Imperial College London
Department of Computing

Trustless Communication Across Distributed Ledgers: Impossibility and Practical Solutions

Alexei Zamyatin

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College, January 2022

Abstract

Since the advent of Bitcoin as the first decentralized digital currency in 2008, a plethora of distributed ledgers have been created, differing in design and purpose. Considering the heterogeneous nature of these systems, it is safe to say there shall not be "one coin to rule them all". However, despite the growing and thriving ecosystem, blockchains continue to operate almost exclusively in complete isolation from one another: by design, blockchain protocols provide no means by which to communicate or exchange data with external systems. To this date, centralized providers hence remain the preferred route to exchange assets and information across blockchains – undermining the very nature of decentralized currencies.

The contribution of this thesis is threefold. First, we critically evaluate the (im)possibilty, requirements, and challenges of cross-chain communication by contributing the first systematization of this field. We formalize the problem of Cross-Chain Communication (CCC) and show it is impossible without a trusted third party by relating CCC to the Fair Exchange problem. With this impossibility result in mind, we develop a framework to design new and evaluate existing CCC protocols, focusing on the inherent trust assumptions thereof, and derive a classification covering the field of cross-chain communication to date.

We then present XCLAIM, the first generic framework for transferring assets and information across permissionless distributed ledgers without relying on a centralized third party. XCLAIM leverages so-called cryptocurrency-backed assets, blockchain-based assets one-to-one backed by other cryptocurrencies, such as Bitcoin-backed tokens on Ethereum. Through the secure issuance, transfer, and redemption of these assets, users can perform cross-chain exchanges in a financially trustless and non-interactive manner, overcoming the limitations of existing solutions. To ensure the security of user funds, XCLAIM relies on collateralization of intermediaries and a proof-or-punishment approach, enforced via smart contracts equipped with cross-chain light clients, so-called chain relays. XCLAIM has been adopted in practice, among others by the Polkadot blockchain, as a bridge to Bitcoin and other cryptocurrencies.

Finally, we contribute to advancing the state of the art in cross-chain light clients. We develop TxChain, a novel mechanism to significantly reduce storage and bandwidth costs of modern blockchain light clients using contingent transaction aggregation, and apply our scheme to Bitcoin and Ethereum individually, as well as in the cross-chain setting.

Copyright

Statement of Originality

I declare that this thesis was composed by myself, and that the work that it presents is my own except where otherwise stated.

Acknowledgements

I would like to thank

- My supervisor, Professor William J. Knottenbelt for his guidance and support throughout this PhD. Under his supervision I had the freedom to explore as many research topics as I could think of, while always being able to ask for help and feedback.
- Blockchain.com for funding my research and offering a glimpse into the industrial side of the blockchain space.
- My second supervisor, Arthur Gervais for advising me on how to advance my research career. Professor Edgar Weippl for his trust, support and enabling me to collaborate with the great research group at SBA Research.
- My collaborators Mustafa Al-Bassam, Zeta Avarikioti, Ittay Eyal, Guillaume Felley, Peter Gaži, Arthur Gervais, Lewis Gudgeon, Bernhard Haselhofer, Dominik Harz, Dragos Illie, Aljosha Judmayer, Andreas Kern, Aggelos Kiayias, Rami Khalil, William J. Knottenbelt, Elefterios Kokoris-Kogias, Joshua Lind, Sarah Meiklejohn, Pedro Moreno-Sanchez, Daniel Perez, Matteo Rommiti, Itay Tsabary, Phillip Schindler, Iain Stewart, Nicholas Stifter, Edgar Weippl, Sam Werner, Katinka Wolter, and Dionysis Zindros,
- ACE358 and the great people I got to know including Daniel, Dimitrios, Dominik, Dragos, João, Katerina, Lewis, Paul, Rami, Sam, Sirvan, and Toshiko.
- Daniel for being a great friend, collaborator and ever so reliable companion at the pub - and for spontaneously joining me on a trip to Japan making it one of my most memorable journeys, not least due to fleeing a typhoon on the last train from Osaka, first class and supplied with plenty of beers. My friend Sam, for all the fun memories and the many conference trips worth remembering. Paul, for keeping up class during all the nights out in London. My dear friend Zeta, for our amazing conference trips to Japan, Spain, Switzerland, USA and the Caribbean, and enduring all the nonsense I conjured upon us.
- My trusted friend, collaborator, and co-founder Dominik Harz. In good times as in bad, I could not have wished for a better companion. May our journey at Interlay be a success worth remembering.

