smart contracts once deployed on a blockchain cannot be removed. In Ethereum, smart contracts are deployed using special transactions to call their constructors. After the deployment, public functions of a smart contract can be called using a transaction. Message calls to a constructor or public messages that modify the state of a smart contract consume gas, which must be paid in Ethereum's native cryptocurrency called Ether. Gas consumption of a transaction depends on executed

Ethereum operations and used storage. There are no costs for calling public functions that only return a value without modifying the state of a smart contract [36].

Smart contracts have some limitations as well. For instance, the maximum contract size is limited to 24KB. Another limitation is that a smart contract cannot access the external world and it cannot autonomously trigger an action [7].

2.3.2 Smart Contracts Security

Security is a critical consideration when designing and implementing smart contracts. The reason for this is that smart contracts run in trustless and decentralized environments. In contrast to centralized solutions, vulnerabilities in smart contracts are much harder to fix. Once a bug in a smart contract is exposed, it is hard to prevent exploitation of the bug due to the decentralized nature of smart contracts. Assets stolen during such exploits are usually irrecoverable due to immutability of smart contracts [12].

A well known example of a security incident, which shows how expensive a vulnerability in a smart contract can be, is the DAO attack. A DAO is a Decentralized Autonomous Organization in which anyone can participate by purchasing DAO tokens. Governance rules of the organization are formalized, automated and enforced by a smart contract. A hacker exploited a vulnerability in the DAO smart contract and stole more than three million Ethers. Furthermore, the incident led to a fork of the Ethereum blockchain. Although developers of the DAO were already aware of the issue and they had prepared the fix for it, they were not able to patch the security flaw on time. The DAO attack demonstrates how important security aspects are for development of smart contracts [12, 14].

To develop a secure smart contract, various steps can be taken during development and operation of the smart contract. One of the steps is the definition of proper access controls. Since anyone can make a call to public functions of a smart contract, developers can apply the ownable pattern and the role-based access control to limit which entities are allowed to call these functions. An address of a smart contract owner is remembered during the creation of the smart contract. If a public function is marked with the `OnlyOwner` modifier, only function calls from the owner's address are allowed. Calls from any other address are reverted. Apart from the predefined `OnlyOwner` modifier, a role-based access control allows developers to define custom modifiers for multiple administrative accounts. Similar to the ownable pattern, only addresses which have specific roles assigned can call public functions marked with the respective modifiers [12].

To avoid centralized control of smart contracts, multi-signature accounts are another approach for defining secured access control. With a multi-signature account, developers can avoid a single point of failure if account keys are compromised or in the case of a malicious insider. These accounts require signatures from multiple entities to execute a transaction [12].

Smart contracts enforce terms of a contract. To guarantee correct behavior of smart contracts, developers can also guard function calls with the `require`, `assert` and `revert` statements. The `require` statement defines preconditions (e.g., valid user inputs, state variables, etc.) that must be met before the execution of a function. The `assert` statement is similar to the `require` statement, but it is used to check for internal errors. If such errors occur, changes to state variables are rolled back. Finally, the `revert` statement is usually used together with if-else statements to trigger exception and revert state variables if certain conditions are not satisfied [12].

Beside the code specific enforcements of the smart contract security, other steps during the smart contract development phase can be applied to ensure development of secured smart contracts. These steps include: unit testing, static and dynamic code analysis, formal verification, independent code audits, etc. [12].

Even with all previously mentioned development best practices applied, vulnerabilities can still appear after a smart contract deployment. To minimize the risk and exposure in such cases, a disaster recovery plan should be incorporated as well. One of the steps of the disaster recovery plan is to enable contract upgrades. For instance, this can be achieved with the proxy pattern. In the proxy pattern, a contract state and logic are separated in two contracts. The first contract is the proxy contract that users and other smart contracts can interact with. This contract also stores state variables and it forwards function calls to the second contract. The second contract contains logic of the smart contract, but it stores the state in the storage of the first contract. Thus, in order to upgrade the smart contract logic, one would have to deploy another logic contract preserving the state from the proxy contract [12].

Upgrading a smart contract requires time, and as it was already mentioned in the DAO attack example, the time is critical if a vulnerability in the smart contract is exposed. To guard the smart contract in such situations an emergency stop can be implemented [38]. An emergency stop is a Boolean variable, which is by default set to `false`. In case a vulnerability in the smart contract is exposed, it can be set to `true` by the contract owner or other trusted address with a specific role. To avoid the risk that a single entity can activate the emergency stop, a multi-signature account could be used. Functions of the smart contract have the `require` statement that ensures that the emergency stop is not activated before execution under normal circumstances. In case a vulnerability is exposed, it offers a fast response mechanism to disable functions execution, and thus prevent a vulnerability exploit and potential assets lost [12].

The previously mentioned best practices enable developers to build secure, robust, and resilient smart contracts. By implementing and applying these best practices in the smart contract development and operation, we can prevent vulnerabilities and reduce the risk of assets lost during security incidents.

CHAPTER 3

Related Work

In this chapter, we survey existing approaches that are related to blockchain interoperability. Not all approaches tackle the problem of cross-blockchain asset transfers, but they all aim to enable interoperability between blockchains. The work discussed in Section 3.1 shows how to enable blockchain interoperability using pegged sidechains. In Sections 3.2 and 3.3, we discuss two different solutions for Atomic Swaps. Finally, two projects that aim at cross-blockchain asset transfers are discussed in Sections 3.4 and 3.5.

3.1 Enabling Blockchain Innovations with Pegged Sidechains

Since the introduction of Bitcoin, many alternate blockchains, or altchains, have been developed, which share the codebase of Bitcoin with various modifications and extensions of blockchain features. One of the main reasons for these modifications was the very limited feature set the Bitcoin blockchain provides, which only supports transfer of a single digital currency. Although this allowed many simplifications in the implementation and increased auditability of this cryptocurrency, there were many trade-offs between scalability and decentralization, security and cost, and in technology improvements (e.g., composite signatures [17] or ring signatures [40]). Changes in the implementation are difficult to deploy, because they must be accepted and executed by all network participants. As a solution to this problem, various altchains have emerged. They allow experimentations with different blockchain technologies and economics of digital currencies [16].

However, this has also lead to a significant fragmentation of the cryptocurrency market and of blockchain infrastructure. Each altchain has its own cryptocurrency with different economics and floating price. This exposes users to the high risk and volatility of these new cryptocurrencies. Also, the risk of security vulnerabilities is much higher, because each altchain has its own technology stack that has to be maintained. This is especially a problem considering a security-specific domain of cryptocurrencies, where exploited security weaknesses can lead to total loss of assets [16].

As a solution to the fragmentation problem, the paper [16] introduces interoperable blockchains called pegged sidechains. Pegged sidechains allow the movement of digital assets from one blockchain to another. The main observation of the paper is that the Bitcoin cryptocurrency is independent from the Bitcoin blockchain, and this can be applied to any cryptocurrency or digital asset [16].

The process of transferring assets from a parent chain to a sidechain starts by creating a locking transaction on the parent chain. Then, on the sidechain, another transaction is created with a cryptographic proof that the lock on the parent blockchain was done correctly. Inputs of the second transaction contain proofs of possession and they are tagged with an asset type (e.g., the genesis hash of the parent blockchain). The transferred assets could also be moved to a third blockchain and then back to the original blockchain using the same principle [16].

In the introduction part of the thesis in Section 2.2.1, we provided descriptions of blocks, block headers and simplified payment verification (SPV). The authors of [16] additionally define block headers as a dynamic-membership multi-party signature (DMMS). The terms dynamic-membership and multi-party refer to miners, because anyone can join a network and contribute at any time without an enrollment process [16].

A DMMS is a signature of computational power of miners, and since blockheaders are chained together, DMMS is cumulative as well. Any chain fragment is a DMMS on its first block, whose computational strength is equal to the sum of computational strengths of each DMMS the chain fragment consists of [16].

An example of a cumulative DMMS is a SPV proof, which is composed of a list of block headers demonstrating proof-of-work that a transaction has been accepted in one of the blocks. SPV proofs can be used to transfer coins from a parent chain to a sidechain.

A transfer starts by sending a specific amount of coins to a SPV-locked output on a parent chain. Then, a user needs to wait for a sufficient cumulative DMMS (SPV) to be created, which contains a transaction that the coins were sent to the SPV-locked output. This wait is called the confirmation period and it typically takes one to two days. After the confirmation period, the user creates a transaction on a sidechain providing the SPV proof with enough proof-of-work that the coins are locked on the parent chain. The transferred coins on the sidechain cannot be used immediately, because SPV proofs can be invalidated by a longer chain. The user needs to wait for the contest period in order to prevent double-spending. This wait takes another one to two days [16].

The coins on the sidechain retain their identity, and they can be transferred on the sidechain independently from the parent chain. They can also be sent back to the parent chain using the same approach as described above. The transfer process is demonstrated in Figure 3.1.

The process described above is called the symmetric two-way peg, because both chains have to validate SPV proofs from the other chain. To simplify the process, sidechain nodes could be full validators of a parent chain. This would allow transfer of coins from the parent chain to the sidechain without SPV proofs, and only transfer of coins back to the parent chain would require SPV proofs validation on the parent chain. The simplified process is known as the asymmetric two-way peg.

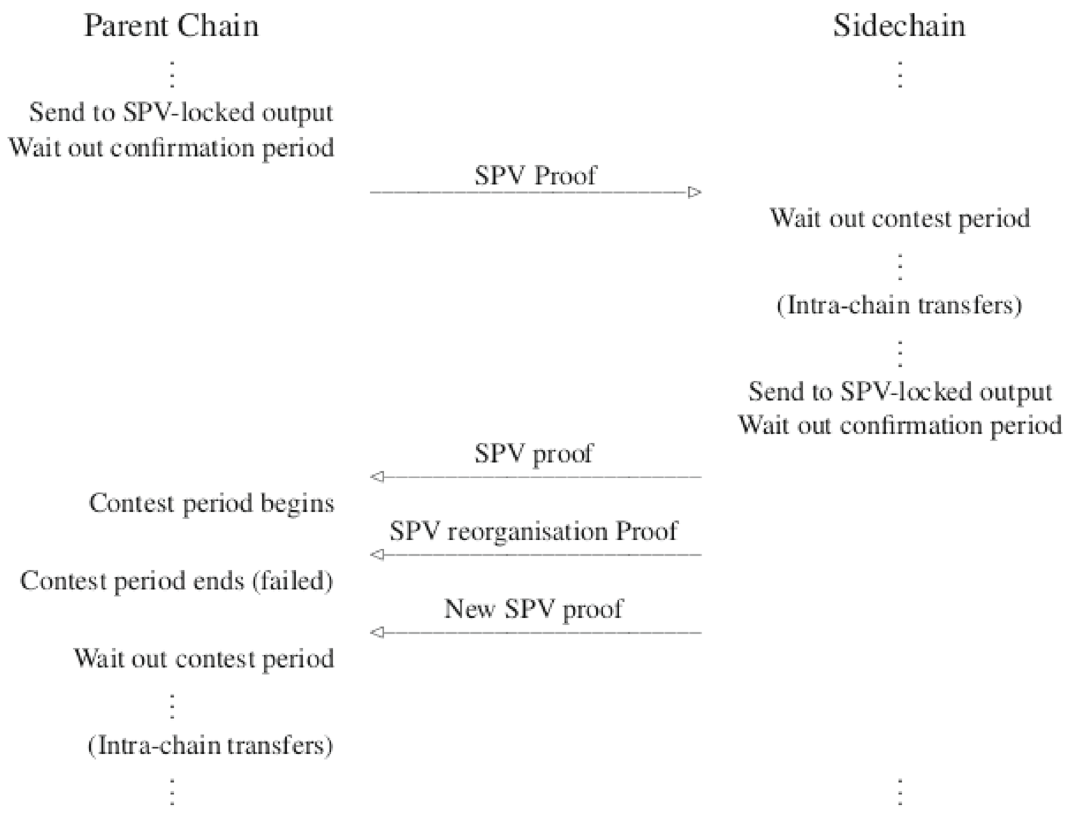


Figure 3.1: Two-Way Peg Protocol

3.2 Decred On-Chain Atomic Swaps

Blockchains and cryptocurrencies are based on a peer-to-peer communication, but in order to exchange one cryptocurrency for another, users usually rely on online exchanges. The problem with a direct exchange of coins between two parties is that there are no guarantees that both parties will fulfil their obligations. Such exchange must be enforced by a protocol so that either both parties get their coins or the exchange is not conducted at all. An Atomic Swap protocol allows two or more parties to exchange coins on different blockchains without a third party intermediary, and it guarantees by its design that each party will get its coins even in a trustless environment. Let us assume that Alice owns some Bitcoins and Litecoins, and that she wants to use Bitcoins for a payment in a Web shop that accepts only Bitcoins. However, she does not have enough Bitcoins for this transaction. One solution for this scenario would be to buy more Bitcoins on one of the online exchanges, but since Alice also owns some Litecoins, she decides to directly exchange her Litecoins for more Bitcoins. On the other side, Bob owns some Bitcoins and Litecoins as well, and he believes that the value of Litecoins will grow in the future, and he is willing to trade some of his Bitcoins for more Litecoins. Since Bob and Alice do not know and do not trust each other, they can apply an Atomic Swap protocol to make sure that they both get their new coins.

The Decred cross-chain atomic swapping project is a proof-of-concept for Atomic Swaps that supports many different cryptocurrencies (Bitcoin, Bitcoin Cash, Decred, Litecoin, Monacoin, Particl, Polis, Vertcoin, Viacoin and Zcoin). The project provides the following six commands to perform a cross-chain swap [3]:

- initiate <participant address> <amount>
- participate <initiator address> <amount> <secret hash>
- redeem <contract> <contract transaction> <secret>
- refund <contract> <contract transaction>
- extractsecret <redemption transaction> <secret hash>
- auditcontract <contract> <contract transaction>

A swap is started by executing the initiate command to create a hash-locked smart contract on the first blockchain that is also enforced by a timelock of 48 hours. The initiator needs to specify an address of the second participant and an amount of coins to be sent, and the command will generate a `Secret s` and a hash value `h` of the secret. The participant must then execute the participate command to create a hash-locked smart contract on the second blockchain using the initiator's address and the same hash value `h`, but this time with a timelock of 24 hours. After the creation of the smart contracts on both blockchains, the initiator is the first to run the redeem command to get his coins as defined in the smart contract created by the participant. The participant can then extract the `Secret s` with the extractsecret command, and redeem his coins as well. If any of the timelocks expire, respective sides can use the refund command to get back their original coins. Additionally, the auditcontract command should be used by both sides to verify the contract of the other side by checking addresses, output values, and timelock durations [3]. The workflow of a Decred cross-chain Atomic Swap is illustrated in the Figure 3.2.

