h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik –

Atomic Cross-Chain Swaps with Ethereum Blockchains

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von

Christopher Glißner

Matrikelnummer: 732978

Referent	:	Prof. Dr. Ronald Moore
Korreferent	:	Prof. Dr. Stefan Rapp

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 28. April 2020

Christopher Glißner

ABSTRACT

Today's blockchain ecosystem has spread into multiple chains using different consensus algorithms and architectures. After a decade of blockchain applications and development there is a need for interoperability between upcoming and current blockchain innovations. Interoperability presents a complex task due to the architectural differences of blockchain technologies, to swap assets and tokens between chains in a steadily growing environment. This ecosystem consists of many chains without any interoperability. For example if Alice wants to trade her bitcoin with Bob's ether they have to use a centralized exchange, which exposes them to the possibility of scam, fraud and malicious hacks. This thesis provides a prototype with a three-way swap between three ethereum blockchains, to execute such a swap between bitcoin and ethereum further research is needed since bitcoin's script language is not suited for swaps at this time. Atomic Cross-Chain Swaps propose a small step towards interoperability of blockchains, by removing the intermediate parties such as exchanges. Blockchain interoperability enables more security for the end-user and closes attack vectors for fraudulent actors.

ZUSAMMENFASSUNG

Das heutige Blockchain-Ökosystem hat sich durch die Implementation verschiedener Konsensusalgorithmen und Architekturen in viele verschiedene Blockchains aufgeteilt. Nach einem Jahrzehnt der Blockchain-Anwendungen und -Entwicklungen besteht ein Bedarf für Interoperabilität zwischen kommenden und aktuellen Blockchain-Innovationen. Die Interoperabilität stellt aufgrund der architektonischen Unterschiede der Blockchain-Technologien eine komplexe Aufgabe dar, um somit Assets und Token in einer stetig wachsenden Umgebung zwischen Blockchains auszutauschen. Dieses Ökosystem besteht aus vielen Blockchains ohne Interoperabilität. Wenn Alice beispielsweise ihre Bitcoin mit Bobs Ether tauschen möchte, müssen sie zentralisierte Börsen für Kryptowährungen verwenden und sind da durch der Möglichkeit von Betrug und böswilligen Hacks ausgesetzt. Diese Arbeit liefert einen Prototyp mit einem Drei-Wege-Austausch zwischen drei Ethereum-Blockchains, um einen solchen Austausch zwischen Bitcoin und Ethereum durchzuführen sind weitere Untersuchungen erforderlich, da die Skriptsprache von Bitcoin derzeit nicht für den Austausch geeignet ist. Atomic Cross-Chain Swaps schlagen einen kleinen Schritt in Richtung Interoperabilität von Blockchains vor, indem diese Zwischenparteien wie Börsen entfernen. Die Blockchain-Interoperabilität bietet mehr Sicherheit für den Endbenutzer und schließt Angriffsmethoden für betrügerische Akteure.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CT Confidential Transactions

DeXTT Determistic Cross-Blockchain Token Transfers

DApp Decentralized Application

ECDSA Elliptic Curve Digital Signature Algorithm

EVM Ethereum Virtual Machine

EVM Code Ethereum Virtual Machine Code

ISO International Organization For Standardization

NIPoPoWs Non-Interactive-Proofs-of-Proof-of-Work

UTC Universal Time Coordinated

PBFT Practical Byzantine Fault Tolerance

PoW Proof-of-Work

PoS Proof-of-Stake

SHA-256 Secure Hash Algorithm 256

SHA Secure Hash Algorithm

SPV Simple Payment Verification

Part I

THESIS

1 INTRODUCTION

The following subsections of this chapter discuss the motivation, research objectives and related work of this theses. The goal is to implement a prototype which executes a three-way swap that helps Alice, Bob and Carol to execute their trade and protect them of fraud if one of the parties behaves irrationally. At the current state of the cryptocurrency environment they still need to use a centralized exchange to swap assets between different blockchains. Decentralization of the trading process has the potential to protect users from scams and to remove a single point of failure in the case an exchange gets hacked or steals the funds of their users.

1.1 MOTIVATION

Atomic cross-chain swaps have the potential to remove the intermediate parties such as exchanges in the cryptocurrency environment. By removing centralized exchanges and handling the exchange process with smart contracts and a swap protocol attack vectors which can be used by malicious actors are reduced. Scams and hacks of the exchange are not possible anymore, resulting in increased security for the user who wants to trade crypto assets. Interoperability has the potential to let companies exchange information or assets with hard coded rules they agreed on before to be execute by smart contracts. Since blockchain is a relative new technology that is evolving and changing rapidly achieving interoperability brings difficulties, due to different architectures, consensus algorithms and missing support for turing-complete smart contracts. This theses contains a summary of related work on the blockchain interoperability research field and collected knowledge about the different technologies addressing swaps. An implementation of a three-way swap prototype, performance tests and an outlook whats possible in the future by adding more features to the prototype are discussed. An overview over the current state on whole research field addressing blockchain interoperability and a proposal of a reference architecture for universal swaps are also part of this theses. Swaps and blockchain interoperability are an interesting topic since it could protect Alice and Bob ending up worse, if Carol would behave irrationally or tries to scam them. The smart contracts ensures that funds are returned to their owners if the trade is canceled. Several things have been learned from implementing the three-way swap prototype. Through modifications and by adding more features it could execute any kind of swap from $2 \dots n$ actors with $2 \dots n$ blockchains. In the case of the ethereum blockchain, its available clients and libraries more work is needed to improve visiblity for the user, since internal transactions by a smart contract do not show up on the balance of an account. Also blockchain inter-

operability can only be achieved by software that supports turing-complete smart contracts.

1.2 RESEARCH OBJECTIVES

This chapter 1 discusses the motivation and several historic information which led to the research objective covered by this thesis. The next chapter 2 introduces the model and general principles serving as foundation to address the research objective. Followed by chapter 3 which presents a requirement analysis for the three-way swap prototype. In chapter 4 the implementation of atomic cross-chain swaps on Ethereum blockchains is presented to help Alice, Bob and Carol with the trades they want to do. The last chapter 5 discusses a conclusion of this work, to reflect the whole thesis and addresses future research directions. Finally the last chapter 6 presents important terminology, which is crucial to understand the topics mentioned above. This theses aims to collect and summarize the current state of blockchain interoperability. By the implementation of a prototype that executes a three-way swap in order to help Alice, Bob and Carol to execute their trade in a decentralized and atomic way, this work aims to deliver a proof of concept of atomic cross-chain swaps. Possible future research directions are discussed as part of the conclusion at the end of this work. For simplicity three ethereum chains are used instead of bitcoin, alt-coin and a car-title chain. Implementation shows that even with three chains that use similar technology and consensus algorithm, problems arise that suggest further research to fully decentralize intermediate parties like cryptocurrency exchanges.

1.3 RELATED WORK

Decentralization and scaling of blockchains is a widely addressed topic by many researchers. There is a vast amount of cryptocurrencies and technologies out there [18]. Bitcoin [47] was the first successful blockchain application which decentralizes money, combining the well-known concepts of Proof-of-Work (PoW) algorithms and the distributed ledger concept. Based on previous work by Wood [63], Buterin described the next-generation smart contract and decentralized application platform known as Ethereum [13] [14]. After the success of ethereum his research continues towards chain interoperability [11] and addresses the scalability limits of today's blockchains with his work on sharding blockchains [12]. While most present blockchains still remain unconnected, pegged sidechains are formally defined by [4] and is the foundation for many proposals towards blockchain interoperabilty. Originally known as BitMonero, the Monero cryptocurrency was created in April 2014 [3] as a proof-of-concept currency of the CryptoNote [58] technology. Cryptocurrencies dont just exist with different consensus algortithms and architecures, they also aim to serve a specific purpose. Monero addresses privacy and utilitizes borromean ring signatures [44] to implement

ring Confidential Transactions (CT) [49] in order to improve untraceability. Cardano uses a research-first driven approach to develop a smart contract platform which seeks to deliver more advanced features than any protocol previously developed. For example the initiators developed their own Ouroboros Proof-of-Stake (PoS) protocol [38] and proposed a work about PoS sidechains [27]. Cardano also did research on Non-Interactive-Proofs-of-Proof-of-Work (NIPoPoWs) to increase PoW performance and to enable cross PoW based blockchain transfers of assets [37] and utilities the results to propose PoW sidechains [39]. There is more work about sidechains and interoperability by Johnson [34], Dagher who puts focus on interoperability security [21] and Robinson's work about ethereum private sidechains [52]. Herlihy proposed the concept of atomic cross-chain swaps to exchange tokens and assets between blockchains safely [29] to enable communication between two blockchains without intermediaries. The concept of two-party atomic cross-chain swaps is believed that it emerged from an online forum discussion, see footnotes for references to bitcoinwiki and bitcointalk threads [67] [55]. There is open source code available on github [9] [23] implementing a two-party cross-chain swaps protocol for selected currencies, as well as another proposal for applications using swaps [64]. To address the scalability limits of existing blockchains several off-chain payment networks do exist, for further details refer to [48] and to [51] [24] [28]. These concepts process multiple transactions off-chain and eventually resolve the final balance through a single on-chain transaction. The Revive network [36] is another actor which uses off-chain transactions in a way to ensure, that participating parties do not end off worse in the process of re-balancing. These algorithms also utilitize the concept of hashed timelock contracts, but they address a different set of problems. Multi-party swaps in example become important, when matching kidney donors and recipients. Think of a case, where a transplant recipient has an incompatible donor and wants to swap donor with another recipient where all parties want to assure, that each recipient obtains a compatible organ. A number of algorithms [2] [25] [31] have been proposed for matching donors and recipients. Shapley and Scarf [53] write about the circumstances under which a certain kind of swap markets could have strong equilibrium. Kaplan [35] describes a polynominal-time algorithm, which constructs a swap digraph for a given set of swaps, if one exists. This paper focuses on "the clearing problem" though, which is roughly analogous to constructing a swap digraph, but not on how execute those atomic cross-chain swaps on blockchains. A precursor of atomic cross-chain swaps is the fair exchange problem [26] [45]. It tries to find a solution for an asset exchange between parties, which do not trust each other. Since its proven that a fair exchange between two parties can not be guaranteed without a trusted third party [50], atomic cross-chain swaps could be a successor to the fair exchange problem through new technologies like blockchain and smart contracts. Borkowski proposes Determistic Cross-Blockchain Token Transfers (DeXTT) another uptake on blockchain interoperability by formally describing blockchains and token transfers between those, through

utilitizing concepts such as claim-first transactions [7] and determistic witnesses [8]. Komodo [40] by SuperNETorg is another available atomic swap decentralized exchange of native coins, their BarterDEX system combines three key component like order matching, trade clearing and liquidity provision in a single system. It allows users to make a coin conversion request, find a suitable match and complete the trade using an atomic cross-chain protocol. Despite the fact that blockchain interoperability is addressed by many researches in different ways, these cross-chain technologies fall into three main categories according to Li and Zhang [43]. The main categories of the cross-chain technologies are notary or multiple signature schemes, side chain/relay chain and hash timelock, the principles of these main categories are shown in table 1.1.