- My parents, Alla and Sergei, for their never ending support, trust and encouragement. It is without doubt that I owe this PhD to you. My brother, Anton, for being thoughtful, caring and reminding me not to sacrifice health for work. My sister, Alina, for always being joyful and making me smile even in the saddest of moods. My sister Natasha and my nephews and nieces, always remembering and welcoming despite the distance. My grandparents, inspirational, encouraging, and always reminding me to stay focused on completing this PhD. My friends Andi, Johann, Max, Raphael, Tino, Vincent, Weissi, and the entire Viennese crew who made Vienna feel like home, no matter how long I was abroad. And last, Sandra, my partner in crime, for her loving care, patience, encouragement, and being there whenever I felt I could not go any further.

Dedication

To my grandfather J. F. for inspiring me to always explore beyond what we know today. You taught me that an idea alone won't change the world. Not unless it is written down and explained in simple terms so it may stand the test of time and scrutiny of other, perhaps brighter, minds. I wish you had lived to witness the conclusion of my academic journey.

May you rest in peace.

‘It is a magnificent feeling to recognize the unity of complex phenomena which appear to be things quite apart from the direct visible truth.’

Albert Einstein

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In 2008, out of the tumults of the collapse of the US housing market and the resulting economic crisis emerged Bitcoin, a decentralized and trustless financial ledger secured by the computational power of thousands around the globe. Created by the, to this date, unknown Satoshi Nakamoto, Bitcoin promised financial freedom for all - no central single controls the ledger, and anyone can join or leave the system whenever they wish. Yet Bitcoin was designed 'merely' as a currency, and soon the value of the underlying technology - *blockchains* - was discovered as a tool to bring decentralization and censorship resistance to numerous applications, starting with identity systems, all the way to universal computations and 'programmable money'.

As the first 'alternative' distributed ledgers emerged, so did the first mechanisms to communicate between them. The first instance of cross-chain communication was deployed as early as 2011. Led by Satoshi Nakamoto him-/herself, Namecoin, the first alternative cryptocurrency, implemented a mechanism to re-use the computational power securing Bitcoin for its own consensus.

Communicating across distributed ledgers, however, proved to be a challenging problem. By design, blockchains are secure within themselves, ensuring that the history of events is im-

mutable. Processing and correctly reacting to information from external sources, however, was never envisioned in the design of Bitcoin, which lacked the functionality to implement anything more complex than payments. While the early Bitcoin community was working on decentralized communication concepts, the rapid growth of the cryptocurrency market demanded fast and simple solutions. Centralized exchanges quickly emerged as the preferred route to exchange cryptocurrency assets, despite requiring trust and lacking transparency. Despite millions of dollars being lost in the infamous collapse of the Mt. Gox exchange (and the following failures of numerous other centralized platforms), the emergence of flexible and expressive systems like Ethereum, and the global regulatory offensives against digital assets service providers - centralized exchanged continued to dominate the cross-chain market.

In 2018 decentralized financial applications, spearheaded by the exponentially growing Ethereum ecosystem, finally began to challenge the status quo. Over the next few years, billions of dollars would be traded through decentralized exchanges on Ethereum, some of which outgrowing their centralized counterparts. However, these applications are mainly available to Ethereum's native assets. Other networks must either copy applications or, if the functionality is not natively supported as in the case of Bitcoin, migrate assets onto Ethereum. In the former case, smaller networks will often attempt to attract liquidity from larger networks by offering easy-to-use cross-chain migration services.