Atomic Swap smart contracts are enforced with a hashlock `h` and a timelock `t`. The first enforcement, hashlock `h`, allows both parties to redeem their new assets from the contract if they send a `Secret s`, such that `h = H(s)` where `H(.)` is a hash function. The

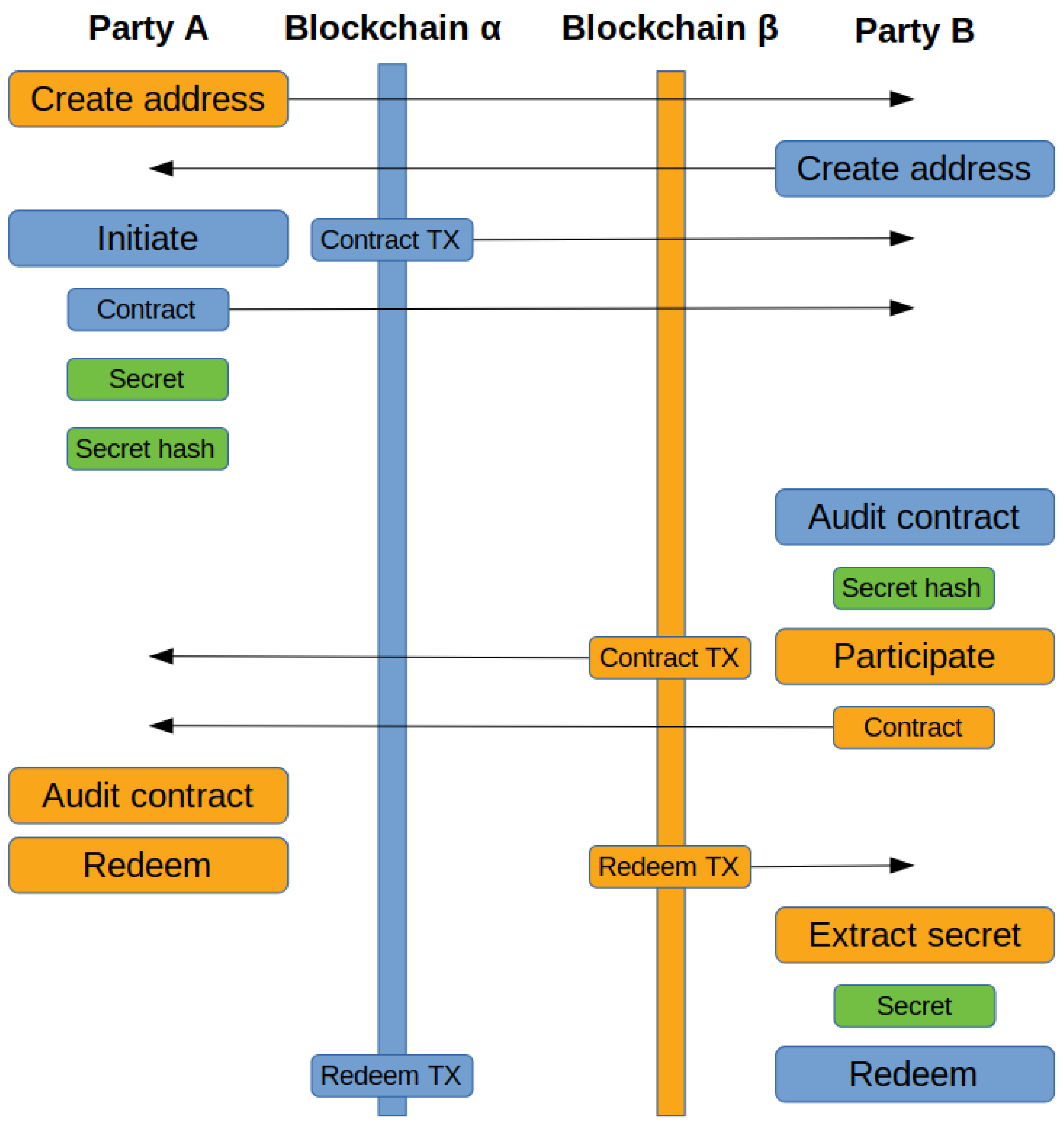


Figure 3.2: Visualization of Steps Performed During Decred Atomic Swap

second enforcement, timelock `t`, allows both parties to refund their original assets after the defined time if the actual swap was not finished successfully (e.g., `Secret s` was not revealed, or either party withdraws from the contract). Bitcoin and many altcoins support this feature due to the CheckLockTimeVerify and CheckSequenceVerify options in their transaction scripts [27].

If any side is behaving irrationally (e.g., the initiator withdraws himself after the participant deploys his smart contact on a blockchain), the other side will be able to refund his original coins after the timelock `t` expires. Also, the order of the single steps of Atomic Swaps is significant. If, for example, the initiator publishes the `Secret s` before the participant deploys his smart contract, then the participant might use the secret to redeem his new coins without giving the initiator the possibility to redeem his coins. Furthermore, if a participant's timelock is not shorter than a timelock defined by the initiator, then the initiator might wait until the very last moment of the timelock to redeem the new coins, and since in the meantime the timelock of the other contract has expired, he could use the option to refund his original coins as well. This way, the initiator gives no time for the participant to redeem his coins as defined in the contract [27].

Atomic Swaps are not limited to only two parties, and it is possible to conduct Atomic Swaps between three or more parties where once all smart contracts are published on the respective blockchains and a Secret s is revealed, all parties are able to redeem their coins [27].

The biggest disadvantage of the Decred Atomic Swap solution is that it requires some type of a side channel (e.g., instant messenger) to exchange data required as input parameters of the commands. Furthermore, it is required to manually monitor the respective blockchains in order to find out relevant contracts and transactions.

3.3 Lightning Network Off-Chain Atomic Swaps

Although the Bitcoin blockchain is suitable for storing financial transactions in a safe and verifiable way, when it comes to performance characteristics of Bitcoin as a payment system, it performs much worse compared to the traditional payment systems like Visa. For example, the Bitcoin network supports only around 7 transactions per second, whereas the Visa network supports up to 47000 transactions per second. There are numerous parameters that affect the number of transactions that can be conducted on the Bitcoin network such as computational power of miners, size of blocks on the blockchain, current difficulty of cryptographic puzzles, etc. Adjusting these parameters cannot be done without consequences. If we, for example, increase the size of blocks on the blockchain, there is a very high chance that this would cause centralization of mining nodes. This leads to a scalability problem of the Bitcoin blockchain [28].

The Lightning Network tries to solve the scalability problem of the Bitcoin blockchain by introducing off-chain payment channels that support millions of transactions per second. These payment channels operate on top of blockchains, and they do not record every transaction on the blockchains. Two parties can open a new payment channel by sending a funding transaction to a blockchain using a special wallet that signs transactions with two private keys, one owned by the first participant and one owned by the second participant. After creating the payment channel, the involved parties can exchange as many transactions as they want, where every new transaction is signed by both parties. Transactions that are exchanged only on a payment channel are called commitment transactions, and the latest commitment transaction can be used by either side to close the payment channel by sending a closing transaction to the blockchain. The closing transaction represents the final balance of all transactions from the payment channel that will be recorded on the blockchain. Although two parties might have exchanged millions of transactions, only the funding and closing transactions are actually published on the blockchain [28].

However, two parties that want to conduct transactions between each other do not necessarily have to open a new channel if they are already connected through a route of payment channels. If Alice has a payment channel with Bob, and Bob has a payment channel with Carol, then Alice can route her transactions through these two channels to Carol. This is enabled using a Hashed Time-Locked Contract that is similar to the previously mentioned hashlock and timelock contracts used to implement Decred Atomic Swaps, with the additional option to extend the duration of a channel. In this concrete example, Carol would have to generate a `Secret s` (also called a preimage in the Lightning Network), and send a hash value `h=H(s)` to Alice, who then needs to forward the hash value to Bob. Upon receiving the hash value, all parties create a timelocked contract, where the timelock period is decreasing with every new contract. Finally, Carol needs to send the secret in a reverse order until Alice receives it. All parties are motivated to activate their contracts using the secret, and send the secret to the next participant, because otherwise they might lose their assets [25].

In order to conduct Atomic Swaps in the Lightning Network, we can simply utilize a Hashed Time-Locked Contract, and do not need any additional extensions. Similar to creating a route of payment channels, we just need to construct a closed loop between parties involved in an Atomic Swap using payment channels that are opened on blockchains of different cryptocurrencies [25]. Since the Hashed Time-Locked Contract is one of the core features of the Lightning Network, this implementation of Atomic Swaps is much more sophisticated compared to the Decred Atomic Swaps implementation, and it does not require any side channels for exchanging required data like secrets, contracts and transactions. However, since this type of Atomic Swaps operates off-chain, it can be affected by potential bugs or security vulnerabilities in the Lightning Network. Also, only cryptocurrencies that are supported by the Lightning Network can benefit from this type of Atomic Swaps. Currently, only a few main cryptocurrencies are supported, and most of them are in a prototype phase at the moment.

3.4 Token Atomic Swap Technology

The "Token Atomic Swap Technology" (TAST) project is a research project which aims to develop a platform for cross-blockchain interoperability. In a series of white papers, the authors review the state of the art in blockchain technologies with a focus on blockchain interoperability, introduce and formalize the cross-blockchain proof problem, and discuss possible solutions to this problem.

The first white paper [33] provides an overview of some of the prevailing blockchains, discussing their properties like support for the User-Issued Assets (e.g., tokens) and support for smart contracts. Differences in the implementation of these blockchains make them more or less suitable for interoperability. Whereas some blockchains like Ethereum use Turing-complete smart contracts, others like Bitcoin use only a stack-based, non-Turing-complete scripting language. Achieving interoperability on blockchains without support for smart contracts poses a challenge. However, some projects (CounterParty [1], OpenAssets [10] and OmniLayer [29] create an additional layer on top of, e.g., Bitcoin that adds support for the User-Issued Assets. Transactions issued by these layers look like regular transactions to the underlying blockchain.

Apart from differences in blockchain technologies, a crucial challenge in interoperability is the cross-blockchain proof problem, which is discussed in the second white paper [32]. Using the lemma of rooted blockchains, the authors show that for two independent Blockchains A and B, there are no practical means to verify the existence of specific data (e.g., transactions) on the Blockchain A from the Blockchain B. There are two main issues that make interoperability between blockchains difficult. First is that blockchains have to support a sufficiently powerful transaction consensus of other blockchains included in an interoperability process. For instance, the Bitcoin blockchain would have to execute a smart contract instruction set. Another issue is that in order to verify the presence of some data on a single blockchain, one must traverse all blocks up to the genesis block (i.e., the root block). In case of multiple blockchains, we would have to store blocks from one chain on another chain. This is infeasible, as storage in blockchains is expensive [34].

The cross-blockchain proof problem implies that it is not possible to check if a digital asset is spent on one blockchain in order to claim it on another. This makes cross-chain execution of traditional spend-first transactions difficult. As a solution to the problem, the authors introduce the concept of claim-first transactions, in which the order of transactions is inverted. This concept uses a special CLAIM transaction that both the sender and receiver have to sign. The cryptographic signatures of the sender and receiver represent the so-called proof of intent, which allows users to claim an asset on the target blockchain. Once the CLAIM transaction is executed, anyone can participate in the execution of a SPEND transaction on the source blockchain to destroy the transferred asset. Parties responsible for submitting the SPEND transaction are called witnesses. As account balances from all supported blockchains are held on each of these blockchains, the witnesses are also responsible for updating the account balances on all involved blockchains. The witnesses get rewarded for submitting SPEND transactions and updating account balances. Deciding which witness is responsible for which SPEND transaction, and thus which witness collects the reward, poses an additional challenge. This challenge is resolved using the concept of the deterministic witness, which introduces a contest between the witnesses. The winner in this contest is determined by the lowest distance of the hash value of a witness address from the hash value of the proof of intent signature of a transaction in question [31].

CHAPTER 4

Motivating Scenarios

This chapter presents motivating scenarios and high-level requirements that we will use throughout this thesis. We start by defining the Cross-Chain Token in Section 4.1. In Sections 4.2 and 4.3, we elaborate the cross-blockchain proof problem, and a possible solution for it. Use cases of a cross-chain token transfer are presented in Section 4.4. Finally, Section 4.5 imposes requirements that will drive architecture and implementation of a solution.

4.1 Token Definition

In the blockchain field, a token is a smart contract that keeps track of mappings between user accounts and balances. Tokens can serve as a base medium for any application that requires assigning a balance to a specific user, such as cryptocurrencies or general voting rights. Since smart contracts are usually written in a Turing-complete programming language like Solidity, there are no special rules about how one needs to implement a smart contract of a token. However, to allow any wallet or distributed application to interact with tokens, additional standardization of the tokens is required [13].

The ERC20 Token Standard [23] specifies the interface for a smart contract implementation of fungible tokens. Fungibility of tokens means that all tokens are equal. The standard defines basic functions that must be provided by each interface implementation. These functions include a possibility to transfer tokens to another user (address), and to approve tokens so that they can be spent by another user. Listing 4.1 shows all functions required by the standard.

When deploying a token smart contract to a blockchain, we usually have to deploy a so-called crowdsales smart contract as well. The crowdsales define how tokens are going to be distributed to users, that is, they define the economics of the tokens. There are various properties to specify when defining a crowdsale, the most important being: price and rate, emission, capitalization, distribution, etc. A properly defined crowdsale can have a huge impact on how new tokens are accepted by users [2].

In this thesis, we present the Cross-Chain Token (XCT) that is based on and compliant with the ERC20 Token Standard, but that, in addition, can be transferred from one blockchain to another. The token is blockchain-independent, meaning that it can be deployed on multiple blockchains. Users can choose on which blockchain they are going to hold their tokens, distribute their tokens across several blockchains, and also move tokens from one blockchain to another whenever they want. For the proof of concept of this thesis, we will not implement a corresponding crowdsale smart contract. Instead, the token will be mintable, so that the contract owner can mint new tokens when required. Nevertheless, in a real-world scenario the XCT supports crowdsales similarly to any other ERC20 token.

Listing 4.1: The ERC20 Token Standard Interface

```
function name() public view returns (string);
function symbol() public view returns (string);
function decimals() public view returns (uint8);
function totalSupply() public view returns (uint256);
function balanceOf(address _owner) public view returns (
    ↪ uint256 balance);
function transfer(address _to, uint256 _value) public
    ↪ returns (bool success);
function transferFrom(address _from, address _to, uint256
    ↪ _value) public returns (bool success);
function approve(address _spender, uint256 _value) public
    ↪ returns (bool success);
function allowance(address _owner, address _spender) public
    ↪ view returns (uint256 remaining);
```

4.2 Cross-Blockchain Proof Problem

The main challenge in achieving interoperability between blockchains is the cross-blockchain proof problem discussed in [32]. The problem implies two requirements for verification on Blockchain B if data D is stored in a block of Blockchain A. The first requirement is the presence of the block lineage of A on B. The lineage of the block represents the line of descent of the block from the genesis block. The second requirement is the ability of Blockchain B to verify if the block containing D is a valid next block of its lineage. In other words, the transaction consensus of Blockchain B must be able to resemble the transaction consensus of Blockchain A [32].

The formal definition of the problem is given as the Lemma of Rooted Blockchains [32]:

For any $D \in B$, the existence of $D_{proof} \in Ca \iff D \in B$ implies (i) access to the lineage of the block containing D on A, and (ii) a sufficiently powerful transaction consensus tca to mimic tcb.