Cross-chain technology	Algorithmic principle	Shortcoming
Notary	Transfer of assets to notary accounts	Weak centralization
Side chain	Read the data in the side chain	Verification difficulties
Relay chain	Establishing relay chain	Data redundancy

Table 1.1: Cross-chain technology comparison

The notary or multi-signature scheme is a solution with weak centralization. It transfers assets by locking them to multiple accounts on the mainchain and requires signatures from multiple notaries to release the locked funds. Ripple by Interledger protocol [30] transfers assets across different blockchains through trusted third parties. To transfer assets cross-chain, the notary or multi-signature scheme is the simplest way to achieve swaps, but still depends on the honesty of the notaries to ensure none of the participating parties end up worse. The side chain technology transfers digital assets by reading the data from the main chain and verifies the authenticity of the transaction payments. For example the BTC Relay [19] project implements those cross-chain payments by verifying the bitcoin Simple Payment Verification (SPV) payment path to trigger ethereum smart contracts [14]. The relay technology links existing blockchains by building a relay chain to implement cross-chain swaps of assets. Cosmos [42] which is based on previous research on the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm [15] and tendermint [41] [10] and Polkadot [62] propose a cross-chain platform compatible with any blockchain, to build such a system they combine the notary and relay technology.

2

MODEL AND GENERAL PRINCIPLES

This chapter discusses the model and general principles, which serve the foundation to understand the implementation reports of the following chapter. Its important to understand every subsection of this chapter, such as the ethereum blockchain (chapter 2.1.1), PoW consensus algorhithm (chapter 2.1.1), the Ethereum Virtual Machine (EVM) (chapter 2.1.2), smart contracts (chapter 2.1.3) and all topics needed for atmoic cross-chain swaps (chapter 2.2 and 2.3).

2.1 ETHEREUM BLOCKCHAIN

The Ethereum blockchain was introduced in Vitalik Buterin's paper in 2013, which addressed several limitations of the Bitcoin's scripting language [13] based on Buterin's white paper Wood released the yellow paper one year later [63]. The main contributions are full turing-completeness and saving all states of computations in between the states [22]. Through its own programming language Solidity, it provides an abstract layer enabling anyone to create their own rules for ownership, formats of transactions, and state transition functions [60]. Development was funded by an online crowdsale that took place between July and August 2014 [56]. The system then went live on 30 July 2015 [57] and is called the ethereum public blockchain. Which means its accessible to everyone who wants to send Ether token or execute smart contracts. In general we have to distinguish between private and public blockchains. The public one is described above. Since everybody can go to github and fork e.g the geth client its possible to start an own private blockchain. In an enterprise software context, where corporate stakeholders are given certain rights and privileges to read and write to the company chain, the deployment is known as a permissioned blockchain. Nevertheless for both the private (permissioned) and the public blockchain its possible to do the following [22]:

- Send and receive Ether
- Write smart contracts
- Create provably fair applications
- Launch your own token based on Ether

2.1.1 *Proof-of-Work Consensus Algorithm*

The PoW system of bitcoin and ethereum is similar to Adam Back's Hashcash [5], but instead of a newspaper or Usenet post, the PoW involves scanning for a value, which is hashed with the Secure Hash Algorithm 256 (SHA-256) to begin with a number of zero bits. The average work required is exponential. That means the more leading zeros, the more work. In the end the work can be later verified by executing a single hash. The PoW is implemented by incrementing a nonce in the block until a value is found that gives the block's hash the required leading zero bits. The majority decision is represented by the longest chain, which has the greatest PoW effort invested in it. Because later blocks are chained after the premature block, the work to change the block would include redoing all the blocks after it. To modify a past block, an attacker would need to redo all the past work until the specific block he wants to change [47].

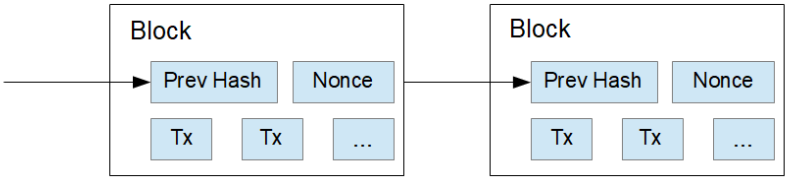

Figure 2.1: Blocks

Beginning with the genesis block, each single block is chained to the previous block by including the the previous hashes (figure 2.1). The SHA-256 takes a 256 bit length of input messages and hash them to fixed-length outputs [59]. So the SHA-256 function starts by padding the message according to the so-called Merkle-Damgård strengthening technique. Further the message is processed block by block with the the underlying compression function, which initializes an appropriate number of chaining variables to a fixed value to hash the genesis block and also the current hash value for the following blocks [20]. In the year 2003 the Secure Hash Algorithm (SHA) was published by the International Organization For Standardization (ISO) making SHA a standard for most countries in the world [1]. Five years later Satoshi Nakamoto proposed the first concept of a PoW based blockchain to allow for public agreement on the order of transactions. The public ethereum blockchain uses PoW to this day. There are several other consensus algorithms today suitable for a blockchain and some are considered to be an improvement to PoW, which are discussed later in this work.

2.1.2 *Ethereum Virtual Machine*

This chapter covers the basics of the EVM and discuss the transaction execution to reach a general understanding what the EVM is and how it works. The EVM is a simple stack-based architecture. The word size of the machine is 256-bit and the stack has a maximum size of 1024 elements, meaning that the EVM can store a total of 1024 stack items with each 256-bit. The machine does not follow the standard von Neumann architecture. Rather than storing program code in generally-accessible memory or storage, it is stored separately in a virtual ROM. The EVM is different from the von Neumann architecure, because it can have exceptional execution thus as the out-of-gas exception [63]. The ethereum network can be seen as one single computational device, where the point of turning the machine on, is the creation of the genesis block. All blocks of a blockchain hold information about state changes of the EVM. The ethereum public blockchain for example holds all state transitions of the EVM since it was initially turned on. Since every state transition of the EVM is modelled as a transaction, a blockchain holds the whole history of states of the specific machine [22]. Before it is put into a block each transaction is verified and validated before the next canonical block can be placed on the last one. Through the transaction validation, nodes on the network do not need to individually evaluate the trustworthiness of every single block in the network to compute the present balances of the accounts on the network. The client simply has to verify the hash of the parent block and see if the new block contains the correct hash its parent's transactions and state [22]. The current valid block in an ethereum network is known as world state (state), which is a mapping between addresses (160-bit identifiers) and account states. Thus being the current executional state of the virtual state machine, known as the EVM. It is considered as a quasi-touringcomplete virtual machine. Because the amount of total execution is intrinsically bound to the parameter gas, the quasi qualification is added [63]. The execution model is defined by the ethereum state transition function and specifies how system state is altered through bytecode executions.

The ethereum state transition function, APPLY(S, TX) -> S' is defined by Buterin [13] as follows:

1. Check for transaction and signature validity and check if the nonce of the sender account matches the receivers account's nonce. If not, return an error.

2. Calculate the transaction fee and determine the sending address from the signature. Increment the sender's nonce and substract the fee from the sender's account. Return an error, if the balance of the sender's account is lower than the fee.

3. Initialize GAS including a certain quantity of gas per byte to pay for the bytes in the transaction.

4. If the receiver's account doesnt exist, create it and then send the transaction value from sender's account to the receiver. If the receiver's account is a contract, run code execution until completed or gas is depleted.

5. In case the value transfer or code execution failed because theres not enough gas, revert all state changes except the payment of the fees and add the fees to the miner's account.

6. If code execution has completed, refund the fees for all remaining gas to the sender and also send the fees paid for gas consumed to the miner.

The Figure below shows all calculations applying the function APPLY(S, TX) -> S' to an example state (figure 2.2). Due to the distributed nature of the EVM and the fact its built of many nodes around the world to ensure distributed security, it must be purpose-built to solve the diffmatching problem that can occur, when there are many near-simultaneous changes to the same database from many actors in the system all around the world [17]. Solving this problem in a verifiable and trustworthy way is basically what the EVM does. It's resilience and security does increase with the amount of machines in the network and is implemented and backed by the gas fee system. The opportunity to earn gas as a miner is one reason for many nodes to participate in the network and at the same time raises security [22].