This new, growing demand for secure cross-chain communication sparked a new wave of research. Numerous academic papers, blog posts, standards, prototypes, even entire blockchain networks, have emerged in the hunt for the holy grail of blockchain interoperability: a truly trustless and decentralized cross-chain communication protocol. Comparable to a gold rush, interoperability start-ups raised millions of dollars promising to solve this hard problem, hurrying to launch and competing for the market. The result reminds, of the dot-com bubble: dozens of products promising decentralization, yet under the hood relying on the same centralized providers they hoped to replace. And so, communication across blockchains remains in the hands of a few powerful institutions.

1.2 Contributions

This thesis explores the problem of trustless blockchain interoperability from both the theoretical and practical perspectives, defining the underlying research problem, discovering its theoretical impossibility, and proposing novel cross-chain asset transfer mechanisms as viable solutions for interoperability in practice.

More specifically, the contributions of this thesis are the following:

- We define the problem of Correct Cross-Chain Communication (CCC) under the distributed ledger model extended from the Bitcoin backbone protocol [GKL16], analyze its properties, and derive the step-by-step execution for a generic cross-chain protocol, applicable to all interoperability approaches created to date. We identify parallels to the fair exchange protocols and consequently derive an impossibility result for trustless cross-chain communication by reduction to the Fair Exchange problem [ASW98a, PG99], negating common assumptions about interoperability within the blockchain community.
- With the impossibility result in mind, we introduce the Cross-Chain Design framework as a tool to create new and evaluate existing cross-chain protocols based on security and trust assumptions. Specifically, we identify the main challenge of cross-chain communication as selecting the most suitable trusted third party (TTP) model at each step of the protocol based on the use case and system model, and enable protocol designers to pick and choose from existing implementation techniques.
- We introduce XCLAIM, a first-of-its-kind *financially trustless* mechanism to move assets across blockchains (e.g. Bitcoin to Ethereum), leveraging incentives and punishment in alignment with rational exchange protocols [Syv98] to work around the theoretical impossibility result in practice. We define the concept of cryptocurrency-backed assets and use this mechanism to create e.g. 1:1 Bitcoin-backed assets on Ethereum, and guarantee that users can always redeem the backed assets for the underlying asset ("physically") or be reimbursed in a collateral currency at a beneficial rate ("cash" redemption). XCLAIM makes use of a fully permissionless set of collateralized intermediaries, who hold backing

assets in custody and must proactively prove correct behavior via a cross-chain light client (chain relay) on the target/issuing blockchain. We introduce a series of protocols to balance the ratio of collateral and issued cryptocurrency-backed assets, even in case of attempted theft or significant exchange rate fluctuations. XCLAIM allows transferring assets from all existing blockchains to systems with smart contract support. As a result, XCLAIM has been adopted by major blockchains like Polkadot [Woo15] as a mechanism to import assets from "traditional" cryptocurrencies like Bitcoin, enabling a wide range of novel, decentralized financial applications.

- Finally, we address the issue of efficient cross-chain state verification. We introduce TxChain, a novel mechanism to batch a large number of transaction inclusion proofs into a single on-chain transaction, contingent on the existence and validity of the batched transactions. We show how TxChain can be used to increase the efficiency of blockchain light clients, improving in particular upon new techniques such as NiPoPoWs [KMZ17] and FlyClient [BBB+17]. TxChain can be deployed on Bitcoin with and without protocol changes, as a hard fork to Ethereum, and on top of chain relays e.g. to improve verification of Bitcoin transactions on Ethereum.

1.3 Statement of Originality

I declare that this thesis was composed by myself and that the work that it presents is my own except where otherwise stated.

1.4 Publications

Most of the work presented in this thesis is based on the following publications:

- SoK: Communication Across Distributed Ledgers. **Alexei Zamyatin**, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias,

and William J. Knottenbelt *25th International Conference on Financial Cryptography and Data Security*, 2021 (Chapters 3 and 5).

- XCLAIM: Trustless, Interoperable, Cryptocurrency-backed Assets. **Alexei Zamyatin**, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. *IEEE Symposium on Security and Privacy*, 2019 (Chapters 4 and 5).

- TxChain: Efficient Cryptocurrency Light Clients via Contingent Transaction Aggregation. **Alexei Zamyatin**, Zeta Avarikioti, Daniel Perez, William J. Knottenbelt. *International Workshop on Cryptocurrencies and Blockchain Technology*, 2020 (Chapter 5).