As a consequence of the cross-blockchain proof problem, it is difficult to verify on one blockchain whether a digital asset has been recorded on another blockchain. To satisfy the first requirement, we would have to store blocks from Blockchain A on Blockchain B. Furthermore, if multiple blockchains support interoperability, every single blockchain would have to store blocks from all other blockchains. In blockchains, data is already stored in a compact way, since storage is expensive [34]. This makes implementation of the requirement practically not feasible.

Even if the first requirement can be fulfilled by providing a block lineage, the transaction consensus of Blockchain B would have to be able to check if the block lineage is valid. However, blockchains use different instruction sets for validation of transactions. For instance, Bitcoin is using a simple scripting system that is not Turing-complete. Ethereum, on the other hand, is using Turing-complete virtual machines (EVM). Thus, the Bitcoin blockchain is not able to simulate instruction sets of the EVM, and consequently it is not able to check whether the block lineage from the Ethereum blockchain is valid.

4.3 Semi-Decentralized Cross-Chain Validation

In the previous section, we discussed the cross-blockchain proof problem, and why it is the main challenge in achieving blockchain interoperability. For instance, a cross-chain token transfer is practically infeasible due to this problem, since it is not possible to verify on the target blockchain if the token exists on the source blockchain.

To enable cross-chain token transfers, we propose the concept of an off-chain token transfer validation. Such transfers are initiated on the source blockchain and finalized on the target blockchain. However, the validation of the transfers is performed off-chain by validation parties that we are going to refer to as “validators”. Although the validation is performed off-chain, the validators perform it in a verifiable manner without compromising the integrity of the transferred tokens.

We call this concept semi-decentralized cross-chain validation. On the one hand, the validators operate in a centralized way which is necessary to bypass the cross-blockchain proof problem. On the other hand, the validators are parties trusted by the network of blockchains supporting this type of transfers. Furthermore, validation steps are managed and enforced by natively decentralized mechanisms of these blockchains.

The implementation of the semi-decentralized cross-chain validation poses its own challenges and limitations. Involved blockchains must support smart contracts, as the implementation of such transfers requires Turing-complete instruction sets. Another challenge is how to ensure trustworthiness of the validators. Also, the implementation must prevent double spending, in a case where some party tries to transfer the same token from one source blockchain to two different target blockchains.

4.4 Use Case Scenarios

In this section, we describe the two identified use case scenarios: a valid scenario for transferring tokens between different blockchains, and a malicious scenario where an attacker wants to transfer the same tokens from one blockchain to two different blockchains.

4.4.1 Valid Token Transfers

In case of a valid cross-chain token transfer, we always have one source blockchain where a user is holding some tokens, and one target blockchain where the user wants to send the tokens. Now, various reasons exist why users might want to transfer tokens to a different blockchain. For instance, they might want to distribute their assets across multiple blockchains in order to avoid the lock-in problem, or simply a receiving party might only have an address on a target blockchain. Nevertheless, from a technical perspective, it does not make any difference if a user just wants to move his tokens to his address on another blockchain, or if he wants to transfer the tokens to an address of a different user on another blockchain. In both cases, the balance of the user on the source blockchain is reduced for the transferred amount, and the balance of the receiving party on the target blockchain is increased for the same amount. Also, the number of available tokens on both blockchains are adjusted accordingly, and the total supply of tokens across all blockchains remains the same.

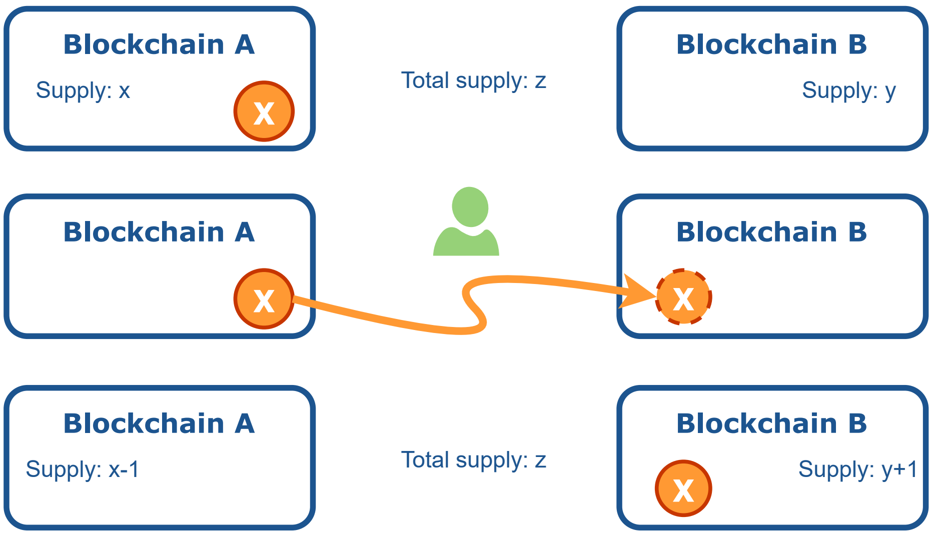


Figure 4.1: Valid Cross-Chain Token Transfer Use Case

Figure 4.1 illustrates a use case of a valid token transfer. Before the transfer, a user owns one token on Blockchain A. Furthermore, the token supply on Blockchain A and Blockchain B is x and y respectively, with the total supply on all blockchains equal to some value z. The user transfers his token from Blockchain A to Blockchain B. As result, the token is destroyed on Blockchain A and a new token is minted on Blockchain B. After the transfer, the supply of tokens on Blockchain A is reduced by one and, consequently, the supply of tokens on Blockchain B is increased by one. However, the total supply of tokens on all blockchains remains unchanged.

4.4.2 Malicious Token Transfers

Blockchains have been introduced as a solution for the double-spending problem in a trustless distributed environment. When transferring tokens from one address to another on a single blockchain, a double spending is prevented by the blockchain itself. However, by introducing a token that can reside on multiple blockchains, and that can be freely moved between these blockchains, a new type of the double-spending problem is created, namely the cross-chain double spending problem.

The cross-chain double spending problem is in essence a replay attack, where a malicious user tries to move the same tokens from one blockchain to two different blockchains, or where an attacker monitors blockchains for valid transfers and then tries to reply transfers to another blockchain using different target addresses. This misuse case will guide security decisions of the protocol design and implementation in this thesis.

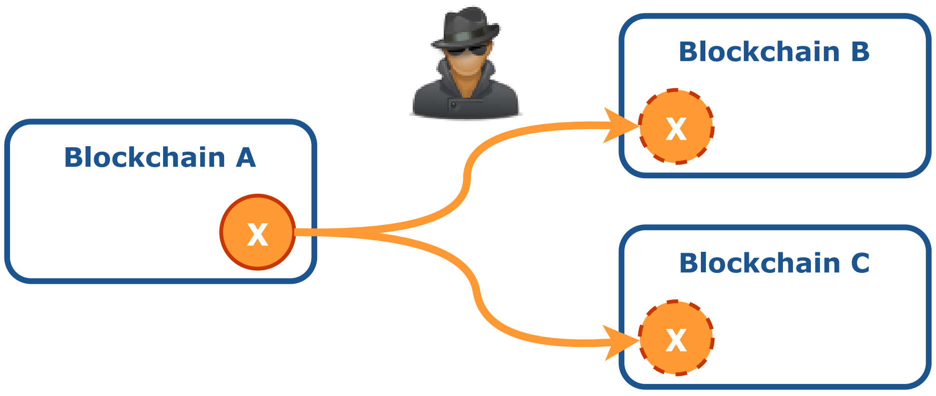


Figure 4.2: Malicious Cross-Chain Token Transfer Use Case

Figure 4.2 illustrates a use case of a malicious token transfer. A malicious user holds one token on Blockchain A and attempts to transfer it to Blockchain B and Blockchain C at the same time. This would result in destroying the token on Blockchain A and minting two new tokens (one on Blockchain B and one on Blockchain C), thus increasing the total supply of tokens by one. As Blockchain B cannot check if the token has already been transferred to Blockchain C, and vice versa, malicious token transfers like this must be prevented by the design of the transfer protocol.

4.5 Identified Requirements

In order to implement a portable and yet secure and reliable token, we identify the following high-level requirements that serve as guidelines throughout this thesis:

- All functionalities of the ERC20 Token Standard shall be supported when interacting with tokens on a single blockchain.
- Users shall be able to transfer their tokens from one blockchain to another at their will.
- Cross-chain transfers are approved by trusted semi-decentralized parties, whereas anybody shall be able to verify correctness of all transfer approvals.
- Reply attacks (double spending) of both malicious users and attackers shall be prevented by the design of a smart contract.
- The integrity of digital assets must not be compromised during a token transfer.

CHAPTER 5

Solution Architecture

In this chapter, we discuss the architecture of a solution for a cross-chain token transfer. The first section of this chapter, Section 5.1, provides an overview of components and actors of the solution. Section 5.2 describes in-depth a protocol which enables cross-chain token transfers. The architecture of distributed applications used to interact with blockchains is discussed in Section 5.3.

5.1 Overall Solution Design

In Section 4.2, we elaborated on the cross-chain proof problem. The key goal of this work is to develop a solution for the problem, which ultimately should enable a cross-chain transfer of tokens. Developing such a solution requires implementing multiple components that are used by various actors.

As mentioned in Section 2.3, we can divide the development of the solution in two phases. The first phase is specification and development of a smart contract, and the second phase is specification and development of distributed applications that communicate with users and with the smart contract.

The core component of the solution is a protocol that defines what actions and in which order must be performed by the actors, in order to perform a XCT transfer. The protocol is enforced by a smart contract, which is the central point of interaction of all involved actors. The actors interact with the smart contract via different distributed applications depending on their role.

The main idea of the solution is to perform off-chain validation of token transfers as discussed in Section 4.3. The so-called validators are responsible for these validations. Validators can only be trusted and verified parties as defined by the smart contract. Beside the validators, the solution distinguishes between two other types of roles: clients and observers. Clients can be anyone who own tokens and are interested in transferring

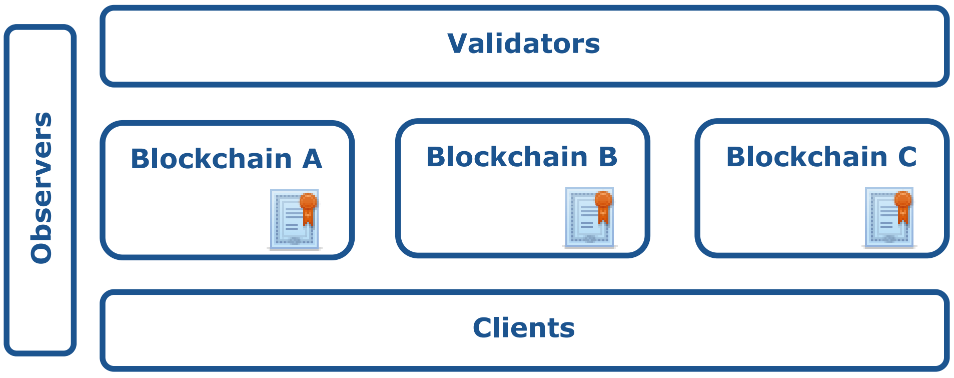


Figure 5.1: Solution Components and Actors

tokens between blockchains. Observers represent anybody who want to verify that the validators are working correctly.

Figure 5.1 shows an overview of the components and actors of the XCT transfer solution.

5.2 Cross-Chain Token Transfer Protocol

The cross-chain token transfer protocol presented in this section is a solution for the cross-chain proof problem, and it enables users to send tokens from one blockchain to another. The protocol defines steps required for the cross-chain token transfer based on the semi-decentralized cross-chain validation discussed in Section 4.3. These steps must be performed in the specific order on different blockchains. Some steps are executed by a client who wants to transfer tokens, and others by validators.

The protocol describes a distributed coordination task, which can be divided in three main phases: initialization, burn and approval. The first two phases, initialization and burn, are performed by the initiator of a transfer (client), whereas the approval phase must be performed by trusted semi-decentralized parties (validators). Furthermore, only the burn phase takes place on the source blockchain (that is, the blockchain holding tokens to be transferred). The initialization and approval phases are, in contrast, always executed on the target blockchain.

We assume that a client owns some tokens on a Blockchain A, and that he wants to transfer the tokens to a Blockchain B. Moreover, the transfer must be approved by a validator. For simplicity, we assume that at any point there is at least one validator awaiting transfer requests from supported blockchains. In the following, all three phases of the cross-chain token transfer protocol are described. The sequence diagram of the transfer phases is shown in Figure 5.2.

Transfer Initialization Phase

The initialization phase is performed by the client on the target blockchain, and it shows

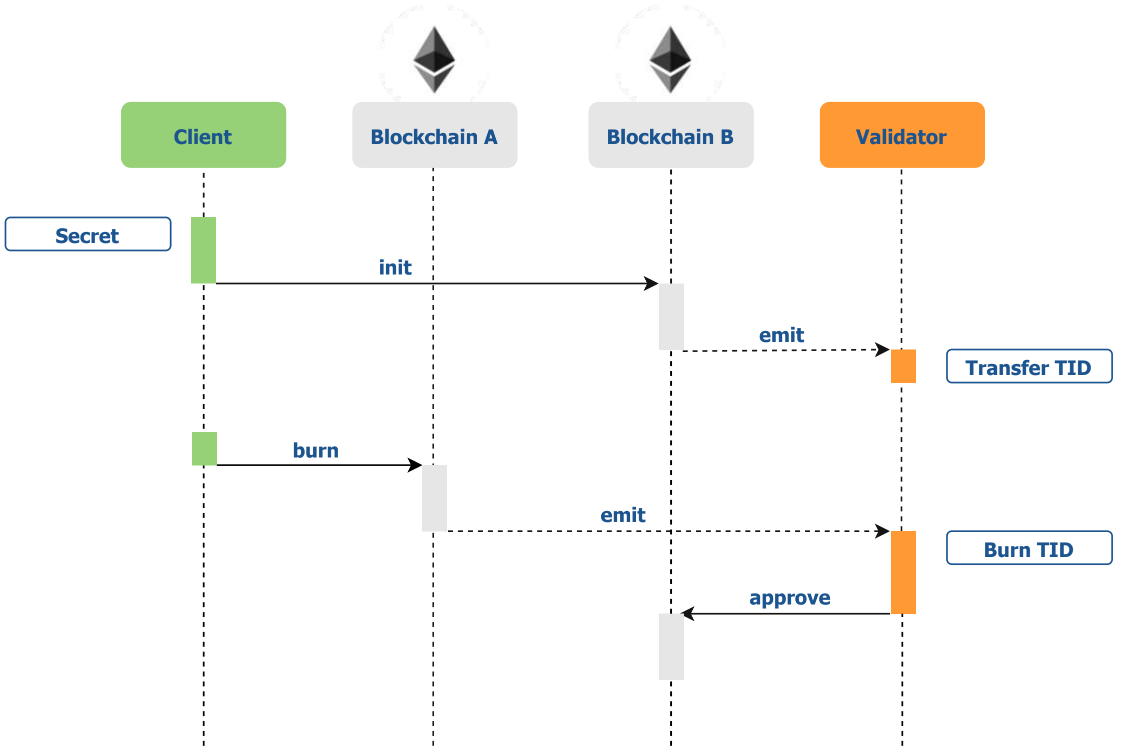


Figure 5.2: Sequence Diagram of Cross-Chain Token Transfer Protocol

an incentive of the client to move the tokens to Blockchain B. Before calling the `init` function of the XCT smart contract on the blockchain, the client first needs to create a random secret, which is used to generate a unique and deterministic identifier of the transfer that we refer to as a transfer ID (TID). The TID is defined as the hash value of a hashed random secret and a chain ID. The chain ID is introduced in EIP-155 to prevent replay attacks between Ethereum based blockchains [11]. When sending the transaction to the blockchain, the client specifies the secret, a target address to which the tokens will be sent, as well as an amount of the tokens to be transferred. The TID is generated on the blockchain from the provided secret and the chain ID already stored on the blockchain. When the execution of the `init` function is finished, the respective event containing the TID is `emitted` to validators. As the result of the transaction execution, we have a new transfer request stored on Blockchain B, however, the tokens are still not issued to the target address.