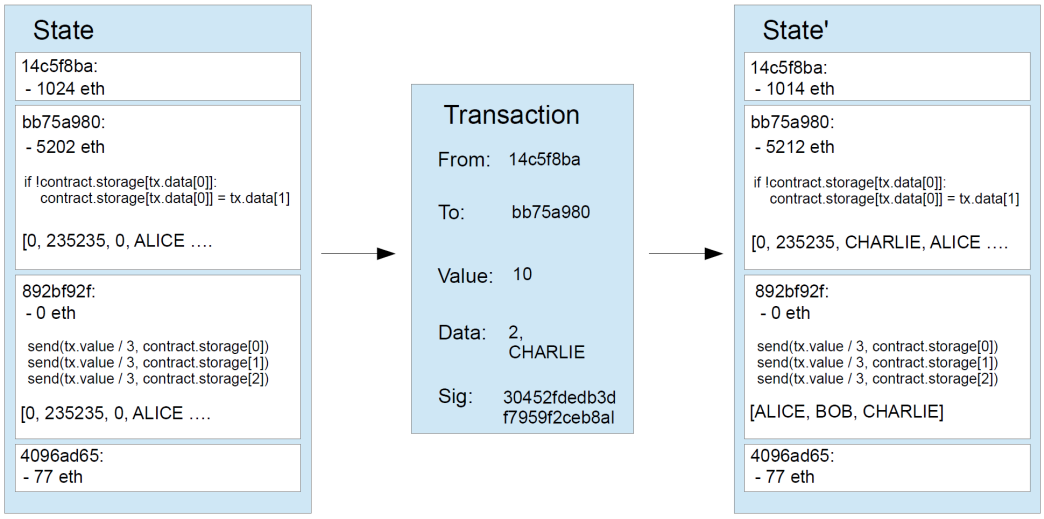

Figure 2.2: Ethereum State Transition Function

2.1.3 *Smart Contracts*

Around the 1990s it became clear that algorithmic enforcement of agreements could become common in the way humans cooperate. Early work towards smart contracts was done by Szabo [54] and Mark Miller [46]. Even though no specific system was proposed by them, they proposed that the future of law would be driven by such systems. Ethereum can be seen as such a system, it delivers a general implementation to execute financial contracts by a machine. The term smart contract was first introduced by Buterin [13], in scientific context they are simply called contracts by Wood [63]. In this context, contract refers to a specific kind of contract: a financial contract, also known as derivative or option. These contracts are agreements between two parties to buy or sell a specific asset at some point in the future or when certain conditions are met, like a specified price. In ethereum context a contract does basically the same thing as a financial contract. The contract is written as code and is an agreement between accounts, to send a payment to a specified address, when certain conditions are met. The reason these contracts are called smart contracts, is because they are executed by a machine, also known as the EVM. When certain conditions are met, the assets (ether or token) are moved automatically by the machine. Assuming the system is still running, these smart contracts can also take effect hundreds of years after being deployed. As long as the EVM is running deployed contract maintain their validity. Its basically impossible for a party to back out of these contracts, because the smart contracts are empowered to move or hold assets and move them to specific accounts when the implemented conditions are met [22]. While smart contract are often written in the high level programming language solidity, the code the EVM understands and executes is written in a low-level, stack based bytecode language, referred to as Ethereum Virtual Machine Code (EVM Code). In general the bytecode consists of a series of bytes, where each byte represents an operation to be executed by the EVM and code execution is an infinite loop executing code and storing data to the following operations until an error or STOP or RETURN is reached [13]:

- Stack (last-in-first-out container to which 32-byte values are pushed or popped)
- Memory (infinitely expandable byte array)
- Storage (of a contract, changes here don't reset after computation ends they are persistent)

2.2 DIGRAPHS

A directed graph (digraph) $\mathcal{D}$ (figure 2.3) in the example consists of a finite nonempty set of objects called vertices. So the vertex set of the example 2.3 is $V(\mathcal{D}) = \{u, v, w, x, y, z\}$. It also has a finite set of A($\mathcal{D}$) of ordered pairs of distinct vertices called arcs (or directed edges according to [16]), in this example 2.3 $A(\mathcal{D}) = \{(u, v), (u, w), (w, u), (z, u), (x, z), (y, z)\}$. So the digraph $\mathcal{D}$ consists of $V(\mathcal{D})$ the vertex set and $A(\mathcal{D})$ the arc set of $\mathcal{D}$, which can also be expressed as $\mathcal{D} = (V, A)$. The order (size) is sometimes denoted as $|\mathcal{D}|$, so the size of the digraph $\mathcal{D}$ in the example 2.3 is $|\mathcal{D}| = 6$. For an arc (u, v) the first vertex u is its tail and the second vertex v is its head, we can also say that the arc (u, v) leaves u and enters v. Another expression is, assuming (u, v) is an arc, one can say that u dominates v (or v is dominated by u) and can be denoted by $u \to v$. A digraph $\mathcal{D}$ is strongly connected (or just strong) if each vertex of $\mathcal{D}$ is reachable from every other of $\mathcal{D}$'s vertices. If its not possible to loop through a digraph $\mathcal{D}$, then it has no cycle and the digraph $\mathcal{D}$ is acyclic. The set S of vertices (arcs) is called a feedback vertex set or an feedback arc set, if $\mathcal{D}$ happens to be acyclic. Let $\mathcal{H}$ be another digraph, if $V(\mathcal{H}) \subseteq V(D)$, $A(\mathcal{H}) \subseteq A(D)$ and every arc in $A(\mathcal{H})$ has both end-vertices in $V(\mathcal{H})$, then H is a subdigraph of digraph $\mathcal{D}$. The distance from x to z is the minimum length of a (x, y)-walk and is denoted as $dist(x, y)$ of the given digraph $\mathcal{D}$ in figure 2.3. If a digraph $\mathcal{D}$ is strongly connected and only if $\mathcal{D}$ is strong, then the diameter of $\mathcal{D}$ is $diam(\mathcal{D}) = dist(V, V)$ [6]. The example in this chapter and its figure 2.3 is important to understand and to know the basic terminology, since the processes of atomic cross-chain swaps itself is based on digraphs.

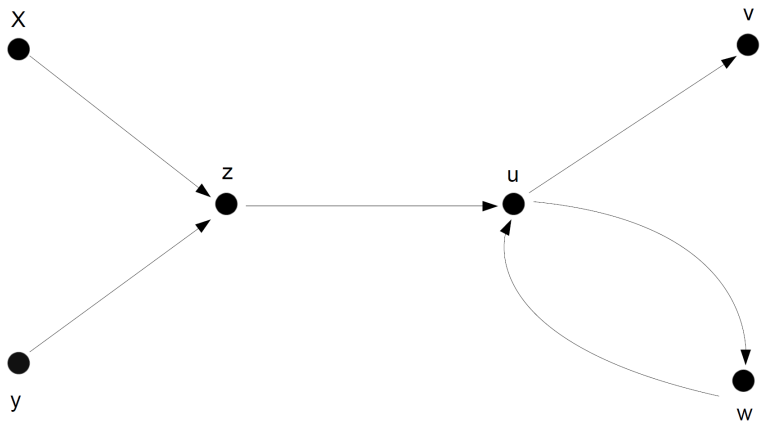

Figure 2.3: A Digraph D

2.3 ATOMIC CROSS-CHAIN SWAPS

An atomic cross-chain swap is like the name indicates an atomic transaction to swap assets, which is also across multiple blockchains. Altough ethereum can be seen as one system, its still a distributed ledger where state transitions are processed by all miners in the network, so the atomic cross-chain swap is a distributed coordination task (special case of distributed atomic transaction) [61]. Since the swap is atomic it either finishes the protocol or initiate refunds in case any party doesn't follow through the whole protocol. The concept of atomic cross-chain swaps is well known to the blockchain community and was discussed previously in Section 1.3 (Related Work). All following concepts and models discussed are based on Herlihy's work [29]. A cross-chain swap is modeled as a directed graph $\mathcal{D}$, the vertexes of the digraph represent the parties and its arcs are proposed asset transfers. For any pair$(\mathcal{D},L)$, where $\mathcal{D} = (V,A)$ is a strongly-connected directed graph and $L \subset A$ a feedback vertex set for $\mathcal{D}$. The three-way swap in specific has to be an acyclic and utilitizes a slightly modified form of a digraph introducing followers and leaders and using a form of hashed timelock contracts, read more in Section 2.3.2. The concept of atomic cross-chain swaps removes the intermediate party, which is usually an exchange marketplace for cryptocurrencies to trade assets or tokens. This chapter starts with an example of a three-way swap (Section 2.3.1), discussing the model of hashlocks and hashkeys (Section 2.3.2) and at last propose a general protocol (Section 2.3.3) for swaps.

2.3.1 *Example*

Lets assume Carol lifes in a state where car titles are managed on a blockchain and wants to sell her car for bitcoins. Alice is interested in Carol's car, but wants to pay in another cryptocurreny(following called alt-coin). Lucky for both that Bob is willing to trade his alt-coins for bitcoins. Since none of the parties trust each other, they can arrange a three-way swap to assure their transaction is atomic across the different blockchains. To complete the transaction Alice has to send her alt-coins to Bob, so he can transfer his bitcoin to Carol who finally gives the car to Alice. If all parties stick to the deal and follow the protocol then everything is fine. To assure that no one looses their funds due to a malicious actor who behaves irrationally this chapter and the following discusses the protocol of atomic cross-chain swaps by the example of Alice, Bob and Carol's transaction. Today many blockchains come with smart contracts to execute a transfer of funds if certain conditions are met. Let $H(\cdot)$ be a cryptographic hash function. Alice could transfer her alt-coins to a smart contract which acts as escrow on the alt-coin blockchain, the contract also holds the information of hashlock h and timelock t. The hashlock h is determined by a contract value s, called a secret and let $h = H(s)$. If Alice sends the secret to Bob it allows him to claim the funds sitting in the smart contract's escrow using hashlock h. Timelock t means that Bob also needs to

produce a secret before the time t elapsed, if he does not manage to produce the secret in time, then the funds on the alt-coin blockchain are transferred back to Alice by the smart contract. Let Δ be enough time for the parties to publish their smart contracts or detect changes. Since Alice, Bob and Carol need to execute a three-way swap illustrated in figures 2.4 and 2.5 they need to apply to the following simple protocol:

- Alice publishes a smart contract on the alt-coin blockchain including a secret s, $h = H(s)$ she created. The contract also holds the information of hashlock h and timelock 6Δ in the future, to send her alt-coins to Bob.
- As soon as Alice's contract is confirmed on the alt-coin blockchain and detected by Bob, he deploys a smart contract on the bitcoin blockchain to transfer his funds to Alice. Bob adds the same hashlock h value to the contract like Alice and a timelock 5Δ in the future.
- When Carol detects Bob's confirmed smart contract on the bitcoin blockchain, she publishes a smart contract on the car title blockchain in order to transfer the title of her car to Alice. Again with same hashlock h, but timelock 4Δ in the future
- After Alice confirms Bob's mined contract on the title blockchain, she sends s to Carol's contract. Alice aquires the title and reveals s to Carol.
- Carol then sends s to Bob's contract, she aquires the bitcoin and reveals s to Bob.
- Bob completes the swap by sending s to Alice's contract and claiming the alt-coins.