The following works, not included in this thesis, arose from work conducted during the course of this PhD:

- XCC: Theft-Resilient and Collateral-Optimized Cryptocurrency-Backed Assets. Theodore Bugnet and **Alexei Zamyatin**. *Technical Report. Online*, 2022

- On the Deployment of FlyClient as a Velvet Fork: Chain-Sewing Attacks and Countermeasures. Tristan Nemoz and **Alexei Zamyatin**. *Cryptology ePrint Archive: Report 2021/782*, 2021

- Commit-Chains: Secure, Scalable Off-Chain Payments. Rami Khalil, **Alexei Zamyatin**, Guillaume Felley, Pedro Moreno-Sanchez, Arthur Gervais. *Cryptology ePrint Archive: Report 2018/642*, 2019

- Pay-To-Win: Cheap, Crowdfundable, Cross-chain Incentive Manipulation Attacks on Cryptocurrencies. Aljosha Judmayer, Nicholas Stifter, **Alexei Zamyatin**, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar Weippl. *Workshop on Trusted Smart Contracts, Financial Cryptography and Data Security*, Online, 2019

- SoK: Algorithmic Incentive Manipulation Attacks on Permissionless PoW Cryptocurrencies. Aljosha Judmayer, Nicholas Stifter, **Alexei Zamyatin**, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar Weippl. *Workshop on Trusted Smart Contracts, Financial Cryptography and Data Security*, 2021

- A Deep Dive into Bitcoin Mining Pools: An Empirical Analysis of Mining Shares. Matteo Rommitti, Aljosha Judmayer, **Alexei Zamyatin**, and Bernhard Haselhofer. *Workshop on the Economics of Information Security.* 2019

- Uncle Traps: Harvesting Rewards in a Queue-based Ethereum Mining Pool. Sam Werner, Paul Pritz, **Alexei Zamyatin**, William J. Knottenbelt. *EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, 2019.

- Echoes of the Past: Recovering Blockchain Metrics From Merged Mining. Nicholas Stifter, Philipp Schindler, Aljosha Judmayer, **Alexei Zamyatin**, Andreas Kern, and Edgar Weippl. *International Conference on Financial Cryptography and Data Security*, 2019.

- Multisignatures for Cryptocurrency-Backed Tokens (Poster). **Alexei Zamyatin**, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais and William J. Knottenbelt. *International Workshop on Cryptocurrencies and Blockchain Technology*, 2018.

- Committing to Quantum Resistance: A Slow Defence for Bitcoin Against a Fast Quantum Computing Attack. Iain Stewart, Dragos Ilie, **Alexei Zamyatin**, Sam Werner, Maryam F. Torshizi, and William J. Knottenbelt. *Royal Society open science, 5(6), 180410*, 2018.

- Flux: Revisiting Near Blocks for Proof-of-Work Blockchains. **Alexei Zamyatin**, Nicholas Stifter, Philipp Schindler, Edgar Weippl, and William J. Knottenbelt. *Cryptology ePrint Archive: Report 2018/415*, 2018.

- Agreement with Satoshi— On the Formalization of Nakamoto Consensus. Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, **Alexei Zamyatin**, and Edgar Weippl. *Cryptology ePrint Archive: Report 2018/400*, 2018.

- A wild velvet fork appears! Inclusive blockchain protocol changes in practice (Short Paper). **Alexei Zamyatin**, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, and William J. Knottenbelt. *Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security*, 2018.

Chapter 2

Background and Related Work

In the following we provide the necessary background theory on blockchain-based distributed ledgers, using Bitcoin as an example and highlighting alternative designs where relevant.

2.1 Fundamentals: Bitcoin, Blockchain, and Consensus

Bitcoin was introduced in a whitepaper released by the pseudonymous Satoshi Nakamoto in 2008 [Nak08], laying the groundwork for an entirely new field of research and industry.