Tokens Burn Phase

The burn phase is conducted by the client as well, however, this time on the source blockchain. In essence, the client needs to provide a proof of burn proving that he has destroyed (burned) the tokens on Blockchain A. This is done by calling the `burn` function of the XCT smart contract on the blockchain. The client needs to specify the same TID generated during the transfer initialization phase, and the same amount of tokens. In the burn phase, the TID is directly provided by the client and not calculated on the chain. When the execution of the `burn` function is finished, the respective event containing the TID is `emitted` to the validators. After this phase, there is a short period of inconsistency across the blockchains, since the tokens on the Blockchain A are already burned, and although we have the transfer initialization request stored on the Blockchain B, the tokens are still not redeemed.

Transfer Approval Phase

The approval phase can only be performed by verified and trusted validators, whose pseudo identities (addresses) are stored on the involved blockchains. The validators monitor supported blockchains awaiting cross-chain transfer requests. When a validator observes the transfer initialization and burn events from both blockchains, he then validates the cross-chain transfer by checking if the amount of tokens burned on Blockchain A is equal or greater than the amount of tokens that the client wants to transfer to Blockchain B. Since the TID is unique on both blockchains, the validator can easily associate the transfer initialization request on Blockchain B with the corresponding burn request on Blockchain A. If the transfer request is valid, the validator approves the transfer by calling the `approve` function of the XCT smart contract on the blockchain. As result, the tokens specified in the initialization phase are issued to the specified address. Otherwise, if the transfer is invalid (e.g., the burn step is not executed, or the burned amount is less than the transfer amount), the validator does not take any actions, and simply waits until all steps performed by the client are valid.

Finally, as a result of the successful token transfer, the tokens on Blockchain A are irrevocably destroyed, and the total supply of tokens on this blockchain is reduced. Moreover, the total supply of tokens on Blockchain B is increased, as well as the balance of the address specified in the initialization phase. However, the total supply of the tokens on all blockchains remains identical as before the transfer.

The order of the execution of the first two phases is not significant for a successful token transfer. The client can either first initialize a transfer on Blockchain B, or instead he could first burn the tokens on Blockchain A. As soon as the validator receives both events, he will validate and approve the transfer if valid. However, it is recommended to always start with the initialization phase, because the burn phase is a destructive operation, which irrevocably destroys tokens on a source blockchain. Tokens that are burned on one blockchain can only be redeemed on a target blockchain by providing the corresponding transfer initialization request.

The protocol is designed to prevent replay attacks (double spending) by both malicious users and attackers. The first design decision, which prevents malicious users from conducting a replay attack, is the usage of a unique chain ID to generate a TID. Chain IDs are stored on blockchains and unique across well-established blockchains. In the initialization phase, the client needs to provide a previously generated secret. As part of an initialization transaction, a TID is calculated by firstly concatenating the hashed secret and the chain ID, and then applying a hash function to the resulting value. This way, a transfer request is always tied to exactly one target blockchain. If the client attempts to use the same secret on another blockchain, the resulting TID would be completely different, because the chain ID on that blockchain will not be the same as on the first blockchain. In contrast, in the burn phase the client directly provides the TID. Since this phase is destructive, users should perform it only if they know the corresponding secret and target blockchain.

The usage of a secret in the protocol has a twofold purpose. The first is to generate a unique and deterministic TID, so that a validator is able to associate transfer initialization and burn requests across blockchains. The second is to prevent an attacker from replaying a transfer of another user by creating a transfer request using the identical TID, but with a different target address. The secret is only known to the client so that only he can finalize the transfer request.

Transfers are approved by verified and trusted validators. When the XCT smart contract is deployed on a blockchain, the contract owner specifies addresses of the validators trusted by the owner. Only holders of the respective private and public cryptographic keys that are associated with the specified addresses are able to call the `approve` function of the XCT smart contract. Any other call to the function is rejected by the smart contract.

5.3 Distributed Application Architecture

Traditional distributed systems are based on client-server or peer-to-peer architectures. With the introduction of blockchains, new architectural approaches to design distributed systems have emerged. These blockchain-based systems are usually composed of a blockchain, and of an off-chain application which is able to interact with it through transactions. An off-chain application could be a standalone, serverless application operating on top of a blockchain, or it could be a part of a more complex system, where components are communicating with each other over the blockchain. This type of distributed applications is often called a dApp [36].

A dApp provides a user interface to end users, so that they do not have to interact with a blockchain directly. Moreover, users may not even be aware that they are using a system based on a blockchain [35], since a user interface could look like any other Web-based frontend, or it could be as simple as a command line tool.

A blockchain-based architecture does not tend to replace the traditional architectures of distributed systems, but instead it enriches them by providing means to store date in an immutable and tamper-resistant ledger. For instance, in a hybrid architecture [24] most business logic could still be handled in a direct communication between a client and a server, whereas only certain parts of applications that benefit from the use of blockchain technologies could send transactions to and query data from a blockchain.

5.3.1 Distributed Application Design Patterns

In case of Ethereum-based blockchains, dApps communicate with the blockchains through the web3 library which is available for many modern programming languages (e.g., Java, Python, JavaScript, etc.). The library provides an implementation of Ethereum's JSON RPC[1] client API allowing dApps to work with smart contracts and integrate with Ethereum nodes [15].

There are multiple design patterns, how a dApp based on the web3 library can establish a connection to Ethereum nodes [5]. Each transaction being sent to a blockchain needs to be signed by a user's account. These design patterns differ from each other in the way that transactions are signed and propagated to nodes, and in the way that user accounts are stored and managed.

One option, in case of a Web-based dApp, is to use a browser plugin like Metamask[2] to inject the plugin's provided web3 instance into the dApp. In this case, Metamask is responsible for managing user accounts, signing transactions and sending them to Metamask public nodes, so these features do not have to be implemented by the dApp, nor do they require any special local infrastructure. However, the main drawback of this approach is that the dApp depends on a third-party plugin that must be installed separately by users [5].

Another design pattern is to run a private Ethereum node with an unlocked account in a local infrastructure. Transactions are then sent by the dApp to the private node, where they are being signed and broadcasted to other Ethereum nodes. The JSON RPC interface of the private node should only be accessible from the local infrastructure, otherwise one could be able to misuse the unlocked account on the node [5].

The third pattern is from a development point of view probably the most challenging, since it includes implementing account management and offline signing of transactions directly into the dApp. To send a transaction to a blockchain, the dApp must first read a private key of a user's account, sign the transaction offline using the private key, and finally submit the signed transaction to a public node. Although the public node cannot modify the transaction, it could ignore the transaction by not broadcasting it to other nodes. To increase the chance of submitting the transaction to the blockchain, the signed transaction could be sent to multiple public nodes at once [5].

Combining these patterns is possible as well. For instance, a dApp could first check if there is a third-party web3 instance (e.g., Metamask) provided, and use that library if it exists. Otherwise, it could fall back to a local Ethereum node or sign transactions offline [5].

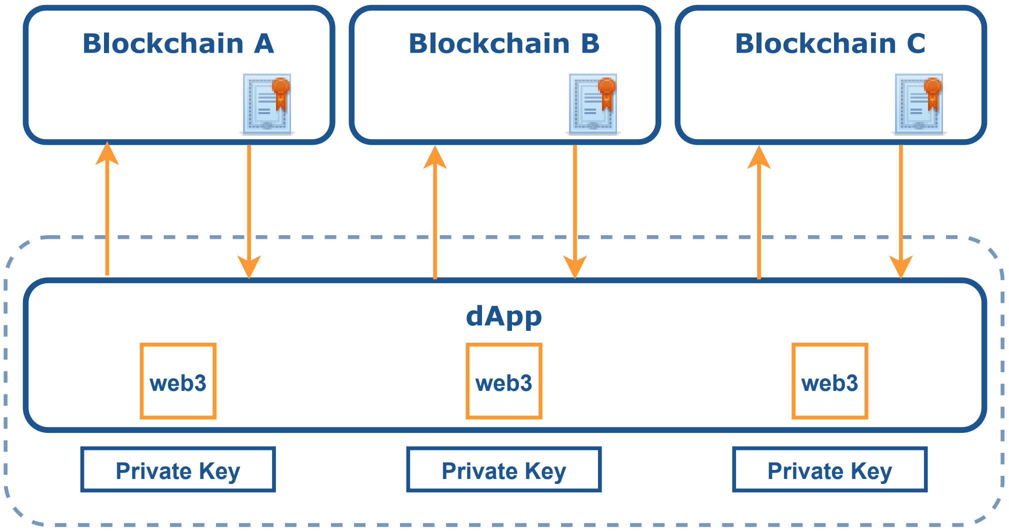


Figure 5.3: Architecture of Cross-Chain Token Distributed Application

5.3.2 Cross-Chain Token Distributed Applications

In order to successfully perform a cross-chain token transfer, two distinct distributed applications are required. The first dApp allows users to transfer tokens from one blockchain to another, whereas the second dApp can only be used by verified and trusted validators to approve transfers. Optionally, a third dApp can be implemented to demonstrate how anybody could verify the validity of transfers. This third dApp does not send any transactions to blockchains, and it only queries states of transfers.

The client and validator dApps have a very similar architecture in regard to how they interact with blockchains. Designing and implementing these dApps pose a challenge compared to other dApps, since they need to communicate with multiple blockchains, and therefore to manage multiple instances of a web3 library. The dApps must be able to asynchronously listen to events from two or more blockchains, and send transactions to these blockchains in a specific order.

In the previous section, we discussed various design patterns how dApps can interact with blockchains. For the client and validator dApps, we choose the pattern of in-application accounts management and sending offline-signed transactions to Ethereum nodes. This is illustrated in Figure 5.3. Although this approach requires much more complex codebase compared to other solutions [5], it offers at the same time the most flexibility for managing multiple instances of a web3 library. Using a third-party plugin like Metamask is not even feasible, because it provides a connection to only one Ethereum blockchain network at a time. Also, it can be cumbersome to have to run multiple private Ethereum nodes with unlocked accounts in a local infrastructure.

[1] `https://github.com/ethereum/wiki/wiki/JSON-RPC`
[2] `https://metamask.io/`

When the client and validator dApps are started, they first read blockchain network configurations and accounts, and create web3 instances for all supported blockchains. The web3 instances and accounts are identifiable by chain IDs. If a user of the client dApp wants to transfer tokens from one blockchain to another, he needs to specify a source chain ID, a target chain ID, an account on the target blockchain to which the tokens are transferred and an amount of tokens to be transferred. The client dApp first initializes the transfer on the target blockchain, and then burns the transferred tokens on the source blockchain.

The validator dApp once started does not require any interaction by users. It simply awaits burn and transfer initialization events from supported blockchains, and sends approval transactions for valid transfers. All transactions sent by validator dApps must be signed exclusively by private keys of accounts that are registered as validators on the blockchains. Otherwise, transactions are rejected by the blockchains.

CHAPTER 6

Protocol Implementation

In this chapter, we discuss implementations of components that are required to perform the steps of the XCT transfer protocol. In Section 6.1, we discuss the implementation of the XCT smart contract for Ethereum-based blockchains. Section 6.2 describes reference implementations of dApps that are able to interact with the smart contract.

6.1 Cross-Chain Token Smart Contract

One of the core components of an implementation of the cross-chain token transfer protocol is the smart contract implementation of a token. Section 4.1 provides the definition of the XCT, and in this section we will go through the most important implementation details of the token's smart contract. For the purpose of this proof of concept, we have implemented the smart contract using the Solidity programming language for Ethereum-based blockchains.

The implementation of the cross-chain token smart contract is based on the OpenZeppelin[1] implementation of the ERC20 Token Standard. The OpenZeppelin framework provides secured implementations of common smart contract libraries for various token standards, crowdsales, access control, etc. The XCT smart contract directly inherits the `ERC20`, `ERC20Detailed`, `ERC20Mintable` and `ERC20Burnable` smart contract libraries from the OpenZeppelin framework. The `ERC20` smart contract implements the base functions of the ERC20 Token Standard as defined in Section 4.1. In the original ERC20 Token Standard, the name, symbol and decimals getters are optional, so these immutable values are added to the token through the `ERC20Detailed` smart contract. The `ERC20Mintable` and `ERC20Burnable` smart contracts provide the functions to mint and burn tokens, respectively. These smart contract libraries are required since a

[1] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/ERC20.sol/

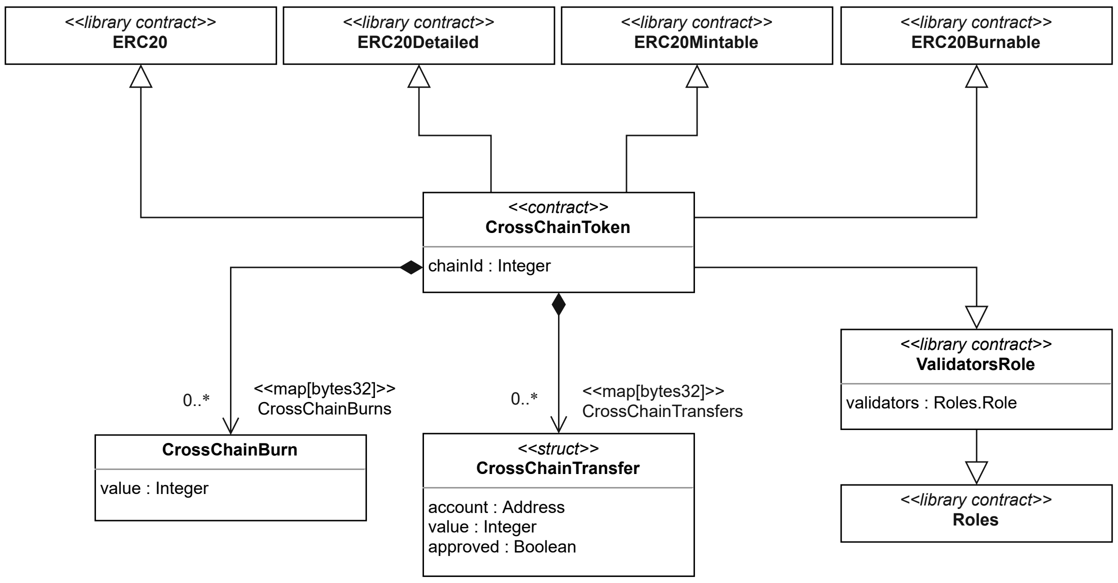


Figure 6.1: UML Class Diagram of the Cross-Chain Token Smart Contract

cross-chain transfer is a matter of burning tokens on a source blockchain followed by a subsequent minting of new tokens on a target blockchain.