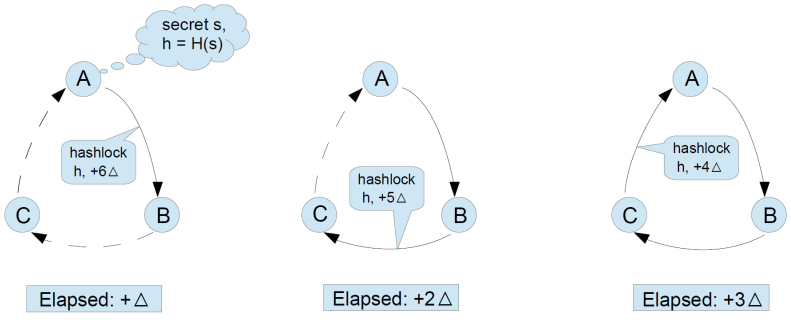

Figure 2.4: Atomic cross-chain swap: deploying contracts

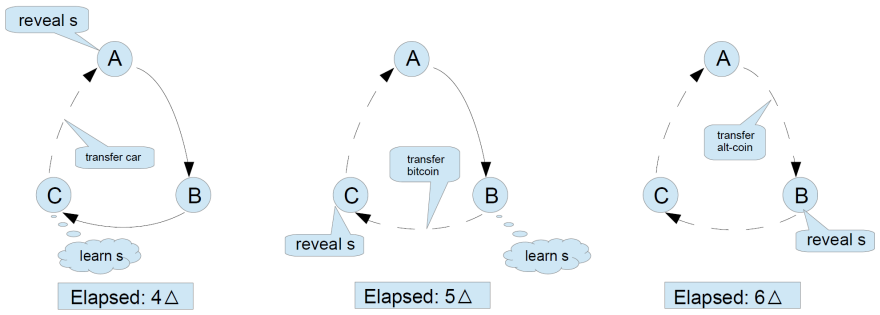

Figure 2.5: Atomic cross-chain swap: triggering arcs

So what could possibly go wrong in this three-way swap? If any party fails to publish their contract during deployment phase, then all contracts timeout and trigger refunds of the assets. If any party fails while triggering arcs, then only that party ends off worse. For example, if Carol halts execution of the swap without triggering her contract, then Alice gets the car and Bob gets a refund. So in that case Carol would only harm herself. Also the order of contract's deployment matters. Because if Carol posts her contract with Alice, before Bob posts his contract with Carol, then Alice could claim the car title without paying Carol. The timelock values are also important. For example, if Carol's contract with Bob and Bob's contract with Alice would expire at the same time, then Carol would reveal s to collect Bob's bitcoin at the very last moment, leaving Bob no time to claim his alt-coins from Alice. If parties behave irrationally, then Alice might end up not getting her car. This would be the case if Alice (irrationally) reveals s before phase completion, then Bob can take Alice's alt-coin and Carol might be able to take Bob's bitcoin.

2.3.2 *Hashlocks and Hashkeys*

In a simple two-party swap each participating party publishes a contract that assumes temporary control of the assets. This hashed timelock contract [65] stores a pair (h, t) and holds a secret s. This contract triggers and releases funds if $h = H(s)$ is available before time t has elapsed. If the contract does not receive the secret before time t is elapsed, then the assets are refunded and the trade is cancelled. The example discussed earlier is a three-way swap, where each arc holds a single hashlock h and timelock t. Timeouts are assigned in a way that each entering arc of a follower v ensures that the timeout for the leaving arc is at least Δ ahead of the arc entering v. This gap ensures that if any of the digraph's arcs leaving v is triggered so v has enough time to trigger every entering arc. For example, if a cross-chain swap digraph has only one leader $\hat{v}$, then the follower's subdigraph is acyclic. In the case of a three-way swap, the one that was discussed earlier in the example subsection, the timeout on arc(u, v) can be $(diam(\mathcal{D}) + D(v, \hat{v}) + 1) \cdot \Delta$. The lefthand side of figure 2.6 illustrates the length of the longest path $D(v, \hat{v})$ from v to the unique leader $\hat{v}$.

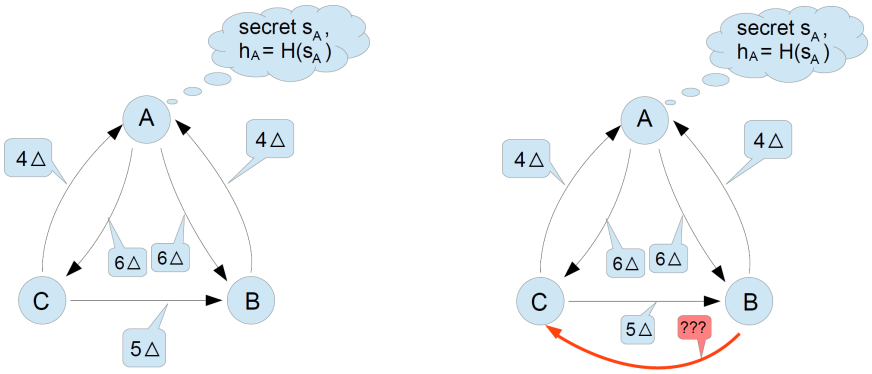

Figure 2.6: A is the leader, B and C followers. Timeouts can be assigned when the follower subdigraph is acyclic (left) but not when it is cyclic (right)

Unfortunately we have to assign timeouts across a cycle that guarantees a gap of at least Δ between entering and leaving arc, due to the fact that there is more than one leader this formular does not work in this case. This is because the subdigraph of any leader's follower has a cycle (see righthand side of figure 2.6). The general timed hashlock digraph can be replaced with a slight different version including leaders and followers to get a mechanism that allows us to assign different timeouts to different paths though. The modified version of the digraph that we need to include the leaders and followers model is described next. Let $L = \{v_0, ..., v_n\}$ be the feedback vertex set for digraph $\mathcal{D}$, called the leaders. Each leader v_i creates a contract with secret s and a hashlock value $h_i = H(s_i)$, yielding a hashlock vector $(h_0, \ldots, h_n)$ with values assigned to each arc. A hashkey for h on arc(u, v) is a triple(s, ρ, σ) with s called the secret $h = H(s)$, ρ is a path $(u_0, \ldots, u_k)$ in digraph $\mathcal{D}$ where $u_0 = v$ and u_k is the leader who initially created the secret s, and $\sigma = sig(\cdots sig(s, u_k), \ldots, u_0)$ is the result, in case all parties have successfully signed s in their path. After the start of the protocol hashkey (s, ρ, σ) time out at $(diam(\mathcal{D}) + |\rho|) \cdot \Delta$ and unlock h on (u, v) as long its available before it times out. Each arc triggers, when all needed hashlocks are unlocked. The hashlock itself times out on an arc, when all hashkeys on the current arc have timed out. Figure 2.7 illustrates the partial hashkeys for a two leader digraph.

2.3.3 *The Protocol*

The protocol for atomic cross-chain swaps has two phases. In phase one the instances of the swap contract are propagated through the digraph $\mathcal{D}$, starting with the leaders. The implementation and its code are discussed later in chapter 4. As soon as a party observes a published contract the current party verifies the contract for correctness of the swap, if verification fails the party abandons the contract. Below the phase one protocol for leaders and followers is shown:

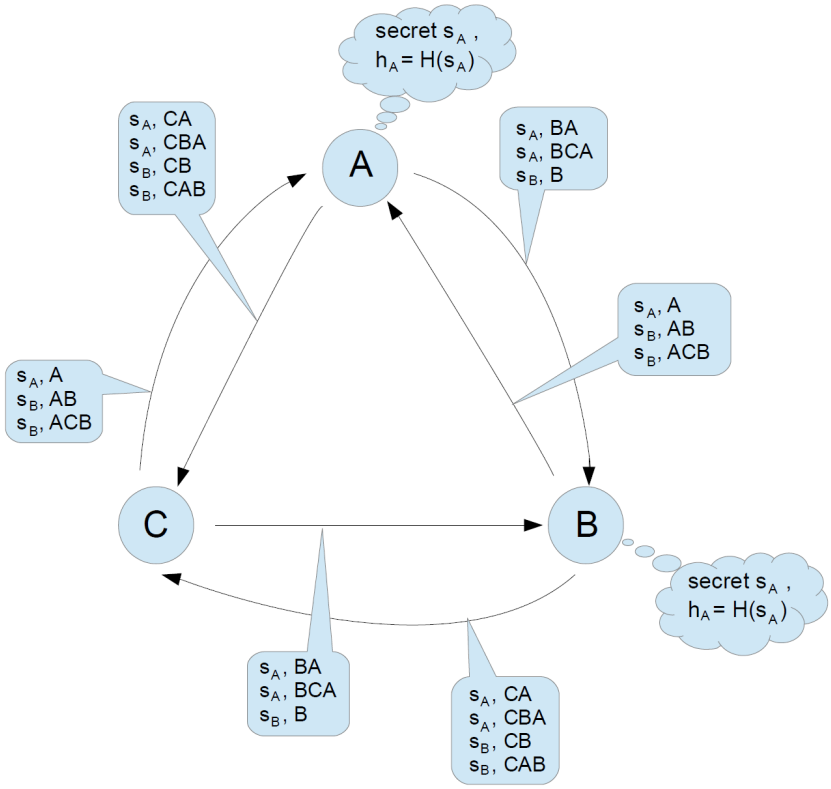

Figure 2.7: Hashkey paths for arcs of two-leader digraph

Protocol for the leaders:

1. Publish a contract on every arc that leaves the leader party, then
2. wait until contracts have been published and mined on all arcs entering the leader.

Protocol for the followers:

1. Wait until correct contracts are published and mined on all arcs entering the vertex, then
2. publish the contracts on all arcs leaving the vertex.

For an example refer to figure 2.7, which shows how contracts are propagated in a two leader swap digraph.

In phase two secrets are disseminated via hashkeys between the participating parties of the swap, while the contracts are propagated from party to counterparty in the direction of the arcs, hashkeys propagate in the opposite direction from counterparty to party. To acquire the assets held by the contracts, each party is motivated to trigger the contracts on entering arcs. In order to complete a swap and come to an understanding how the secret s_i generated by leader v_i is propagated we can trace it through the phase. At the start of phase two v_i calls unlock(s_i, v_i, $sig(s_i, v_i)$) on each contract of an

entering arc. In this case the function's arguments are the hashkeys and v_i is a degenerate path. If any other party v observes that hashlock h_i has been unlocked by a call to unlock(s_i, ρ, σ) on a leaving arc's contract fro the first time, then is calls unlock(s_i, $v + \rho$, $sig(\sigma, v)$) at each entering arc's contract. The propagation of s_i is complete, when h_i has been unlocked on all arcs or if h_i has timed out. Herlihy has formally proven that the following statements apply to the atomic cross-chain swap protocol [29]:

- Assuming that all parties stick to the protocol, the time for triggering every contract from when the protocol started is $2 \cdot diam(\mathcal{D}) \cdot \Delta$.
- None of the conforming parties ends up UNDERWATER (loosing assets or ending up worse).
- The space complexity is $O(|A|^2)$, measured as the number of bits stored on all blockchains for $\mathcal{D} = (V, A)$ with leaders $L \subset V$.
- The set L of leaders is a feedback vertex set in $\mathcal{D}$ for any uniform swap protocol based on hashed timelocks.