Bitcoin's innovation was the combination of a fully permissionless distributed system with incentives created by an underlying digital currency: any consensus participant can join or leave the system at any point in time, while correct behavior is rewarded with newly minted units of fungible digital coins. Permissionless in this context means anyone can join or leave the consensus protocol without requesting permission from any third party

2.1.1 Transactions and Blocks

A fundamental data structure underpinning many cryptocurrencies is the *blockchain* - an append-only immutable record of digitally-signed *transactions*. Transactions encapsulate changes

to the state of the distributed ledger, most frequently representing transfers of some token of value. Each transaction is identified through a unique hash over the transaction data structure containing the details of the state update. To determine the state of the ledger at a particular point in time, transactions are consolidated into *blocks* - hundreds or even thousands at a time. Each block is identified by a unique hash of all consolidated transactions (or transaction identifiers) and the block header, which among other information contains the hash of the previous block. Referencing the hash of the predecessor in each block effectively *chains* blocks together. Assuming the use of a cryptographically secure hash function, this structure makes it impossible to change the content of any block without updating all successors, if the chain of blocks is to be preserved.

Participants of this network run nodes which store the entire history of blocks and communicate in a peer-to-peer fashion using a gossip protocol. Blocks are broadcast, verified by recipients and, if valid under the network's consensus rules, propagated to other nodes.

2.1.2 Proof-of-Work and Nakamoto Consensus

Bitcoin has no single point of control. Since anyone can join the system and generate transactions, a mechanism to agree on the state of the ledger at any point in time is required. To this end, Bitcoin employs a distributed consensus mechanism to agree on the latest modifications to the ledger state, i.e., the transactions contained in the latest block appended to the blockchain. The mechanism, used in Bitcoin and most other cryptocurrencies, is termed *Nakamoto consensus* – a random leader election protocol that requires nodes to compete in solving a hard cryptographic puzzle, known as Proof-of-Work (PoW). Each time a node solves the puzzle it is elected leader and may determine which transactions are included in the next block appended to the blockchain.

More detailed, this puzzle involves guessing a hash over the content of the to-be-included block and, by design, there exists no better strategy than enumerating all possible candidates. While computing the hash of some data is trivial, PoW cryptocurrencies require that this hash fulfills some criteria, e.g. contains a number of leading zeroes as in the case of Bitcoin. These criteria

can be adjusted to increase or decrease the difficulty of the puzzle. The more difficult the puzzle, the longer it will take the network to find a solution candidate and generate a new block. The verification of a potential solution candidate, on the other hand, is trivial.

The more computational power a node invests in this so-called 'mining' process, the higher the chance it will be the first to find a suitable solution – and as a result, the higher its voting power in the network. Drawing analogies to mineral mining, participants are often referred to as *miners* and their computational power is termed *hash rate*. As more miners join the network, the overall computational power increases and so does the block generation rate. To ensure the election process is somewhat predictable and to prevent the system for being overwhelmed by too many blocks, the *difficulty* of the PoW puzzle is dynamically adjusted. In the case of Bitcoin the difficulty, i.e., the required number of preceding zeroes, is adjusted approximately every two weeks such that blocks are generated on average every ten minutes. If a miner finds a solution that has a lower difficulty than required, the block is not considered valid by the network.

Proof-of-Work, combined with the immutable nature of the underlying blockchain data structure, achieves two important properties, necessary for achieving a consistent view of the distributed ledger across all participants:

- First, it allows Bitcoin to operate in a permissionless setting. While anyone can join the network at any time and create identities / run multiple consensus nodes, their voting power is capped by the computational resources they allocate to the mining process – yielding the system resilient to so-called *Sybil* attacks [Dou02].

- Second, it guarantees a total ordering of all transactions, mitigating the *double spending problem*: if two transactions spend the same coin, only the first transaction to be included in the blockchain will be considered valid.

2.1.3 Longest Chain Rule and Forks

While miners determine the state of the global transaction ledger, they cannot simply include invalid transactions (or attempt to alter the ledger history) in a block when they are elected leaders. This would fail to pass the verification performed by other nodes in the network and result in an invalid block, which would hence be rejected, i.e., not propagated to other nodes in the network. The result is a so-called *fork*: the miner in question creates her own version of the chain, while the rest of the network follows another.