The cross-chain token smart contract stores a minimum set of information, which is necessary to enable cross-chain transfers. First of all, the smart contract stores a unique chain ID. This value is immutable, and it is provided by the contract owner as a constructor parameter during a contract deployment. Furthermore, the contract stores mappings of all initialized cross-chain transfer requests as well as all cross-chain burn requests performed on the blockchain in question. The cross-chain burn requests are stored as a simple mapping between TIDs and burned amounts, whereas the cross-chain transfer requests are stored as a mapping between TIDs and the complex data structure `CrossChainTransfer` encapsulating a target account, an amount to be transferred and a flag if a transfer is approved as shown in Figure 6.1. Additionally, the contract contains a mapping of Ethereum addresses which have the validators role assigned to them. Figure 6.1 shows also the `ValidatorsRole` that is implemented as a separate smart contract, which inherits the `Roles` smart contract library from the OpenZeppelin framework. By default, the contract owner has automatically the validators role assigned, and existing validators are able to nominate new validators.

The UML class diagram of the cross-chain token smart contract is shown in Figure 6.1. This diagram is based on a modified UML class diagram [36], which has been adjusted to support smart contract specific entities (e.g., contract, struct, mapping, etc.).

The basic standard token has been extended by additional functions to enable the cross-chain portability of the token. These functions correspond to the three phases of the XCT transfer protocol specified in Section 5.2.

Table 6.1: Public Functions of the Cross-Chain Token Smart Contract

Function	Modifiers	Description
init	notNull uniqueInitRequest	Initializes a new cross-chain token transfer. **Parameters:** bytes32 secret, address to, uint256 value
burn	uniqueBurnRequest	Burns the specified amount of token to enable the cross-chain tokens transfer. **Parameters:** bytes32 tid, uint256 value
approve	onlyValidator notApproved	Approves the cross-chain token transfer. **Parameters:** bytes32 tid
transferStatusFor		Gets the approval status for the given transfer ID. **Parameters:** bytes32 tid
transferAmountFor		Gets the amount to be transferred for the given transfer ID. **Parameters:** bytes32 tid
burnedAmountFor		Gets the burned amount for the given transfer ID. **Parameters:** bytes32 tid

Table 6.2: Modifiers of the Cross-Chain Token Smart Contract

Modifier	Description
notNull	Assures that a given address is valid (not null). **Parameters:** address address
uniqueInitRequest	Makes sure that the given secret has not been used before to initialize a transfer. **Parameters:** bytes32 secret
uniqueBurnRequest	Makes sure that the given transfer ID has not been used before in a burn request. **Parameters:** bytes32 tid
onlyValidator	Allows only users having the validator role assigned to execute a function.
notApproved	Makes sure that a transfer request is not already approved. **Parameters:** bytes32 tid

The functions and their modifiers are described in Tables 6.1 and 6.2. The first function is the `init` function, which is always called on a target blockchain during the transfer initialization phase. This function is called by the client who wants to transfer tokens. The client needs to provide three parameters to the function. The first parameter is a 32 bytes long random secret, which is generated by and only known to the client, and which, together with the chain ID defined by the contract owner during the contract deployment, forms a unique and identifiable transfer request. Thus, only the client can start and finalize one specific token transfer request. The second parameter is the target Ethereum address to which the tokens will be transferred, and the third parameter represents the amount of tokens to be transferred. To make sure that the function can only be executed with valid parameters, a call to the function is enforced by two modifiers. The `notNull` modifier assures that the given target address is valid, and the `uniqueInitRequest` modifier assures that the transfer initialization request is unique by checking if the provided secret has not been used in any previous request. If parameters are valid, the function will generate a new transfer ID by first concatenating the secret and the chain ID, and then applying a hash function to the resulting value. In Ethereum, the KECCAK algorithm [26] is a well-established hash function, which is also the hash function used to generate a TID. Since the Ethereum implementation of KECCAK returns 32 bytes, the TID is 32 bytes long as well. The TID is stored in a blockchain as a key of a mapping relation to a `CrossChainTransfer` structure encapsulating basic transfer information such as target address, amount of tokens, and a flag denoting if a transfer is approved. Finally, the function emits a `CrossChainTransferInitialized` event containing the TID and the amount of tokens, thus informing validators and other interested parties that a new token transfer has been initialized.

The `burn` function is called by the client as well, only this time on a source blockchain during the token burn phase of the XCT transfer protocol. When calling the function, the client specifies the same TID as generated by the `init` function, as well as the same amount of tokens. This function calls the `burn` function of the `ERC20Burnable` smart contract library, which irrevocably destroys the specified amount of tokens from the message sender's balance. Tokens that are burned this way can only be redeemed by calling the `init` function with a corresponding secret. To assure that users do not burn their tokens multiple times by accident, the function is enforced by the `uniqueBurnRequest` modifier to make sure that the TID is unique. The function stores in a blockchain a mapping of TIDs to burned amount of tokens. Finally, it emits a `CrossChainBurnIssued` event containing the TID and the burned amount of tokens, thus informing validators and other interested parties about a new burn request.

The third function is the `approve` function, which is called on a target blockchain during the transfer approval phase of the cross-chain token transfer protocol. Whereas the first two functions can be called by anyone, only trusted validators are eligible to execute the `approve` function. This is enforced by a role-based access control, which is implemented using authorization design patterns for smart contracts [37]. The validator role smart contract shown in Listing 6.1 stores addresses of validators that are able to

Table 6.3: Events of the Cross-Chain Token Smart Contract

Event	Description
CrossChainTransferInitialized	Notifies all listeners that a new transfer has been initialized. **Parameters:** bytes32 tid, uint256 value
CrossChainBurnIssued	Notifies all listeners that a new burn request has been issued. **Parameters:** bytes32 tid, uint256 value
CrossChainTransferApproved	Notifies all listeners that a transfer has been approved. **Parameters:** bytes32 tid

approve cross-chain token transfers. As shown in Listing 6.1 in line 10, the contract owner is automatically added as the validator in the constructor of the validator role smart contract. For simplicity, existing validators can add additional validators using the `addValidator` function (line 22).

To restrict the access to the `approve` function, the `onlyValidator` modifier from the validator role smart contract (line 13) is used to check whether an address of a function's caller has the validator role assigned. When calling the `approve` function, a validator only specifies a TID. The TID is used to identify a transfer request that is stored in a blockchain during the transfer initialization phase. Another function's modifier (`notApproved`) makes sure that the transfer request is not already approved. If all modifiers are successfully passed, the function marks the transfer request as approved and issues tokens to the address specified in the transfer request. The tokens are issued by calling the `mint` function of the `ERC20Mintable` smart contract library. Finally, a `CrossChainTransferApproved` event is emitted to all subscribers. The events that are published by the XCT smart contract and awaited by validators are listed in Table 6.3.

Apart from the functions described above, which are required for performing a XCT transfer, the smart contract also provides the following functions to query a current state of a transfer with a specified TID: `transferStatusFor`, `transferAmountFor` and `burnedAmountFor`. The functions are listed in Table 6.1 as well. These functions only return requested values, and they do not modify the state of the smart contract.

Listing 6.1: Smart Contract for Role Based Access Control

```
contract ValidatorRole {
    using Roles for Roles.Role;

    event ValidatorAdded(address indexed account);
    event ValidatorRemoved(address indexed account);

    Roles.Role private _Validators;

    constructor () internal {
        _addValidator(msg.sender);
    }

    modifier onlyValidator() {
        require(isValidator(msg.sender));
        _;
    }

    function isValidator(address account) public view
        ↪ returns (bool) {
        return _Validators.has(account);
    }

    function addValidator(address account) public
        ↪ onlyValidator {
        _addValidator(account);
    }

    function renounceValidator() public onlyValidator {
        _removeValidator(msg.sender);
    }

    function _addValidator(address account) internal {
        _Validators.add(account);
        emit ValidatorAdded(account);
    }

    function _removeValidator(address account) internal {
        _Validators.remove(account);
        emit ValidatorRemoved(account);
    }
}
```

6.2 Distributed Applications Implementation

For the proof of concept of a XCT transfer, we implemented the client, validator and monitor distributed applications, which are based on the architecture discussed throughout Section 5.3.2. All applications are implemented as command line applications in the Java 8 programming language, and therefore they utilize the Java version of the web3 library (web3j[2]). In this section, we discuss the most relevant implementation details of these applications.

The monitor dApp subscribes to events that are published by XCT smart contracts from supported blockchains. The dApp is not required to successfully perform a transfer. Instead, it only demonstrates how observer actors can monitor the blockchains to verify if validators are working correctly. When an event occurs on the blockchains, it simply displays the event details on a monitor keeping track of which event occurred on which blockchain.

Listing 6.2: Initialization of Client Distributed Application

```
1    private Map<Integer, CrossChainToken> contractWrappers;
2
3    public void init() {
4       int chainId = 1;
5       while (env.containsProperty("app.bc" + chainId + ".url
          ↪ ")) {
6          Web3j web3 = Web3j.build(new HttpService(env.
              ↪ getProperty("app.bc" + chainId + ".url")));
7          Credentials credentials = Credentials.create(env.
              ↪ getProperty("app.bc" + chainId + ".private.key
              ↪ "));
8          CrossChainToken wrapper = CrossChainToken.load(env.
              ↪ getProperty("app.bc" + chainId + ".contract.
              ↪ address"), web3, credentials, new
              ↪ DefaultGasProvider());
9
10         contractWrappers.put(chainId, wrapper);
11         web3s.add(web3);
12         chainId++;
13      }
14   }
```

The client dApp provides a user the functionality to transfer XCT tokens. The dApp initialization is shown in Listing 6.2. Upon initialization of the client dApp, the application first iterates over environment configurations and reads the blockchain location (line 6), the account credential (line 7) and the address of a XCT contract (line 8) for

[2]https://github.com/web3j/web3j/

all supported blockchains. For each supported blockchain, as shown in line 10, the application creates a new `Web3j` object, which is configured to communicate with exactly one specific blockchain. The `Web3j` object is then encapsulated by an instance of the `CrossChainToken` class (line 12), which is a wrapper class that provides methods to call the functions of a cross-chain token contract. The dApp stores a map of these `CrossChainToken` wrappers that can be identified by the respective chain ID.

Once started, the dApp awaits a transfer command in the following format:

```
transfer <sourceChainId> <targetChainId> <address> <value>.
```

The transfer command is shown in Listing 6.3. For each command, as shown in lines 2 to 4, the application generates a new random secret and the corresponding TID. Depending on the `sourceChainId` and `targetChainId` parameters, the application issues the transfer initialization transaction on the target blockchain (line 6), and the burn transaction on the source blockchain (line 7), using the methods of the respective `CrossChainToken` wrappers.

Listing 6.3: Transfer Method of the Client dApp

```
1       public void transfer(final TransferRequest request) {
2           byte[] secret = new byte[32];
3           new Random().nextBytes(secret);
4           byte[] tid = Hash.sha3(Bytes.concat(secret, Numeric.
              ↪ toBytesPadded(BigInteger.valueOf(request.
              ↪ getTargetChainId()), 32)));
5
6           contractWrappers.get(request.getTargetChainId()).init(
              ↪ secret, request.getAddress(), request.getValue())
              ↪ .send();
7           contractWrappers.get(request.getSourceChainId()).burn(
              ↪ tid, request.getValue()).send();
8       }
```

Similarly to the client dApp, the validator dApp also first reads environment configurations and initializes `CrossChainToken` wrappers for all supported blockchains. Furthermore, the application subscribes to the `CrossChainTransferInitialized` and `CrossChainBurnIssued` events discussed in Section 6.1. By subscribing to these events, the application is asynchronously listening to new XCT transfer requests.

The validator dApp does not require any interaction from a user and is completely autonomous. It operates as a state machine for a received event. On each request, the `validate` method shown in Listing 6.4 is called, which is responsible for approving cross-chain transfers. As shown in line 2, if the received event is of the type `CrossChainTransferInitialized`, then the method first checks if there is already a corresponding `CrossChainBurnIssued` event (line 3). In case that such an event exists, the method verifies that the transferred amount is the same as the burned amount (line 4) and approves the valid transfer (line 5). Otherwise, if the corresponding `CrossChainBurnIssued` event does not exist, the event is simply stored in-memory (line 9). The same logic is applied if the newly received event is of the type `CrossChainBurnIssued` (line 11). Only this time, the method checks if there is a corresponding `CrossChainTransferInitialized` event already issued (line 13). The validation code remains the same.

Listing 6.4: Validate Method of the Validator dApp

```
 1        private synchronized void validate(String type, int
              ↪ chainId, String tid, BigInteger value) {
 2            if (type.equals(FUNC_INIT)) {
 3                if (burnRequests.containsKey(tid)) {
 4                    if (value.equals(burnRequests.get(tid))) {
 5                        contractWrappers.get(chainId).approve(
                              ↪ BaseEncoding.base16().lowerCase().decode
                              ↪ (tid)).send();
 6                        burnRequests.remove(tid);
 7                    }
 8                } else {
 9                    transferRequests.put(tid, new InitRequest(
                          ↪ chainId, value));
10                }
11            } else if (type.equals(FUNC_BURN)) {
12                if (transferRequests.containsKey(tid)) {
13                    if (value.equals(transferRequests.get(tid).
                          ↪ getValue())) {
14                        contractWrappers.get(transferRequests.get(tid
                              ↪ ).getChainId()).approve(BaseEncoding.
                              ↪ base16().lowerCase().decode(tid)).send()
                              ↪ ;
15                        transferRequests.remove(tid);
16                    }
17                } else {
18                    burnRequests.put(tid, value);
19                }
20            }
21        }
```

For simplicity, the dApp only considers transfer requests that are issued after the application is started, and we assume that at each moment there is at least one validator awaiting cross-chain transfers. However, since all requests are stored on blockchains as well, future extensions of the dApp may include an option to consider also transfers issued prior to the startup of the application. Also, it is sufficient to store received events only in-memory, since they are stored on blockchains, and can be queried from these blockchains on demand.

CHAPTER 7

Evaluation and Analysis

In this chapter, we evaluate the XCT transfer protocol and the proof of concept implementation of the protocol described in Chapter 5 and 6, respectively. We start by describing a test lab environment that we used to perform transfers. In Section 7.2, we evaluate basic use case scenarios described in Section 4.4. Section 7.3 provides the evaluation of the transfer metrics. Finally, Section 7.4 discusses the limitations of semi-decentralized validation parties.