3

REQUIREMENT ANALYSIS AND CONCEPT

Alice, Bob and Carol want to trade assets they own on different blockchains. Since they do not trust each other they need a solution where the trade is either executed completely or reversed. They want their assets to be returned in case a party behaves irrationally. For example if Carol tries to scam them, Alice and Bob do not want to loose their assets and end up worse in the swap process. The example 2.3.1 discussed in chapter model and general principles 2 is the user story that is addressed by the prototype implementation. Requirements to implement a system feasible of solving their initial situation in an atomic matter is discussed next. First the blockchain needs to support turing-complete smart contracts to hold information during the trade and to decide weather the swap contract is unlocked or timed out. By holding these information the contract can decide if the locked funds are eligible to be claimed by the counter party or if the party can claim a refund in case the contract is timed out. Second is a digraph implementation as a smart contract to execute the swap decentralized. For simplicity delegation can be handled by the middleware, by splitting the trade into two phases that are discussed in chapter 4. Third requirements are the hashlocks and timelocks. The correct hashlock is needed to unlock the contract by appending it to the secret revealed by the party leading the digraph paths. To unlock the contract before it times out timelocks passed to the unlock function need to be smaller than the prematurely added timelocks that are held by the contract state. If the unlock limits are surpassed the contract times out and allows the party which locked their funds on contract creation to reclaim them. For full decentralization the hashlocks and timelocks need to be created by the smart contract, but for simplicity the middleware creates them. By passing the values to the smart contract's claim and refund functions as parameter it can decide if the phases are completed in the correct time windows or if the trade is canceled. The UML use case diagram 3.1 below shows interactions between the actors and systems. In a productive environment the participating parties would be outside the system, but again for simplicity the implemented prototype handles all steps of the actors to complete the swap. The implementation chapter 4 discuss the realization of the use case diagram in detail. Including an UML class diagram and abstraction of the swap digraph.

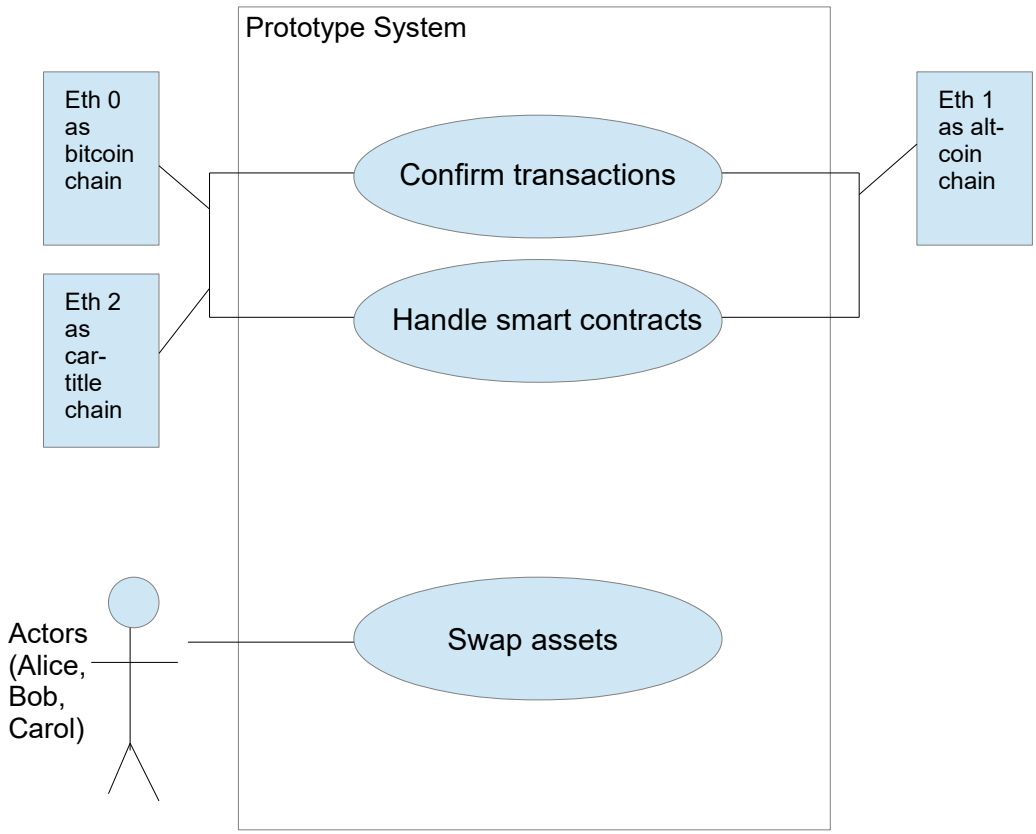

Figure 3.1: UML Use Case Diagram

4

IMPLEMENTATION

The prototype presented in this chapter aims to implement the three-way swap example of Alice, Bob and Carol discussed earlier in chapter 2.3.1. Implementing the protocol from chapter 2.3.3 to execute any kind of swap would be too much work for this thesis. For simplicity this work uses three ethereum blockchains posing as the bitcoin, alt-coin and car-title blockchain. All communication with the blockchains is handled through the Geth client [69]. Also for simplicity certain parts of Herlihy's proposed smart contracts are executed off-chain, such as the generation of the hashlock and timelock values and it's verification. The prototype utilitizes class Pools, which holds a wallet one each chain containing ethereum accounts with mined ether to prepare the initial situation of Alice, Bob and Carol. The Pools class also serves another purpose, due to a problem that came up during the implementation. Ether moved by a smart contract are executed as internal transactions, which are only held by the state of the EVM's stack and thus are not visible on an account's balance. To overcome this issue the class Pools sends one ether upon a successful call of the claim() function of the corresponding smart contract to the callers's account, to make the transfer visible during run time. There are methods to trace internal transactions of a smart contract by instrumenting the EVM [32], which is discussed in chapter **??**. In order to delegate the actions made by the participating parties a java middleware is needed, which implements the wallets and communicates with the blockchains through the geth clients. It also prepares the initial situation for the parties mentioned in the example by controlling the pool accounts to distribute assets to the participating parties. Classes Alice, Bob and Carol represent the parties and have control of their assets in order to send ether and deploy contracts. The middleware uses the web3j library [71] to communicate with the geth clients. The class HashLock generates hashlock h with the given formula $h = H(s)$, where $H(\cdot)$ is a cryptographic hash function using the SHA-256 hashing algorithm. It takes the secret s with the appended current block and resolves to the hashed secret s (hashlock h), which can be reached only with the correct secret and block. Smart contracts can determine the current time only by the system state, meaning the contracts can only measure time by the current block number in the blockchain e.g. block # 10 is later than block # 5. To measure a real time like Universal Time Coordinated (UTC) the implementation still needs to be off-chain with a corresponding time server. Class TimeLock generates start times and holds information and saves them into the published smart contract, when phases have to be completed in order to execute the swap protocol. The middleware needs a smart contract solidity file, so all parties can deploy and publish the contract onto the blockchains. Class Digraph delegates the swap protocol for the parties. Instances Alice,

Bob and Carol representing the actors are passed to Digraph's constructor upon initialization. Paths are triggered by instance Digrap by the according string representations (table 4.1), which then call the functions in a given order and executes the swap in behalf of Alice, Bob and Carol. The swap is executed in two phases, called Phase 0 (contract deployment and locking of funds figure 4.1) and 1 (revealing secrets figure 4.2).

①	String s = "AB"	Alice deploys on alt-coin chain
②	String s = "BA"	Bob deploys on bitcoin chain
③	String s = "CC"	Carol deploys on car-title chain
④	String s = "AC"	Alice unlocks Carol's contract and claims car-title
⑤	String s = "CA"	Carol unlocks Bob's contract and claims bitcoin
⑥	String s = "BB"	Bob unlocks Alice's contract and claims alt-coin

Table 4.1: Digraph Path String Representations

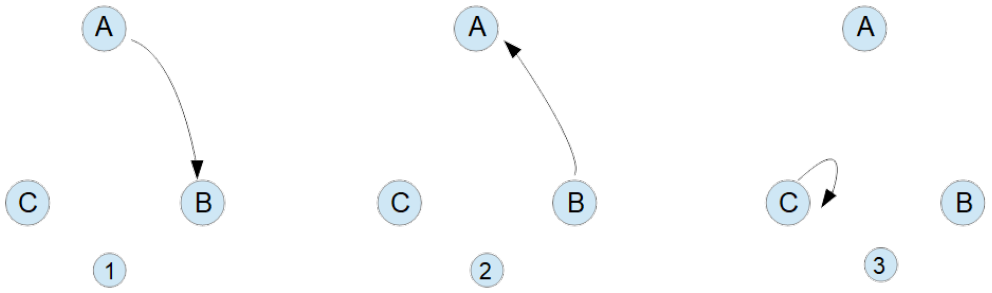

Figure 4.1: Swap Protocol Phase 0

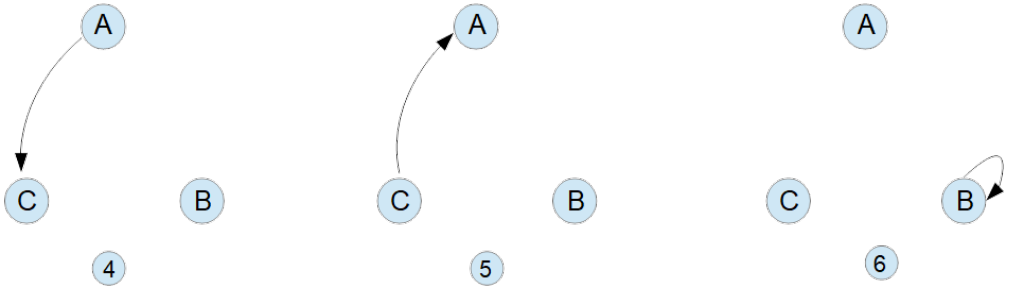

Figure 4.2: Swap Protocol Phase 1

4.1 REFERENCE ARCHITECTURE

This chapter discusses the reference architecture (figure 4.3) resulting from the swap prototype implementation. Alice, Bob and Carol are the actors and represent any user in a productive environment. In the swap prototype implementation actors are controlled and delegated by the middleware, in the case of the reference architecture actors would be real users though and each of them would be responsible to stick to the swap protocol by themselves. This includes the two phases, meaning actors have to deploy contracts and send funds in the correct order, as well as revealing the secret and claiming the traded funds. If any of the the actors do not apply to the protocol, the swap is canceled and funds are returned to the actors so no party ends up worse in the case of malicious behavior. The java middleware implements the web3j library to communicate with the ethereum geth clients, which then handle all interactions with the blockchains. In a real productive environment the three ethereum chains can be swapped with any blockchain that supports smart contracts and turing-completeness. Implementation of additional libraries to communicate with the corresponding blockchain is needed. The reference architecture shown below can be used as foundation for universal swap systems to support any kind of swap. To get an architecture for a two-way swap this can easily be done by detaching actor 2 and chain 2 from the reference architecture for example.