However, forks can occur even if all involved miners are honest: since there can be hundreds or thousands of miners competing to become leaders, sometimes two or more PoW solutions can be found at or around the same time. As a result, multiple valid but conflicting blocks (reference the same predecessor) are propagated to the network, creating forks. To handle such disputes, Bitcoin's code dictates that nodes follow the chain with the most Proof-of-Work, meaning the chain which is approved and "mined" upon by the majority of the network. The approval of each chain is measured by the cumulative sum of the PoW difficulty of included blocks. Considering that the PoW difficulty is adjusted infrequently, this is often referred to as the "longest chain rule". As a result, the "main" chain is defined as the chain which is agreed upon by the majority of computational power, and honest network participants will follow and extend this main chain.

From the perspective of distributed systems theory, Nakamoto consensus falls into the class of *stabilizing consensus* [AFJ06] protocols or protocols that reach *eventual agreement* [Fis83]: all correct processes (nodes) reach agreement not in the same communication round but only eventually, after a number of rounds. Applying this to the model of Bitcoin, a transaction is considered *stable* if a number of blocks have been mined on top of the block it is contained in [Nak08, GKL15, GKL16, PSs16]: the deeper a block (transaction) is included in the blockchain, the less likely it is to be excluded from the chain by a fork. The number of the necessary block "*confirmations*" to achieve sufficient confidence about a transaction's stability is a matter of dispute: while numerous community discussions suggest 6 confirmations in Bitcoin, research argues that this number must be based on the economic value of the transaction

in question [SZ16].

Agreement on Network Upgrades

Forks also occur whenever the network's protocol rules are upgraded, e.g. when protocol improvements or consensus-relevant bug-fixes are introduced. We can thereby differentiate between two main fork scenarios:

- *Soft forks*, where consensus rules are made more strict, i.e., upgraded miners accept fewer blocks than non-upgraded (or *legacy*) miners. As long as the computational power of the upgraded miners exceeds that of the non-upgraded miners, the network will converge to a single, longest chain.
- *Hard forks* occur when consensus rules are expanded, i.e., upgraded miners accept blocks that are rejected by legacy miners, or conflicting rules are introduced. As a result, two incompatible versions of the chain are created. The outcomes of hard forks vary: sometimes, new cryptocurrencies are created, as in the case of Bitcoin and Bitcoin Cash; in other cases, only one chain gains sufficient traction, forcing the "losing" miners to comply with the network majority. It is also worth noting that a soft work that fails to gain the support of the majority of computational power can also lead to a hard fork, i.e., if legacy miners mine significantly more blocks than upgraded miners.
- *Velvet forks*, first described in [KMZ17], are a mechanism to extend the functionality of existing blockchains without causing a consensus split. Upgraded miners accept both upgraded and legacy blocks, while the upgraded functionality is encoded such that legacy miners ignore it during the verification process, interpreting the additional information as arbitrary data. However, velvet forks are only safely applicable to upgrades that do not require the support of the consensus majority, limiting their usability in practice.

There exist other, more sophisticated mechanisms for network upgrades and we direct the interested reader to our work on *velvet forks*, conducted in the course of this PhD [ZSJ+18].

2.1.4 Incentives and Rewards

As a reward for investing computational effort, the winning miner is rewarded with newly minted coins of the underlying cryptocurrency each time a block is generated. Furthermore, transactions include a fee that serves as incentive for miners to include them in blocks, paid by users. This ensures miners are motivated to behave correctly and follow the longest chain, as creating a contentions blockchain fork bears the risk of losing out on rewards if the attack is unsuccessful.

Bitcoin's code specifies that there shall only exist 21 million Bitcoins, with an emission schedule that halves the newly minted coins per block approximately every four years. As a result, the block rewards will cease at some point in the future, leaving the transaction fees as the sole incentive mechanism for miners. In light of security concerns raised by research [CKWN16], other cryptocurrencies follow alternative approaches, e.g. unlimited supply and continuous block rewards.