7.1 Lab Environment

A mainnet is an official network of a blockchain that requires spending a native currency (e.g., Ether in the case of the Ethereum blockchain) to use it. This makes the mainnet inadequate for testing a smart contract, especially since bugs in smart contracts can be expensive as we have already discussed in Section 2.3. Furthermore, for the evaluation of cross-chain token transfers, we require at least two Ethereum-based blockchains.

As an alternative to the mainnet, local blockchains or testnets could be utilized for testing smart contracts. During the development of the proof of concept of cross-chain token transfers in this thesis, we deployed and tested the smart contract on a Ganache local blockchain. On local blockchains, Ether can be acquired in unlimited amounts and blocks are mined instantly. Optionally, we can artificially define mining time in seconds, for instance, to match the average mining time on the Ethereum mainnet of 15 seconds [4].

Currently, there are multiple Ethereum testnets of which the most popular are: Rinkeby, Goerli, Kovan and Ropsten[1]. The first three testnets are proof-of-authority blockchains,

[1]End of 2022 the Ethereum mainnet has been transformed from a proof-of-work to a proof-of-stake consensus mechanism [8]. As a consequence of this transformation, Ropsten has been deprecated as of December 2022. Before the transformation, Ropsten was similar to the proof-of-work mainnet. The evaluation of the proof of concept of this thesis was performed before the transformation.

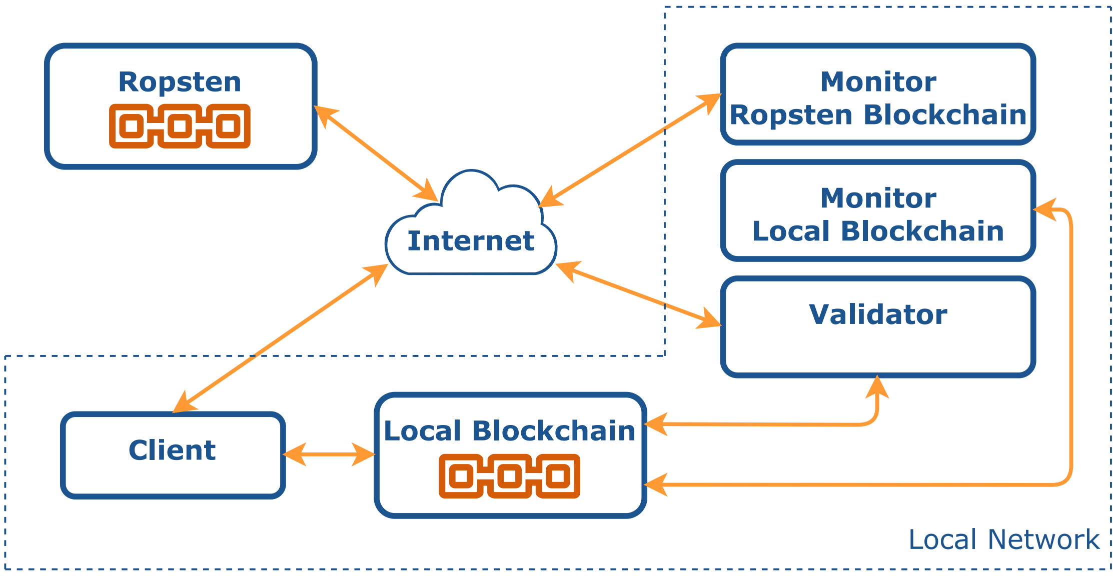


Figure 7.1: Lab Environment

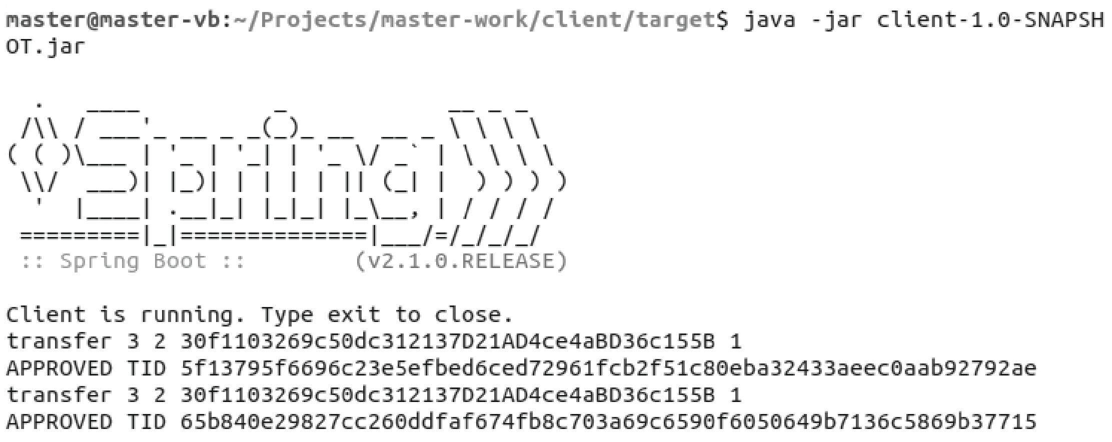
whereas Ropsten is the only proof-of-work blockchain that closely resembles the Ethereum mainnet. Ether for these blockchains is free, but must be requested from a faucet.

For the evaluation phase of this thesis, we conduct cross-chain token transfers between the Ropsten testnet and a local blockchain. We choose the Ropsten testnet as it is the most similar to the Ethereum mainnet. And since we need a second Ethereum-based blockchain as well, we decided to use a local blockchain as it is easy to deploy and to acquire Ether on this type of blockchains.

To demonstrate the feasibility of the cross-chain token transfer protocol introduced in Section 5.2, we prepare a test environment consisting of the Ropsten testnet, the Ganache local blockchain and the dApps implemented and discussed in Section 6.2. Apart from the Ropsten testnet, all other components of the test environment are deployed on a local Linux-based virtual machine.

The Ganache local blockchain is configured to mine one block every 15 seconds. We deploy the XCT smart contract introduced in Section 6.1 to both the Ropsten and local blockchains. Furthermore, we deploy one validator, one client and two monitor dApps. The validator is configured to listen for transfer initialization and burn requests from both blockchains, and perform approvals of valid transfers. We use the client application to initiate token transfers, whereas the two monitor applications observe events from the blockchains.

Figure 7.1 illustrates the lab environment.



```
master@master-vb:~/Projects/master-work/client/target$ java -jar client-1.0-SNAPSH
OT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.0.RELEASE)

Client is running. Type exit to close.
transfer 3 2 30f1103269c50dc312137D21AD4ce4aBD36c155B 1
APPROVED TID 5f13795f6696c23e5efbed6ced72961fcb2f51c80eba32433aeec0aab92792ae
transfer 3 2 30f1103269c50dc312137D21AD4ce4aBD36c155B 1
APPROVED TID 65b840e29827cc260ddfaf674fb8c703a69c6590f6050649b7136c5869b37715
```

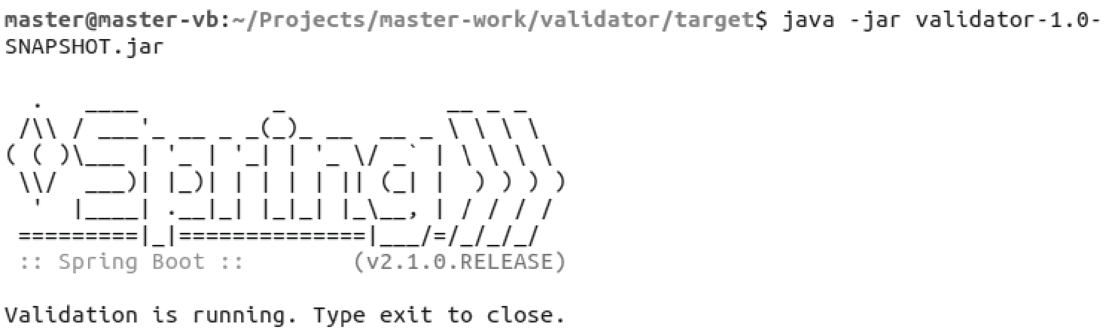
Figure 7.2: Client Console Output

7.2 Evaluated Use Case Scenarios

In Section 4.4, we introduced use cases of valid and malicious cross-chain token transfers. In this section, we present in detail the execution of a valid cross-chain token transfer in the lab environment.

The lab environment is deployed as described in the previous section. Before any transfer, we mint 1000 XCT tokens to the account `E7AE6cb3f69d070ea774CAFaD8d99754 65c1200a` on the Ropsten blockchain. In the execution of the valid use case scenario, we want to transfer a XCT token from this account on the Ropsten blockchain to the account `30f1103269c50dc312137D21AD4ce4aBD36c155B` on the local blockchain. The account on the local blockchain does not have any tokens before the execution, meaning also that the total supply of XCT tokens on both blockchains is 1000 tokens.

Using the client dApp, we initiate a transfer of one XCT token from the Ropsten blockchain to the local blockchain. For this we use the client's `transfer` command and specify the source blockchain (chain ID 3 for the Ropsten), the target blockchain (chain



```
master@master-vb:~/Projects/master-work/validator/target$ java -jar validator-1.0-
SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.0.RELEASE)

Validation is running. Type exit to close.
```

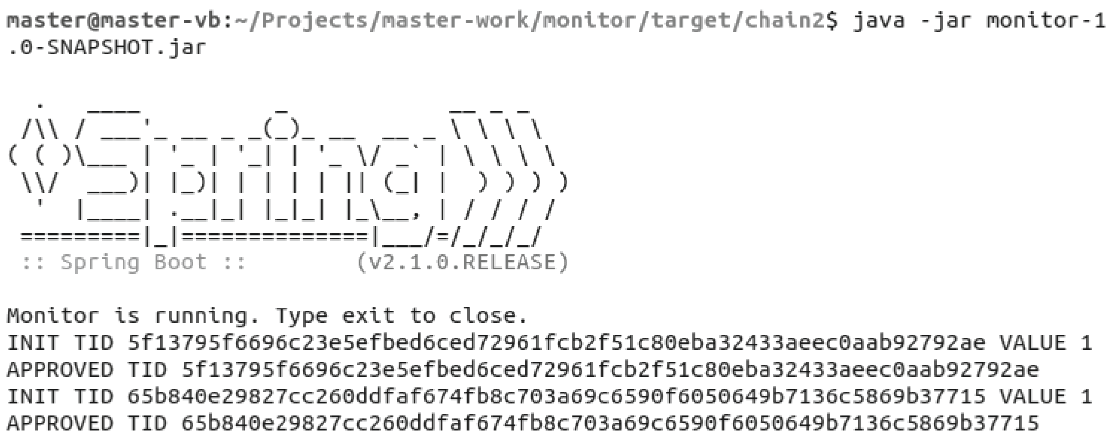
Figure 7.3: Validator Console Output

ID 2 for the local blockchain), the target account (30f1103269c50dc312137D21AD4ce4aBD36c155B), and the amount of tokens that we want to transfer (1 in this case).

Figure 7.2 shows the client console output.

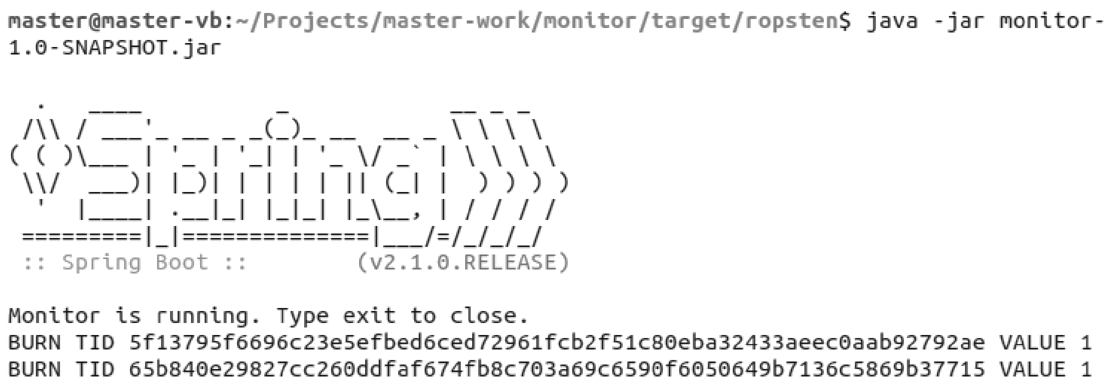
The client initializes the transfer by sending an INIT transaction to the target blockchain and a BURN transaction to the source blockchain. The local blockchain monitor dApp records and outputs the INIT events with the corresponding TID. Similarly, the Ropsten blockchain monitor dApp records and outputs the BURN events with the same TID. The console outputs of the monitor dApps are shown in Figures 7.4 and 7.5.

The validator dApp awaits new transfers and approves them without any user interactions. Since the validator has to approve the transfer on the target blockchain, the local blockchain monitor dApp records and outputs the APPROVE events as well. After the approval of the transfer by the validator dApp, the client gets notified about the successful transfer. Figure 7.3 shows the validator console output.