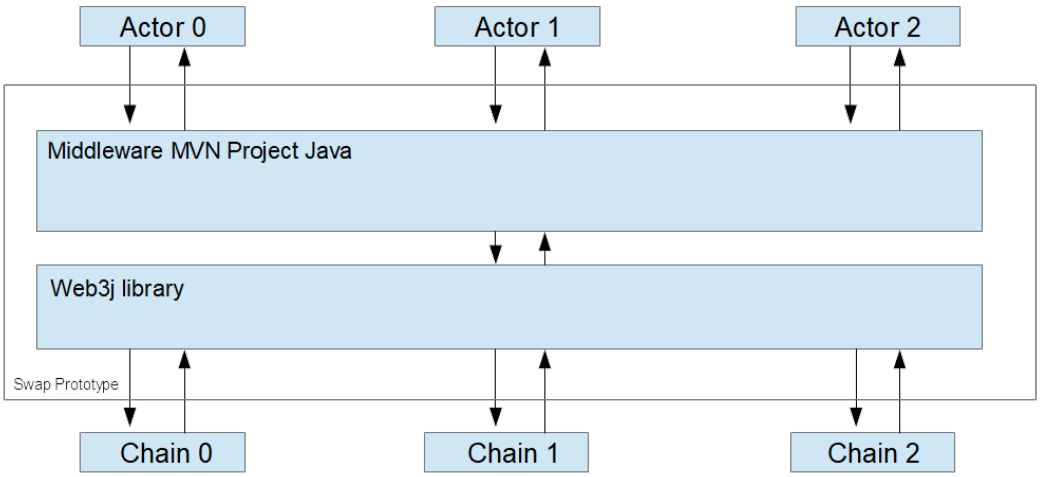

Figure 4.3: Swap Protocol Reference Architecture

4.2 SWAP CONTRACT

The swap contract (figure 4.4) holds several attributes (line 4 - 9) to force correct execution of the swap protocol, which are passed to the constructor (line 11 - 21) upon initialization. The contract holds the addresses of the party and counter party. The contract is always deployed by each actor on the blockchain he owns the assets to be traded in the swap process. The timelock and start values are saved in the swap contract's state to ensure that the participating parties stick to the protocol. With these attributes the swap contract can decide if assets are eligible either for a claim or refund. The hashlock attribute ensures that funds can only be claimed by unlocking all values in the unlocked bool array. To unlock the entries in the array the counter party needs the correct revealed and hashed secret. Each entry of the unlocked array is initialized false during initialization by the swap contract constructor. The unlock function (line 24 - 29) can only be called by the counter party, which was saved in the swap contract's state upon initialization. Entries in the unlocked array are only set to true by passing the correct hashlock and time. If the timeNow value passed to the function is greater than the saved timelock value in the contract's state, then the unlocked entry is not set to true and prevents the funds being claimed by the counter party. The isUnlocked function (line 32 - 34) returns the bool value of the unlocked array for the passed function's argument entry i. The lockEther function (line 36 - 38) is called by the party after successful deployment of the swap contract to lock the funds offered for trade in the contract. The refund function (line 40 - 46) takes timeNow as an argument and can only be called contract's owner or party that deployed the swap contract. If timeNow value is greater than the last entry in the timelock array, then the funds are returned to the contract's owner and the trade is canceled. The claim function (line 48 - 53) can only be called by the counter party saved in the swap contract's state upon initialization. Funds are only released to the counter party, if all entries in the unlocked array are set to true. In the case of Alice, Bob and Carol each party deploys its own contract on the blockchain they offer funds to be swapped in the process of protocol execution. The contract shown below is written in solidity and is only compatible with ethereum blockchains, but the code logic can be transferred to any other blockchain that supports smart contracts to execute swaps between other types of blockchains.

```
pragma solidity >=0.4.25;

contract Swap {
    address payable party;
    address payable counterParty;
    uint[] timeLock;
    bytes32[] hashLocks;
    bool[] unlocked;
    uint start;

    constructor(address payable _party,
                address payable _counterParty,
                uint[] memory _timeLock,
                bytes32[] memory _hashLock,
                uint _start) public {
        party = _party;
        counterParty = _counterParty;
        timeLock = _timeLock;
        hashLocks = _hashLock;
        start = _start;
        unlocked = [false, false, false];
    }

    function unlock(uint _i, bytes32 _hashLock, uint _timeNow) public {
        require (msg.sender == counterParty);

        if (_timeNow < timeLock[_i] && hashLocks[_i] == _hashLock) {
            unlocked[_i] = true;
        }
    }

    function isUnlocked(uint _i) public view returns (bool) {
        return unlocked[_i];
    }

    function lockEther() public payable {
        require (msg.value == 1000000000000000000 && msg.sender == party);
    }

    function refund(uint _timeNow) public {
        require (msg.sender == party);

        if (_timeNow > timeLock[3]) {
            party.transfer(address(this).balance);
        }
    }

    function claim() public payable {
        require (msg.sender == counterParty);

        if (unlocked[0] == true && unlocked[1] == true && unlocked[2] == true) {
            counterParty.transfer(address(this).balance);
        }
```

Figure 4.4: Swap Contract

4.3 MIDDLEWARE (INTERMEDIATE PARTY)

This chapter discusses the middleware of the swap prototype, it's classes and shows an UML diagram (figure 4.5) on the next page. Class EthNode holds information for each ethereum node like the name, host and port. These values are needed to for initialization of the Web3j object, which handles the communication with the geth clients to interact with the blockchains. Class Wallet retrieves the Web3j object from the EthNode class it wants to interact with. The Wallet class wraps all functions of the Web3j library that are needed to execute the swap protocol and implements an API that can be extended with other libraries to support different blockchains. The current implementation of the swap prototype only supports ethereum chains though. Class Pools is a helper to prepare the initial swap situation of Alice, Bob and Carol. It holds wallets on each of the three blockchains with enough funds to supply the participating parties with the assets they intend to swap during runtime. The class is also used to make the internal transactions of the smart contracts visible on the balance by sending additional funds. Class HachLock generates the hashlock h with the secret s which is passed to the constructor when instantiated. By calling the function getHashLock-AsByteArrayWithCurrentBlockAppended and passing the current block as function's argument, it digests the hashlock h as a byte array. Class TimeLock is instantiated with the current time and the time values, which determine the window for completion of the phases 0 and 1. It generates all Date values to enforce that each contract receives the correct times in which the swap phases have to be executed. The Digraph class delegates the swap protocol for the participating parties. It triggers the paths one after another in the correct order and handles all actions of Alice, Bob and Carol to fully execute the swap. Package Parties holds the classes Alice, Bob and Carol and has full control over their assets. In a productive environment these classes would be real users and the actor classes could be abandoned from the swap prototype. Class SwapProtocol instantiates the classes Alice, Bob, Carol and Digraph. The parties hold their private keys as attributes and connect to their nodes during initialization, when al parties are ready and the initial trade situation is prepared the class Digraph triggers the paths in the correct order to execute the swap protocol. A simple console interface is provided by the class App, where the user can choose between running different tests or execute a three-way swap. For simplicity hashlock h, timelock t and digraph D are generated off-chain by their corresponding classes.