Depending on the network usage, transaction fees may outperform the block rewards and hence play an important role in sustaining incentives for miners. Thereby, most blockchains maintain a free fee market: each block can contain a maximum number of transactions (due to network security and latency reasons), meaning that in times of high network utilization, some users may have to wait for multiple blocks until their transaction is included in the blockchain. Thereby, users can increase their fee to "bribe" miners to include their transactions earlier than that of others. The result is an auction for "block space". This self-regulating market is thereby both cure and curse. On one hand, miners and users always find an equilibrium in terms of demand and fee rates. On the other hand, in times of high network utilization, fees have been shown to spike so high, that some users are unable to compete – leaving large numbers of transactions unprocessed and creating a large processing backlog, effectively "congesting" the network.

2.1.5 Peer-to-Peer Network

Nodes in Bitcoin's peer-to-peer network are globally distributed, each node (ideally) randomly connecting to a set of "neighbors". Each time a node receives a transaction or block, it verifies it against the network's consensus rules and, if valid, forwards it to its neighbor nodes. As a result, different nodes in different locations in the network topology may temporarily have different views of the blockchain. Eventually, however, all nodes converge to the same ledger state, enforced by the longest chain rule discussed earlier.

An important assumption of Bitcoin is that the network is *synchronous*, i.e., messages are propagated among honest nodes within a known delay [CDE+16, GKL15, PSs16]. Ensuring timely propagation of information across the network is one of the reasons for limiting the size of blocks: while larger blocks allow to *stabilize* more transactions per time unit, they also incur a higher propagation delay due to network latency. Optimization of blockchain scalability while maintaining security has been an active research topic ever since the early adoption of Bitcoin [CDE+16, GMSR+19, GKW+16, KKJG+18, PD16, Chr15].

Full Nodes and Light Clients

By default, nodes download and store the entire history of transactions since the genesis block. However, as time progresses and the network keeps growing, the storage and bandwidth requirements for running a node may become too high for some use cases. The entire Bitcoin blockchain, for example, amounts to 344 GB of data, while downloading the entire state of Ethereum requires up to 5 TB of storage space. While feasible on most modern PC, this becomes a particular problem for nodes hosted on mobile, wearable, or IOT devices – which are arguably necessary for better user experience.

To tackle this challenge, the original Bitcoin whitepaper introduced the *Simplified Payment Verification* (SPV) or *light client* protocol – a scheme which allows nodes to download only a few bytes of data per block (the *block header*), skipping most transnational data, while still being able to verify that specific transactions have occurred. Specifically, a block header consists

of the root hash of the Merkle tree containing the identifiers of all transactions included in the block, the previous block hash, and other metadata relevant for PoW verification. To convince a light client that a transaction has been included in a specific block one must only provide the path to that transaction's identifier in the block's transaction Merkle tree – a list of hashes, logarithmic in size given the number of all transactions contained in the block. As a result, Bitcoin's SPV clients today require merely 70 MB of storage space. In the case of Ethereum, this number lies higher, at around 5 GB.

The security model of light clients, however, differs slightly from that of full nodes. By design, light clients can:

- Given two chains, detect which is the chain with the most accumulated PoW/PoS, i.e., the *main chain*;
- Given a transaction identifier, verify that this identifier is indeed part of a block's transaction Merkle tree.

However, a light client *cannot* check whether a transaction is fully *valid* under the network's consensus rule, as this requires access to the entire transactional history. As a result, light clients operate under the assumption that if a block has been included in the main chain, then it must hence be valid under the system's consensus rules – otherwise, the honest majority of network participants would have rejected a particular block. We provide a visualization of the interaction between full nodes and light clients in Figure 2.1 and point the interested reader to Appendix A for a in-depth analysis and classification blockchain state verification and light clients.

Recently, two improved light client protocols have been introduced, aiming to further reduce the storage and bandwidth requirements of blockchain nodes. FlyClient [BKLZ20] and Superblock NiPoPoWs [KMZ17] require to download and store only a (poly-)logarithmic number of block headers. Both protocols rely on probabilistic sampling techniques: instead of requesting all block headers, such light clients randomly sample a logarithmic subset. By ensuring that the sampled set is indeed random, these protocols ensure that a malicious full node cannot