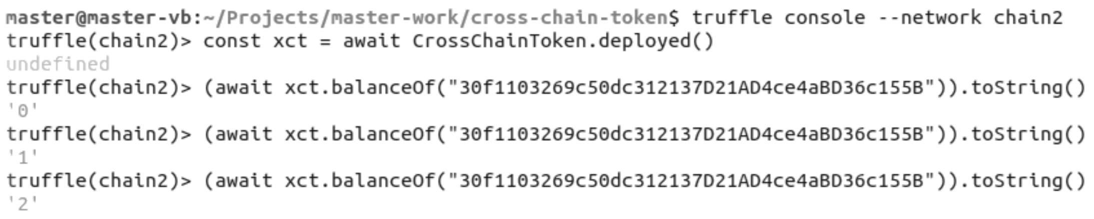
```
master@master-vb:~/Projects/master-work/monitor/target/chain2$ java -jar monitor-1
.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.0.RELEASE)

Monitor is running. Type exit to close.
INIT TID 5f13795f6696c23e5efbed6ced72961fcb2f51c80eba32433aeec0aab92792ae VALUE 1
APPROVED TID 5f13795f6696c23e5efbed6ced72961fcb2f51c80eba32433aeec0aab92792ae
INIT TID 65b840e29827cc260ddfaf674fb8c703a69c6590f6050649b7136c5869b37715 VALUE 1
APPROVED TID 65b840e29827cc260ddfaf674fb8c703a69c6590f6050649b7136c5869b37715
```

Figure 7.4: Local Blockchain Monitor Console Output



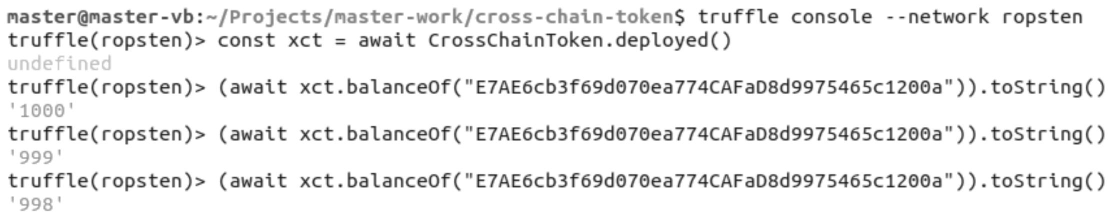
```
master@master-vb:~/Projects/master-work/monitor/target/ropsten$ java -jar monitor-
1.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.0.RELEASE)

Monitor is running. Type exit to close.
BURN TID 5f13795f6696c23e5efbed6ced72961fcb2f51c80eba32433aeec0aab92792ae VALUE 1
BURN TID 65b840e29827cc260ddfaf674fb8c703a69c6590f6050649b7136c5869b37715 VALUE 1
```

Figure 7.5: Ropsten Blockchain Monitor Console Output



```
master@master-vb:~/Projects/master-work/cross-chain-token$ truffle console --network chain2
truffle(chain2)> const xct = await CrossChainToken.deployed()
undefined
truffle(chain2)> (await xct.balanceOf("30f1103269c50dc312137D21AD4ce4aBD36c155B")).toString()
'0'
truffle(chain2)> (await xct.balanceOf("30f1103269c50dc312137D21AD4ce4aBD36c155B")).toString()
'1'
truffle(chain2)> (await xct.balanceOf("30f1103269c50dc312137D21AD4ce4aBD36c155B")).toString()
'2'
```

Figure 7.6: Local Blockchain Truffle Console Output



```
master@master-vb:~/Projects/master-work/cross-chain-token$ truffle console --network ropsten
truffle(ropsten)> const xct = await CrossChainToken.deployed()
undefined
truffle(ropsten)> (await xct.balanceOf("E7AE6cb3f69d070ea774CAFaD8d9975465c1200a")).toString()
'1000'
truffle(ropsten)> (await xct.balanceOf("E7AE6cb3f69d070ea774CAFaD8d9975465c1200a")).toString()
'999'
truffle(ropsten)> (await xct.balanceOf("E7AE6cb3f69d070ea774CAFaD8d9975465c1200a")).toString()
'998'
```

Figure 7.7: Ropsten Blockchain Truffle Console Output

As shown in Figures 7.6 and 7.7, we performed two transfers of one XCT token per transfer from the Ropsten blockchain to the local blockchain. To check the balances of the accounts before and after the transfers, we used the truffle console to call the cross-chain token smart contract. After the transfers, the account `30f1103269c50dc312137D21AD4ce4aBD36c155B` on the local blockchain now has 2 XCT tokens. Consequently, the balance of the account `E7AE6cb3f69d070ea774CAFaD8d9975465c1200a` on the Ropsten blockchain is reduced from 1000 to 998 XCT tokens. Furthermore, the total supply of XCT tokens on both blockchains remains the same after the transfers (1000 XCT tokens).

The truffle console outputs are shown in Figures 7.6 and 7.7.

7.3 Token Transfer Metrics

For the evaluation of the transfer metrics of the implemented proof of concept, we measure duration and transfer cost of the three phases of the protocol (`INIT`, `BURN` and `APPROVE`). We perform a total of 50 transfers, 25 transfers from the local blockchain to Ropsten and 25 transfers from Ropsten to the local blockchain. However, we only evaluate metrics of transactions executed on Ropsten as they are closer to the real-world scenario since the local blockchain is running in the same environment as other components (e.g., dApps). The measured values can be found in the Appendix A of the thesis.

The duration is measured from the moment a client sends a transaction until the corresponding event from a blockchain is received by the client. When measuring the duration, we do not include a block confirmation time as this may vary depending on the current underlying metrics of a blockchain (e.g., mining complexity, block size, etc.). We measure the transfer cost of these transactions as well. The transfer cost expresses the

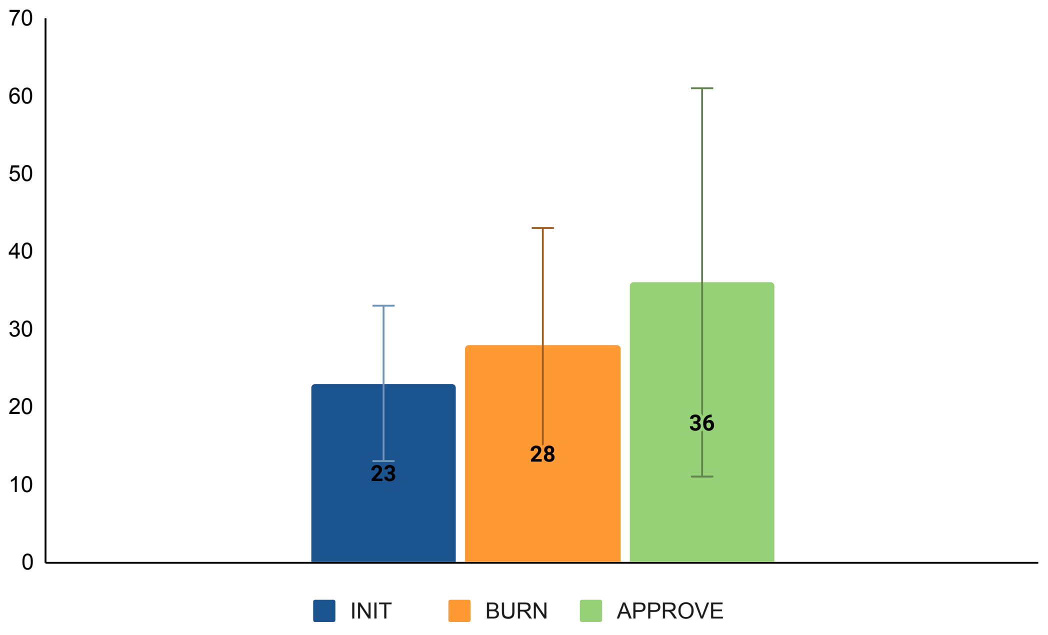


Figure 7.8: Average Transaction Duration

gas consumption of transactions behind each of the three phases executed on blockchains during the transfer.

Since the `INIT` transaction represents a client intention to transfer a token, and only generates and stores a TID on a blockchain without interacting with XCT, it achieves the shortest average duration of 23 seconds (standard deviation of 10 seconds). The `BURN` transaction, which reduces the balance of an owner, requires on average 28 seconds (standard deviation of 15 seconds). The average duration of the `APPROVE` transaction is 36 seconds (standard deviation of 25 seconds). This transaction requires the highest amount of time. It stores a confirmation of a successful transfer and mints new tokens to the owner. Figure 7.8 shows the average duration for each transaction type.

As gas consumption depends on executed Ethereum operations and storage used during a transaction as mentioned in Section 2.3.1, the three transaction types of the protocol consume almost a consistent amount of gas. In Section 6.1 we discuss the implementation details of the XCT smart contract. We showed that the `INIT` transaction stores mappings between a TID and the complex data structure `CrossChainTransfer`, whereas the `BURN` transaction stores only a mapping between a TID and the amount of burned tokens. Consequently, the `INIT` consumes more gas, 74.94 kGas, compared to the `BURN` transaction which requires 58.62 kGas. The gas consumption of the `APPROVE` transaction, that is responsible for minting new tokens, is about 67.41 kGas.

Figure 7.9 shows the average gas consumption for each transaction type.

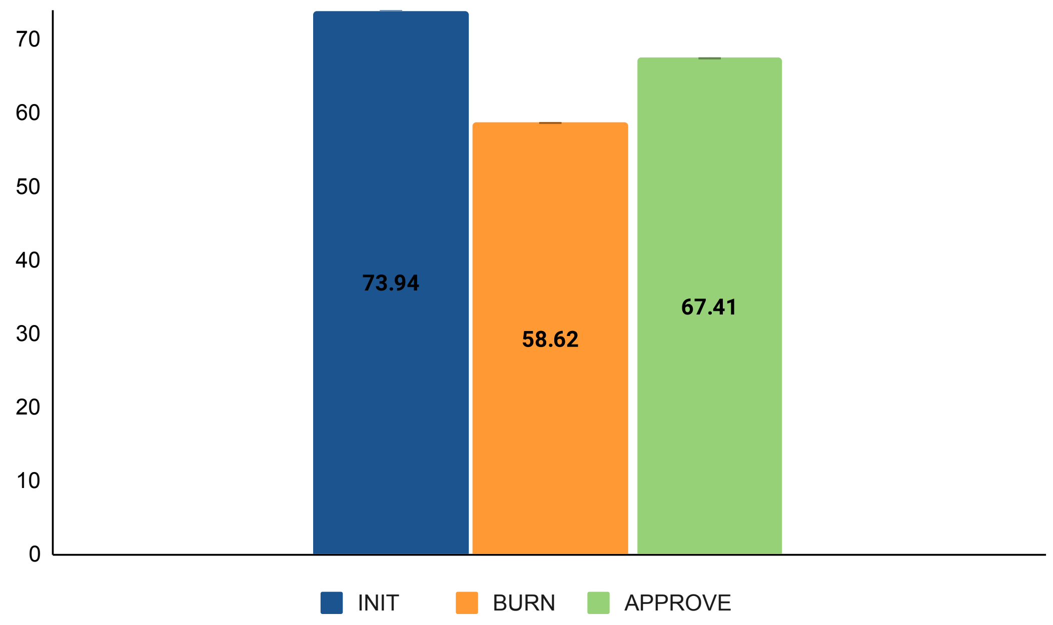


Figure 7.9: Average Transaction Gas Consumption

7.4 Discussion of Semi-Decentralized Validation Parties

The main issue with the concept of semi-decentralized validation parties is that it is not entirely compatible with the main blockchain design principle, which is the complete decentralization. Validations and approvals of transfers occur off-chain by semi-decentralized parties — validators. However, the validation process and steps are enforced by the XCT smart contract. For instance, the XCT smart contract enforces which accounts are able to approve transactions. In the proof of concept, a contract owner specifies these accounts, and validators are continuously running as background applications. Furthermore, anyone can verify the trustworthiness of validators. If the specified accounts are compromised, the entire XCT system is compromised as well. Meaning, that an attacker would be able to approve malicious transfers where, for instance, one token is burned but 10 tokens are issued. The risk could be reduced by using multi-signature accounts. Further research is required on the possible hardening of the validation process. Some ideas gathered during the work on this thesis are documented in the following sections.

CHAPTER 8

Conclusion

This chapter discusses the outcome of the thesis, and presents future and ongoing work.

8.1 Findings

Blockchain technologies are developing rapidly, causing severe infrastructure and market fragmentations. Furthermore, because of the isolated nature of blockchains, further technical means are needed to achieve interoperability between them. In such an environment developers are forced to choose a single blockchain for their applications, and users are exposed to the high risk and volatility of digital assets. Creating digital assets that are blockchain independent could minimize or even completely avoid these risks.

However, to create blockchain-independent digital assets, we first need to solve the cross-blockchain proof problem discussed in Chapter 4. In this thesis, we argue that it is possible to bypass the problem by introducing the concept of semi-decentralized validation parties.

In the course of this thesis, we specified the cross-chain token transfer protocol described in Chapter 5, and implemented the proof of concept consisting of components introduced in Chapter 6. In Chapter 4, we also specified a blockchain independent digital asset: the XCT token.

We showed in our lab environment that the concept of semi-decentralized validation parties enforced by the cross-chain token transfer protocol is feasible. We successfully performed multiple transfers of the XCT token from one blockchain to another and vice versa. All requirements specified in Chapter 4 were fulfilled. Although there is a short period of inconsistency when a token is burned on the source blockchain, but still not issued on the target blockchain, as soon as the corresponding `INIT` transaction is executed and at least one validator is available, the balances will be settled.

During the evaluation of the proof of concept we gathered insights about the required time and costs of cross-chain transfers.

The average durations of `INIT`, `BURN` and `APPROVE` transactions are 23, 28 and 36 seconds, respectively. In the implemented prototype, the `INIT` and `BURN` transactions are sent sequentially one after the other. However, they are independent and can also be executed in parallel, whereas the `APPROVE` transaction has to wait until both the `INIT` and `BURN` transactions are executed. Considering that a block confirmation time will take significantly more time, the average cross-chain transfer can be performed within acceptable time limits.

The gas consumptions of `INIT`, `BURN` and `APPROVE` transactions are 73.94, 58.62 and 67.41 kGas, respectively, and the total gas consumption for a single transfer is about 199.97 kGas. Arguably, taking into consideration two involved blockchains and multiple transactions, the cross-chain transfer costs are not significantly higher than the costs of a standard ETH transfer which is about 21 kGas [6].

8.2 Future Work

The developed prototype is a proof of concept demonstrating the feasibility of a cross-chain token transfer based on the concept of semi-decentralized validation parties. Some parts of the implementation are omitted for simplicity reasons, but they might be required in a real-world scenario. Also, many features and changes have been identified based on our findings in this thesis. In this section, we discuss these identified fields of possible future work.

8.2.1 Definition of Token Economics

Defining token economics was beyond the scope of the thesis, and the XCT token from the proof of concept is mintable by the contract owner. However, the economics of a token can have a decisive effect on the token acceptance by users. Modern tokens should avoid issues of early cryptocurrencies, including Bitcoin, that are distributed in a short period of time to small groups of people, and have supply limits. Similarly to other ERC20 tokens, initial distribution, monetary supply and mintage rate of the XCT token can be defined using a crowdsale smart contract. In the case of the XCT token, the crowdsale could take place on multiple blockchains due to its portability. Another advantage of the XCT token's portability, is that it is not tied to blockchains on which the initial crowdsale occurred, and can be freely moved to future blockchains as the blockchain technology improves.

8.2.2 Support for Non-Ethereum Blockchains

The XCT token is currently deployable to any Ethereum-based blockchain. The portability of the token is not limited to these blockchains. Any blockchain with a support of Turing-complete smart contracts and User-Issued Assets represents a potential candidate for cross-chain token transfers. Since validation is performed off-chain and validators are already able to manage multiple Web3 instances, validation of transfers on new blockchains does not require changes in the validation process that is specific to the new blockchains. Even blockchains whose capabilities are limited to simple scripts like Bitcoin can be supported by using on-blockchain token layers [1, 10, 29]. Another option is the usage of sidechains with support for smart contracts like Bitcoin's sidechain RootStock [43].

8.2.3 Improvements of Validation Process

We already discussed limitations of the semi-decentralized validation parties, and further research is needed in how to improve the validation process. To minimize the risk of compromised accounts, access control to `APPROVE` transactions can be enforced with multi-signature accounts, or that each transfer must be verified and approved by more than one validator.

Optimization of transfer costs is another field of a possible validation process improvement. A TID is calculated from a hash function and is 32 bytes long. Using a shorter TID might reduce storage usage and with that also the transfer costs, without compromising its uniqueness.

8.2.4 Web-Based Distributed Applications

The distributed applications used in the proof of concept are implemented as simple console applications. Although the validator console dApp is not intended to be used by end users, we can improve usability of the client and monitor dApps by implementing them as modern Web applications.

APPENDIX A

Measurements

A.1 Cross-Chain Token Transfer Metrics

Table A.1: Cross-Chain Token Transfer TIDs.