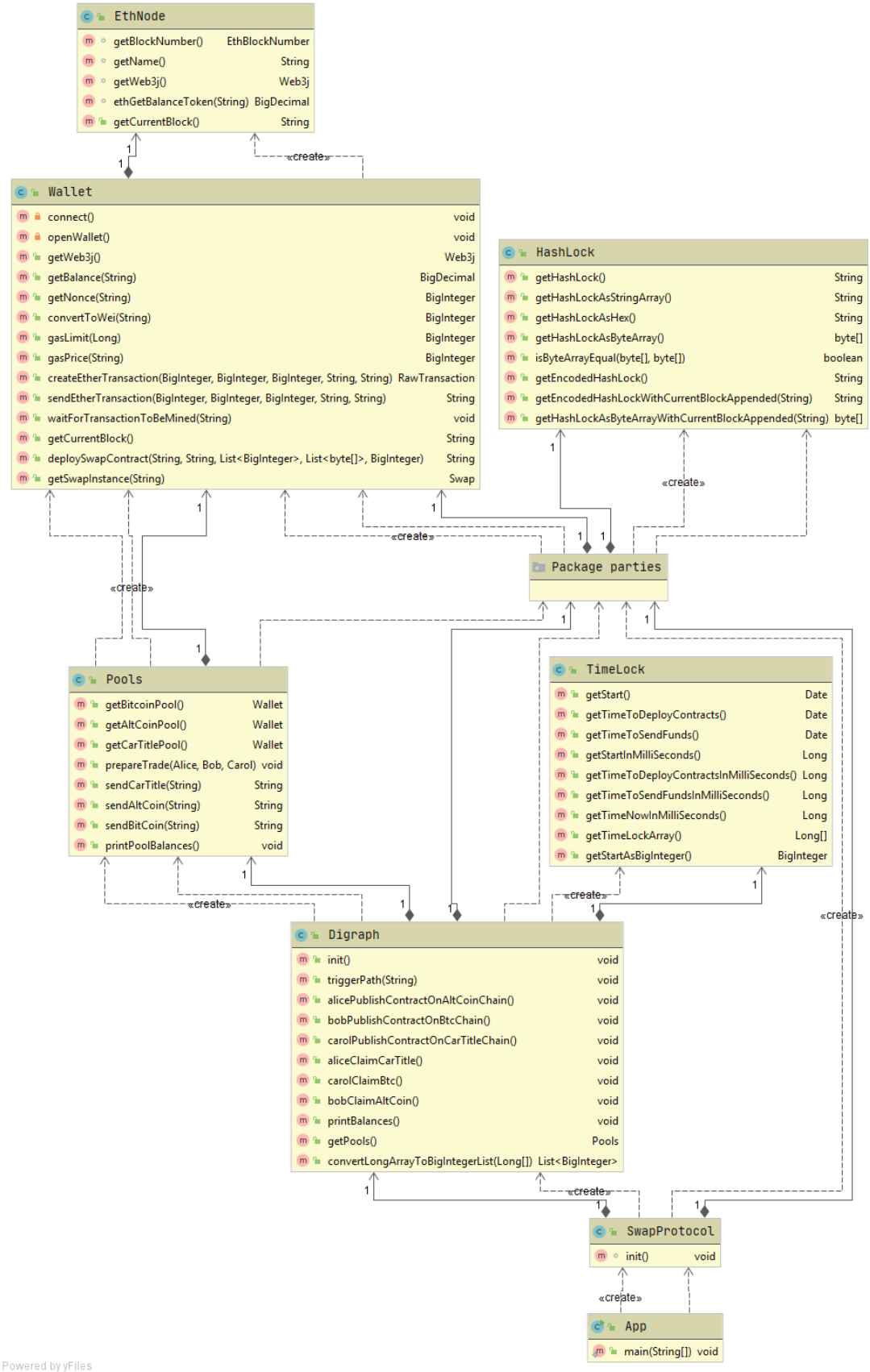

Figure 4.5: UML Class Diagram Middleware

4.4 TESTS (EVALUATION)

Testing is difficult due to a problem that came up during implementation of the swap prototype. The EVM handles funds sent by a smart contract as internal transactions, which do not show up on the recipients account balance. Information who owns the coins after sending it is only available in the contract's state. Making them visible requires tracing the EVM's stack on assembly level to keep track on who owns the funds. The prototype implementation uses a workaround to simulate the result of a claim or refund function call of the swap contract. It does that by sending funds from one of the pool accounts to the caller's address to make the transaction visible on the balance. This influences execution times of the prototype and test results. Since three chains and the prototype are all running on the same system the test results do not represent a productive environment. To reproduce the testing results an equivalent system is required, where the swap prototype and chains are hosted on. Nevertheless test results of the implemented system are discussed next. Swaps per hour (SPH) is calculated by measuring the time one swap needs to complete. One run includes preparation of the Alice, Bob and Carol's initial situation, connecting to wallets and the workaround discussed above. The values shown in table 4.2 result in approximately eight swaps per hour by dividing one hour in nanoseconds through the average time of the three runs. Transactions per second (TPS) are measured by sending ten transaction and measure the time they need until they are published and confirmed on the blockchain. By dividing ten through the average time in seconds of table 4.3 the transactions per second on one chain is approximately 0.049 TPS.

Iteration	Time in nanoseconds	Time in minutes
Run 1	377392865700 ns	6.29 min
Run 2	435784403900 ns	7.26 min
Run 3	423182624900 ns	7.05 min
Average	412119964833 ns	6.87 min

Table 4.2: Test results swaps per hour (SPH)

Iteration	Time in nanoseconds	Time in seconds
Run 1	56932675400 ns	56.93 sec
Run 2	55702292100 ns	55.70 sec
Run 3	71745206200 ns	71.75 sec
Average	61460057900 ns	61.46 sec

Table 4.3: Test results transactions per second (TPS)

5

CONCLUSION AND FUTURE RESEARCH

The atomic cross-chain prototype discussed in chapter 4 shows that swapping assets across different is possible with the use of smart contracts. To reach full decentralized blockhain interoperability more needs to be done though. For simplicity the prototype handles delegation of the trade and generation of the hashlocks and timelocks off-chain. By adding these features as smart contract implementations the prototype can be further decentralized. It remains an open question if bitcoin can implement swaps with the current scripting language it supports, because the lack of turing-completeness is a problem. To implement and generate the timelocks on-chain there needs to be evaluation on how time can be measured in block generation times, because the EVM only knows the time by the current world state and past blocks that have been added. Hyperledger implements their approach with Hashed-Timelock Agreements [70], which are based on the timelock [66] principle of bitcoin. After successful decentralization of delegation and generation of timelocks and hashlocks on-chain next step is supporting other cryptocurrencies that support smart contracts. Further evaluation is needed to see if different consensus algorithms bring up more problems to be solved for executing swaps. A list of cryptocurrencies that have smart contract support created by cryptoslate [68] shows that it needs a lot more research on this topic to reach full interoperability of these blockchains. Before even adding support for other cryptocurrencies to the prototype further development of ethereum's clients and libraries has to be done to make internal transactions visible to the end user by instrumenting the EVM [33]. Also the prototype does not support two-way swaps of assets between two chains at this time. Solving the previous mentioned tasks and the problems that come up during further evaluation of these, the next logical step is to support more cryptocurrencies to create a framework that supports any kind blockchain teechnologies.

6

TERMINOLOGY

Address:	A 160-bit code used for identifying Accounts.
Account:	Accounts have an intrinsic balance and transaction count held as part of ethereum state. Those accounts with empty associated EVM code are controlled by external entities and those with non-empty EVM code (thus the account represents an autonomous object). Every Account has a single address that identifies it.
Alt-Coin:	The term Alt-Coin refers to all cryptocurrencies launched after the success of bitcoin. Generally, they sell themselves as better alternatives to bitcoin or try to solve certain technological problems in a "better" or different way.
App:	An app is a type of software that allows you to perform specific tasks. Its also an acronym for application. Its often named after the device the application is running on. E.g. mobile app, desktop app or Decentralized Application (DApp). Decentralized apps are a referral for applications running on decentralized architectures like the EVM.
Asset:	An asset is a resource with economic value, owned or controlled by an individual, corporation or country. In financial context it comes with the expectation that it provides a future benefit. In context of this work it can also be a car title owned by an individual like Carol.
Atomic Transaction:	Atomic Transactions are associated with Database operations where a set of actions must all complete or else none of them complete (atomicity). Same concept applies to swaps in blockchain context.
Autonomous Object:	A notional object, which is existent only in the hypothetical state of the ethereum public chain. It has an address and thus an associated account, which has non-empty EVM code. Incorporated only as the Storage State of that account.

Block:	A block is a set of transactions, smart contracts and meta data, appended to the previous block in the blockchain. It has to be verified (mined) by the nodes of the network to become valid.
Block Time:	It defines the time that it takes for a block to get confirmed (mined) by participants of the network. In case of ethereum there is an expected block time of 10 to 19 seconds.
Bytecode:	Bytecode is the language which the EVM understands and can execute. Smart contracts are often written in a high level programming language like solidity and compiled down to bytecode, which consists of a series of bytes, where each byte represents an operation for the EVM to execute. Its often referred to as EVM Code.
Cryptocurrency:	A cryptocurrency is a digital asset designed to work as medium of exchange and most of the time not issued by any state or central bank. The individual coin ownership are stored in a ledger, using strong cryptography to secure transaction records, to control the creation of additional coins, and to verify the transfer of coin ownership.
CryptoNote:	CryptoNote is an application layer protocol which aims to solve specific problems of bitcoin. These are respectively traceability of transactions, the PoW function, irregular emission, hardcoded constants and bulky scripts. The protocol is the basis of several decentralized privacy-oriented cryptocurrencies.
Contract:	Generally a contract is a written or spoken agreement, especially one concerning employment, sales, or tenancy, that is intended to be enforceable by law. In ethereum or cryptocurrency environment a contract is more meant to be a financial contract and mostly referred to as smart contract.
External Actor:	An external actor in the context of this thesis is an entity, which resides and acts outside or inside of the cryptocurrency system, with potential malicious behavior.

Ether:	Ether (short ETH) is the internal cryptocurrency of the ethereum blockchain. It can be transferred from account to account, but is also used to pay the gas fees for smart contracts and their execution of functions.
Ethereum Virtual Machine:	Ethereum provides a decentralized virtual machine, called the EVM. It can execute scripts by using an international network of public nodes, in case of the ethereum public chain. The virtual machine's instruction set is Turing-complete.
Gas:	Its the fundamental network cost unit and can be converted freely from and to Ether as required. Gas does not exist outside of the computational engine of ethereum, its price is set by the transactions. Miners are free to ignore transactions, which have a too low gas price.
Genesis Block:	The genesis block is the first ever mined or origin block of a public or private ethereum network. It contains all essential information to configure the network and as well how to find related peers.
Hash:	Is the result of any hash function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are always the same result of same input and are called hash values, hash codes, digests or simply hashes.
Padding:	All six SHA functions (SHA-0, SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512) start by padding the message according to the so-called Merkle-Damgård strengthening technique. The message is appended with a binary one and right-padded with a variable number of zeros, followed by the length of the original message coded over two binary words. The total padded message length must be a multiple of the message block size of the according function.

Protocol:	Generally a protocol is a system of rules how a computer and its developers can connect to, participate in and transmit information over a system or network. These instructions define code syntax and semantics that the system expects. Ethereum's protocol is built and designed for decentralized applications, with emphasis on rapid development time, security and interactivity.
Public Key:	In ethereum context the public key is derived from the private key using Elliptic Curve Digital Signature Algorithm (ECDSA). Its also a synonym for the address.
Public Blockchain:	A public blockchain is a distributed and decentralized open network. Anyone can download the protocol and read, write or participate in it. Transactions are recorded as blocks and linked together to form a chain.
Private Key:	A private key is a large random number, which is also used to create the public key or address. The private key allows the user to sign and send transactions. Whoever owns the private key has full control over the ether associated with the account.
Private Blockchain:	A private blockchain is a completely private blockchain, which is isolated from other public blockchains. Private chains are mainly created by organizations to restrict the read and write permissions, only nodes with the right permissions can access these blockchains.
Storage State:	Storage state is the information particular to a given account, which is maintained between the times that an account runs associated EVM Code.
Solidity:	Solidity is a high-level and object-oriented programming language for implementing smart contracts. The code gets compiled down to bytecode for the ethereum blockchain to execute.

Transaction:	A transaction in ethereum context is the way an external actor interacts with the ethereum network, if someone wants to modify or update the state stored in the blockchain.
Token:	Ethereum tokens are simply digital assets, which are being built on top of the ethereum blockchain. They can also strengthen ethereum's ecosystem, since ether are needed to execute the smart contracts on which the tokens a built on.
Miner:	The process in which transactions are verified and added to the blockchain digital ledger is called cryptocurrency mining. The external actor or entity that runs the node in the network is called a miner.
Nonce:	The nonce is the number of transactions sent from a given address, each time a transaction is sent from a given account the nonce is increased by one. Its specific value before and after increasing can only be used once.
On-chain:	An on-chain transaction (synonym for transaction), occurs and is considered valid when the blockchain is modified to reflect the transaction on the public ledger. Smart contracts and all pieces of code, that get executed by the EVM for example are considered to be on-chain.
Off-chain:	An off-chain transaction takes the value outside of the blockchain. It can be executed using multiple methods and patterns. Off-chain implementations are all kinds of software, which takes some part in transaction or data verification off the blockchain.

BIBLIOGRAPHY

[1] ISO/IEC 10118-3. *Information technology - security techniques - hash functions - Part 3: Dedicated hash-functions*. Tech. rep. 2003.

[2] David J Abraham, Avrim Blum, and Tuomas Sandholm. "Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges." In: *Proceedings of the 8th ACM conference on Electronic commerce*. 2007, pp. 295–304.

[3] Kurt M Alonso. *Zero to Monero*. 2020.

[4] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. "Enabling blockchain innovations with pegged sidechains." In: 72 (2014). URL: http://kevinriggen.com/files/sidechains.pdf.

[5] Adam Back et al. "Hashcash-a denial of service counter-measure." In: (2002).

[6] Jørgen Bang-Jensen and Gregory Gutin. "Theory, algorithms and applications." In: *Springer Monographs in Mathematics, Springer-Verlag London Ltd., London* 101 (2007).

[7] Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. "Caught in chains: Claim-first transactions for cross-blockchain asset transfers." In: *Technische Universität Wien, Whitepaper* (2018).

[8] Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. "DeXTT: Deterministic Cross-Blockchain Token Transfers." In: *IEEE Access* 7 (2019), pp. 111030–111042.

[9] S. Bowe and D. Hopwood. *Hashed time-locked contract transactions*. 2020. URL: https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki (visited on 07/25/2020).

[10] Ethan Buchman. "Tendermint: Byzantine fault tolerance in the age of blockchains." PhD thesis. 2016.

[11] Vitalik Buterin. "Chain interoperability." In: *R3 Research Paper* (2016).

[12] Vitalik Buterin. "On sharding blockchains." In: *Sharding FAQ* (2017).

[13] Vitalik Buterin et al. "Ethereum white paper." In: *GitHub repository* 1 (2013), pp. 22–23.

[14] Vitalik Buterin et al. "Ethereum: A next-generation smart contract and decentralized application platform." In: (2014). URL: https://ethereum.org/en/whitepaper/.

[15] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance." In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[16] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs & digraphs*. Vol. 39. CRC press, 2010.

[17] Google Code. *Diff-Match Patch*. 2016. URL: https://code.google.com/p/google-diff-match-patch/ (visited on 07/25/2020).

[18] CoinMarketCap. *Cryptocurrencies: 5,784*. URL: https://coinmarketcap.com/ (visited on 07/25/2020).

[19] ConsenSys. *BTC Relay*. URL: http://btcrelay.org/ (visited on 07/25/2020).

[20] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. "Merkle-Damgård revisited: How to construct a hash function." In: *Annual International Cryptology Conference*. Springer. 2005, pp. 430–448.

[21] Gaby G Dagher, Chandra L Adhikari, and Tyler Enderson. "Towards Secure Interoperability between Heterogeneous Blockchains using Smart Contracts." In: *Proceedings of the 2018 ACM symposium on principles of distributed computing. November* (2017).

[22] Chris Dannen. *Introducing Ethereum and Solidity*. Vol. 1. Springer, 2017.

[23] DeCred. *Decred cross-chain atomic swapping*. 2019. URL: https://github.com/decred/atomicswap (visited on 07/25/2020).

[24] Christian Decker and Roger Wattenhofer. "A fast and scalable payment network with bitcoin duplex micropayment channels." In: *Symposium on Self-Stabilizing Systems*. Springer. 2015, pp. 3–18.

[25] John P Dickerson, David F Manlove, Benjamin Plaut, Tuomas Sandholm, and James Trimble. "Position-indexed formulations for kidney exchange." In: *Proceedings of the 2016 ACM Conference on Economics and Computation*. 2016, pp. 25–42.

[26] Matt Franklin and Gene Tsudik. "Secure group barter: Multi-party fair exchange with semi-trusted neutral parties." In: *International Conference on Financial Cryptography*. Springer. 1998, pp. 90–102.

[27] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. "Proof-of-stake sidechains." In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 139–156.

[28] Matthew Green and Ian Miers. "Bolt: Anonymous payment channels for decentralized currencies." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 473–489.

[29] Maurice Herlihy. "Atomic cross-chain swaps." In: *Proceedings of the 2018 ACM symposium on principles of distributed computing*. 2018, pp. 245–254.

[30] Adrian Hope-Bailie and Stefan Thomas. "Interledger: Creating a standard for payments." In: *Proceedings of the 25th International Conference Companion on World Wide Web*. 2016, pp. 281–282.

[31] Zhipeng Jia, Pingzhong Tang, Ruosong Wang, and Hanrui Zhang. "Efficient near-optimal algorithms for barter exchange." In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 2017, pp. 362–370.

[32] Nick Johnson. *Instrumenting EVM*. URL: https://ethereum.stackexchange.com/questions/4446/instrumenting-evm (visited on 07/25/2020).

[33] Nick Johnson. *Instrumenting EVM*. URL: https://ethereum.stackexchange.com/questions/4446/instrumenting-evm (visited on 09/06/2020).

[34] Sandra Johnson, Peter Robinson, and John Brainard. "Sidechains and interoperability." In: *arXiv preprint arXiv:1903.04077* (2019).

[35] Randy M Kaplan. "An improved algorithm for multi-way trading for exchange and barter." In: *Electronic Commerce Research and Applications* 10.1 (2011), pp. 67–74.

[36] Rami Khalil and Arthur Gervais. "Revive: Rebalancing off-blockchain payment networks." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 439–453.

[37] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. "Non-Interactive Proofs of Proof-of-Work." In: *IACR Cryptology ePrint Archive* 2017.963 (2017), pp. 1–42.

[38] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A provably secure proof-of-stake blockchain protocol." In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[39] Aggelos Kiayias and Dionysis Zindros. "Proof-of-work sidechains." In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 21–34.

[40] T. K. Organization. Komodo. "Atomic Swap Decentralized Exchange of Native Coins." In: (2020). URL: https://github.com/SuperNETorg/komodo/wiki/barterDEX-Whitepaper-v2.

[41] Jae Kwon. "Tendermint: consensus without mining (2014)." In: (2014). URL: https://tendermint.com/static/docs/tendermint.pdf.

[42] Jae Kwon and Ethan Buchman. "Cosmos: A network of distributed ledgers." In: (2016), pp. 1–41. URL: https://cosmos.network/whitepaper.

[43] Zujian Li and Zhihong Zhang. "Research and Implementation of Multi-chain Digital Wallet Based on Hash TimeLock." In: *International Conference on Blockchain and Trustworthy Systems*. Springer. 2019, pp. 175–182.

[44] Gregory Maxwell and Andrew Poelstra. "Borromean ring signatures." In: *Accessed: Jun* 8 (2015), p. 2019.

[45] Silvio Micali. "Simple and fast optimistic protocols for fair electronic exchange." In: *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. 2003, pp. 12–19.

[46] Mark Miller. "The Future of Law." In: *paper delivered at the Extro* 3 (1997).

[47] Satoshi Nakamoto and A Bitcoin. "Bitcoin: A peer-to-peer electronic cash system." In: (2008). URL: https://bitcoin.org/bitcoin.pdf.

[48] R. Network. *What is the raiden network?* URL: https://raiden.network/101.html (visited on 07/25/2020).

[49] Shen Noether, Adam Mackenzie, et al. "Ring confidential transactions." In: *Ledger* 1 (2016), pp. 1–18.

[50] Henning Pagnia and Felix C Gärtner. *On the impossibility of fair exchange without a trusted third party*. Tech. rep. Technical Report TUD-BS-1999-02, Darmstadt University of Technology . . ., 1999.

[51] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016.

[52] Peter Robinson, David Hyland-Wood, Roberto Saltini, Sandra Johnson, and John Brainard. "Atomic crosschain transactions for ethereum private sidechains." In: *arXiv preprint arXiv:1904.12079* (2019).

[53] Lloyd Shapley and Herbert Scarf. "On cores and indivisibility." In: *Journal of mathematical economics* 1.1 (1974), pp. 23–37.

[54] Nick Szabo. "Formalizing and securing relationships on public networks." In: *First Monday* (1997).

[55] T.Nolan. *Atomic swaps using cut and choose*. 2016. URL: https://bitcointalk.org/index.php?topic=1364951 (visited on 07/25/2020).

[56] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.

[57] Stephan Tual. *Ethereum Launches*. 2015. URL: https://blog.ethereum.org/2015/07/30/ethereum-launches/ (visited on 07/25/2020).

[58] Nicolas Van Saberhagen. *CryptoNote v 2.0*. 2013.

[59] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of cryptography and security*. Springer Science & Business Media, 2014.

[60] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. "Blockchain technology, bitcoin, and Ethereum: A brief overview." In: *2018 17th international symposium infoteh-jahorina (infoteh)*. IEEE. 2018, pp. 1–6.

[61] Gerhard Weikum and Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001.

[62] Gavin Wood. "Polkadot: Vision for a heterogeneous multi-chain framework." In: *White Paper* (2016).

[63] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger." In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[64] Guy Zyskind, Can Kisagun, and Conner Fromknecht. *Enigma Catalyst: A machine-based investing platform and infrastructure for crypto-assets*. 2018.

[65] bitcoinwiki. *Hashed timelock contracts*. URL: https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts (visited on 07/25/2020).

[66] bitcoinwiki. *Timelock*. URL: https://en.bitcoinwiki.org/wiki/Timelock (visited on 09/06/2020).

[67] bitcoinwiki. *Atomic cross-chain trading*. 2016. URL: https://en.bitcoin.it/wiki/Atomic_swap (visited on 07/25/2020).

[68] cryptoslate.com. *Smart Contracts Coins*. URL: https://cryptoslate.com/cryptos/smart-contracts/ (visited on 09/06/2020).

[69] ethereum.org. *Geth (Go Ethereum)*. URL: https://geth.ethereum.org/ (visited on 07/25/2020).

[70] interledger.org. *Hashed-Timelock Agreements (HTLAs)*. URL: https://interledger.org/rfcs/0022-hashed-timelock-agreements/ (visited on 09/06/2020).

[71] web3j. *web3j: Web3 Java Ethereum Đapp API*. URL: https://github.com/web3j/web3j (visited on 07/25/2020).