Trunsfer No.	TID
1	5f13795f6696c23e5efbed6ced72961fcb2f51c80eba32433aeec0aab92792ae
2	65b840e29827cc260ddfaf674fb8c703a69c6590f6050649b7136c5869b37715
3	da57841325ae864cb962b51cab57d1cbee2664d832060154780316f3af1e456f
4	07a2f1a28c6988bf43033a790b0af9b368d24fdade5219e71052a8afcb3e3849
5	d97a094783c6c910d1f127ea4e0f49f6f5ea346857ecd10dfdd7069a405b85de
6	b58534f471e3b4e575df8030dfd3377e8efe9e4651675e19c76d09ca73713f8c
7	767b421dd472635fe9a7dd8be1920e2622c486bf093df24af9eeea739264e4de
8	a7af421c01c59c0b25bc5b7de1debf26767ede04cdd32aeb04c32b574d8add89
9	55ce2de19805300ae827e4883875992d99c276fac456f22597e944f8fc996aa1
10	90d632c3d474e4a985ecbf9cb2325016cca7a9cb74a378c33de41f95775c70b4
11	19842a3c0a9e61af1eecb7eafb4e1b437bdcbb019b7667f8abd6b2539c9d11ea
12	1be3d28d2e09f81b96aa413f2e37349cf7e751c1c3c641544e8cac80dda24c03
13	f892aed10a9867609ee7144808ff0bebd1b631b27d34387417dcc61d8542a953
14	4658a1b6a965497a977694a50ba14acdc76ae2770aec4a61f4d2d6cf90b4439e
15	c9bc5dc3034ebae9432d1f1621fac36c44b128557ebc687c44e36fd5bfbec9b3
16	6629927745470a30021ad6f4dd7bad7c6e5ee25c5ae1456e7c60a0e8fc56fe06

Continuation of TableA.1	
Trunsfer No.	TID
17	a13cfe7afbb26f84cb72d782263b94530bc4623db9a9828a4af113faad0d9677
18	209ee6a4b052a7fba0a1249538e1d6ce9bebd0fc2cc269f611ca1b183e04cbf3
19	21a5ecae470fe8dee93deb6ef875350f1ca66141001177042b85e19e04fef6d4
20	a6c67479ea961ab756300a9eac9836ac6aac3f8bef7739a1178ac648286774cf
21	4408b5ff9fb37859e2ead9d0f509f9d40b089b99712949e4c7c0fe9e18315918
22	b53a505c53017bf6f1449cf18204824eda6f5ba70ac5769cc5a87b33b5028a3d
23	ea76f2e002eb727d3b7e084194932afcb0bc49eac927737fabb1ef0a6a36c0a1
24	1494bb0d85c2f19ceb5c27edbb11b7a47877d6516227570d5a7346bceadf2fb8
25	1896d71c8257be831a11181024c4942786b7c80f743ebae1074b2bcf0dc64498
26	176c5047cbf8a9055a8ab49a61e618aef54798c4b0772edefe531d5b6fdd16c5
27	70cb510d770d99af5f6b3d30fff964b7578c8136ab9b812afcecdfc4f2a63792
28	075aef760234d321fa0743ca1a9eeaa6db7059c87c62421baa9c556ce86faec0
29	4a49a12cfd59fda564b0833e30a40f376cc4179f694c65ecc8ad52d5eb18c05d
30	ac28efe09d238a7596ba359d3a65b7b599425a1fc006f668708cfff0f32c3f37
31	99e09e1a24f056dd4031818040105d7d5f277117dbea615057a056733548fa65
32	97a0b4ecc02de95a3b49debceec9418957e9ae6c2573de2d06215cea0a70b960
33	bfe032afaffa49947fbef2a948706096901263f74b9f99181d6f7eed07ce2c9b
34	b179b40db4dd5c8d23e5b51b5c459657976f91169c95ad05e890ac9cb98c09b6
35	5e424f9d6034a7d885f088e095ac7649b832bbfdba61c4c7d56d1296d0c89bef
36	6fd4da05c6cde493d4b9464cc7ceddc21865e999fbb8e65ff1f593fea6bdd867
37	6f97ab7083e6dc34f6345a7fbf3c60ae11ca170343b413942ceab8db43f3e2ad
38	bd890bd0683fee8bd4326652e759b8fa2e198a2369102c9b0f899a87fe0bb0c1
39	931f1bc89b9deaa80f7be8e283b021ab0a2bbee6cf4778b8a3647fb9e0242fda
40	ef020f9c4d8e836f489ca7e5fce173b360bce58352b628755de5a95eee08291d
41	33891781acb7a147660e5ae2b00524bd4a3f53715624fabc001a582d11308917
42	28e152cced37446220ee5595907409f9030afba0ddda8bdfd1e18c4d1cd4b4c7
43	cb099516274d6be0ef46a35b74e27959501908f946c0d436c7c0c8b8af416bb7
44	10097ea8ffefb18d2ca6e88d57a375a7639e00f136a126cd069f9efd1cccf671
45	3842d9c2dc036cb266c97a59401c087b9b2342b973fceb1302d16a58f2b199fd

Continuation of TableA.1	
Trunsfer No.	TID
46	d7ff8e8da408e560e6d07c52c2729185e06c918c5dbfbe461c7adc6a34e116b4
47	032e6f3662483fa7683347a5445262b256b784d1b27819c4517dee4e7f3d3940
48	259070409da87ae98b62e75a20ba0deb137ae5bab1629777c8876ac231a70f12
49	ff8e99db2650d8da7e7f479f2182fffd8d283461984cf1a6a12854c09f8f959d
50	6a3eefee9d83d1ea18be78558ae8a3e9ceac5ccc815d660eb0892174e28f9d51

Table A.2: Cross-Chain Token Transfer INIT Transaction Metrics.

Trunsfer No.	Source Chain	Target Chain	INIT Duration (s)	INIT Gas
26	Local	Ropsten	21	73937
27	Local	Ropsten	12	73937
28	Local	Ropsten	37	73937
29	Local	Ropsten	27	73937
30	Local	Ropsten	19	73937
31	Local	Ropsten	24	73937
32	Local	Ropsten	10	73937
33	Local	Ropsten	35	73937
34	Local	Ropsten	29	73937
35	Local	Ropsten	52	73937
36	Local	Ropsten	20	73937
37	Local	Ropsten	23	73937
38	Local	Ropsten	29	73937
39	Local	Ropsten	20	73925
40	Local	Ropsten	27	73937
41	Local	Ropsten	23	73937
42	Local	Ropsten	33	73937
43	Local	Ropsten	18	73937
44	Local	Ropsten	36	73937
45	Local	Ropsten	14	73937
46	Local	Ropsten	12	73937
47	Local	Ropsten	8	73937

Continuation of TableA.2				
Trunsfer No.	Source Chain	Target Chain	INIT Duration (s)	INIT Gas
48	Local	Ropsten	12	73937
49	Local	Ropsten	21	73937
50	Local	Ropsten	11	73937

Table A.3: Cross-Chain Token Transfer BURN Transaction Metrics.

Trunsfer No.	Source Chain	Target Chain	BURN Duration (s)	BURN Gas
1	Ropsten	Local	32	58616
2	Ropsten	Local	17	58616
3	Ropsten	Local	34	58616
4	Ropsten	Local	47	58616
5	Ropsten	Local	62	58616
6	Ropsten	Local	32	58616
7	Ropsten	Local	17	58616
8	Ropsten	Local	2	58616
9	Ropsten	Local	17	58616
10	Ropsten	Local	17	58616
11	Ropsten	Local	17	58616
12	Ropsten	Local	32	58616
13	Ropsten	Local	17	58616
14	Ropsten	Local	2	58616
15	Ropsten	Local	17	58616
16	Ropsten	Local	32	58616
17	Ropsten	Local	32	58616
18	Ropsten	Local	32	58616
19	Ropsten	Local	32	58616
20	Ropsten	Local	32	58616
21	Ropsten	Local	17	58616
22	Ropsten	Local	62	58616
23	Ropsten	Local	32	58616

Continuation of TableA.3				
Trunsfer No.	Source Chain	Target Chain	BURN Duration (s)	BURN Gas
24	Ropsten	Local	32	58616
25	Ropsten	Local	17	58616

Table A.4: Cross-Chain Token Transfer APPROVE Transaction Metrics.

Trunsfer No.	Source Chain	Target Chain	APPROVE Duration (s)	APPROVE Gas
26	Local	Ropsten	29	67416
27	Local	Ropsten	90	67416
28	Local	Ropsten	89	67416
29	Local	Ropsten	30	67416
30	Local	Ropsten	30	67416
31	Local	Ropsten	15	67416
32	Local	Ropsten	15	67416
33	Local	Ropsten	75	67416
34	Local	Ropsten	15	67416
35	Local	Ropsten	30	67416
36	Local	Ropsten	60	67416
37	Local	Ropsten	30	67416
38	Local	Ropsten	15	67416
39	Local	Ropsten	30	67416
40	Local	Ropsten	15	67416
41	Local	Ropsten	60	67404
42	Local	Ropsten	15	67416
43	Local	Ropsten	15	67416
44	Local	Ropsten	45	67404
45	Local	Ropsten	15	67416
46	Local	Ropsten	15	67416
47	Local	Ropsten	75	67416
48	Local	Ropsten	15	67416
49	Local	Ropsten	15	67416
50	Local	Ropsten	45	67416

List of Figures

List of Tables

Listings

Bibliography

[1] Counterparty. `https://counterparty.io/docs`. Accessed 2024-04-15.

[2] Crowdsales. `https://docs.openzeppelin.org/v2.3.0/crowdsales`. Accessed 2024-04-15.

[3] Decred-compatible cross-chain atomic swapping. `https://github.com/decred/atomicswap`. Accessed 2024-04-15.

[4] Deploying and interacting with smart contracts. `https://docs.openzeppelin.com/learn/deploying-and-interacting`. Accessed 2024-04-15.

[5] Designing the architecture for your ethereum application. `https://blog.openzeppelin.com/designing-the-architecture-for-your-ethereum-application-9cec086f8317`. Accessed 2024-04-15.

[6] Gas and fees. `https://ethereum.org/en/developers/docs/gas/`. Accessed 2024-04-15.

[7] Introduction to smart contracts. `https://ethereum.org/en/developers/docs/smart-contracts`. Accessed 2024-04-15.

[8] The merge. `https://ethereum.org/en/roadmap/merge`. Accessed 2024-04-15.

[9] A next-generation smart contract and decentralized application platform. `https://ethereum.org/en/whitepaper`. Accessed 2024-04-15.

[10] Openassets. `https://github.com/OpenAssets`. Accessed 2024-04-15.

[11] Simple replay attack protection. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md`. Accessed 2024-04-15.

[12] Smart contract security. `https://ethereum.org/en/developers/docs/smart-contracts/security/`. Accessed 2024-04-15.

[13] Tokens. `https://docs.openzeppelin.org/v2.3.0/tokens`. Accessed 2024-04-15.

[14] Understanding the dao attack. https://www.coindesk.com/learn/understanding-the-dao-attack/. Accessed 2024-04-15.

[15] Web3j documentation. https://docs.web3j.io. Accessed 2024-04-15.

[16] Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf, 2014. Accessed 2024-04-15.

[17] A. Saxena, J. Misra, and A. Dhar. Increasing Anonymity in Bitcoin. In *Financial Cryptography and Data Security*, pages 122–139. Springer Berlin Heidelberg, 2014.

[18] B. Schneier. OneWay Hash Functions. Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C. pages 429–459, 2015.

[19] C. Prybila, S. Schulte, C. Hochreiner, and I. Weber. Runtime verification for business processes utilizing the bitcoin blockchain. In *Future Generation Computer Systems*, 2018.

[20] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO*, pages 199–203, 1982.

[21] J. R. Douceur. The sybil attack. In *Peer-to-Peer Systems, First International Workshop, IPTPS*, pages 251–260, 2002.

[22] F. Tschorsch, and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys and Tutorials*, 18(3):2084–2123, 2016.

[23] F. Vogelsteller, V. Buterin. Eip 20: Erc-20 token standard. https://eips.ethereum.org/EIPS/eip-20, 2015. Accessed 2024-04-15.

[24] F. Wessling, C. Ehmke, M. Hesenius, and V. Gruhn. How much blockchain do you need? towards a concept for building hybrid dapp architectures. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 44–47, 2018.

[25] C. Fromknecht. Connecting blockchains: Instant cross-chain transactions on lightning. https://blog.lightning.engineering/announcement/2017/11/16/ln-swap.html, 2017. Accessed 2024-04-15.

[26] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak sponge function family main document. https://keccak.team/obsolete/Keccak-main-1.0.pdf, 2009. Accessed 2024-04-15.

[27] M. Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC*, pages 245–254, 2018.

[28] J. Poon, and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2016. Accessed 2024-04-15.

[29] J. Willett, M. Hidskes, D. Johnston, R. Gross, and M. Schneider. Omni protocol specification. `https://github.com/OmniLayer/spec`. Accessed 2024-04-15.

[30] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM*, 4(3):382–401, 1982.

[31] M. Borkowski, C. Ritzer, and S. Schulte. Deterministic witnesses for claim-first transactions. `http://dsg.tuwien.ac.at/staff/mborkowski/pub/tast/tast-white-paper-3.pdf`, 2018. Accessed 2024-04-15.

[32] M. Borkowski, D. McDonald, C. Ritzer, and S. Schulte. Caught in chains: Claim-first transactions for cross-blockchain asset transfers. `http://dsg.tuwien.ac.at/staff/mborkowski/pub/tast/tast-white-paper-2.pdf`, 2018. Accessed 2024-04-15.

[33] M. Borkowski, D. McDonald, C. Ritzer, and S. Schulte. Towards atomic cross-chain token transfers: State of the art and open questions within TAST. `http://dsg.tuwien.ac.at/staff/mborkowski/pub/tast/tast-white-paper-1.pdf`, 2018. Accessed 2024-04-15.

[34] M. Conoscenti, A. Vetro, and J. C. De Martin. Blockchain for the internet of things: A systematic literature review. In *IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–6, 2016.

[35] M. Di Angelo, A. Soare, and G. Salzer. Smart contracts in view of the civil code. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 392–399. ACM, 2019.

[36] M. Marchesi, L. Marchesi, and R. Tonelli. An agile software engineering method to design blockchain applications. In *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia*, pages 3:1–3:8. ACM, 2018.

[37] M. Wöhrer, and U. Zdun. Design patterns for smart contracts in the ethereum ecosystem. 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)*, pages 1513–1520, 2018.

[38] M. Wöhrer, and U. Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. pages 2–8, 2018.

[39] N. Szabo. Smart contracts. `https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html`. Accessed 2024-04-15.

[40] N. van Saberhagen. Cryptonote v 2.0. `http://diyhpl.us/~bryan/papers2/bitcoin/cryptonote.pdf`, 2013. Accessed 2024-04-15.

[41] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, 2008. Accessed 2024-04-15.

[42] D. Rakic. Blockchain technology in healthcare. In *Proceedings of the 4th International Conference on Information and Communication Technologies for Ageing Well and e-Health - Volume 1: ICT4AWE*, pages 13–20, 2018.

[43] S. D. Lerner. Rsk: Bitcoin powered smart contracts. `https://rootstock.io/rsk-white-paper-updated.pdf`, 2019. Accessed 2024-04-15.