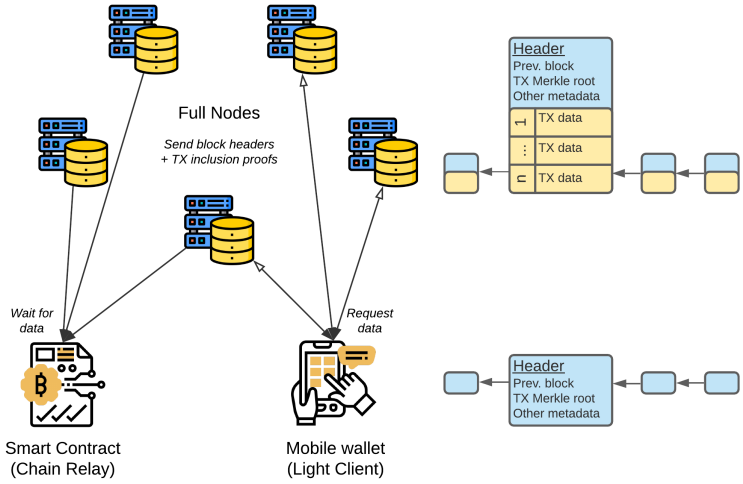


Figure 2.1: Full nodes store the entire blockchain, including all transaction data. Light clients (e.g. in mobile wallets) request and store only block headers and inclusions proofs for relevant transactions. Chain relays, light clients encoded as smart contracts on top of blockchain networks, await to receive and process data.

trick the light client into accepting an incorrect state of the ledger, as long as the majority of computational power is honest.

We take a particular interest in blockchain light clients in this thesis. While initially designed to target mobile devices, light clients play an important role in cross-chain communication, where being able to verify the state of another chain is a useful property utilized by many interoperability protocols. We formalize the requirements for Proof-of-Work cross-chain light clients, so-called *chain relays* [But16, Con17a, ZHL+19], in Appendix B and present novel efficiency improvements in Chapter 5.

2.1.6 Alternative Consensus Mechanisms

Nakamoto consensus is just but one possible option to achieve agreement on the state of the distributed ledger. There exist numerous BFT agreement protocols that have been introduced and tested before Bitcoin was launched and are applicable to blockchain networks [SJS+18, BSAB+17, GK18] – yet BFT protocols typically operate over a predefined set of consensus participants. Such protocols are termed *permissioned* implying that new consensus participants

must first obtain the permission of the existing consensus committee before being able to partake in voting.

Another, *permissionless* way of achieving consensus in blockchains is Proof-of-Stake (PoS) [KRDO17, Mic16, DGKR18, PS16]. Instead of relying on computational power as a scarce resource to calculate the voting power of each participant, Proof-of-Stake blockchains rely on – as the name suggests – stake. Stake is defined as the amount of the cryptocurrency underlying the system that a consensus participant puts down to insure the network against their misbehavior. Should a consensus participant attempt a failed attack on the system, the honest majority of the network will confiscate (or *slash*) the malicious node's stake. The more stake a participant locks, the more voting power they exhibit. Proof-of-Stake systems typically use a source of randomness (e.g. verifiable random functions [MRV99, DY05]) to sample temporary consensus committees for predefined intervals, which in turn run a variant of a BFT agreement protocol.

2.1.7 Blockchain Application Layer

The sophisticated nuances of the network and consensus layer often remain hidden from users, as they interact mainly with applications built on top of blockchains.

Accounts

The first interaction of a user with Bitcoin is the generation of an account – a public/private key pair which is used to send, receive and authenticate transactions, most often payments. However, instead of using public keys to receive payments, Bitcoin generally recommends the use of so-called *addresses* – hash-based, alpha-numeric representations of the ECDSA secp256k1 public key [Bit20d].

We differentiate between two main types of accounting systems in blockchains:

- *UTXO (Unspent Transaction Output)* model. In the UTXO model, each transaction

consists of inputs and outputs, whereby each input is the output of another, existing transaction. Each output consists of a number of coins and the rules of how these coins can be spent. Thereby, outputs must always be spent as a whole. The amount of coins a user controls is hence given by the sum of the value of all outputs this user can spend from, e.g. by providing a digital signature using her private key.

- *Account* model. In the account-based model, e.g. as seen in Ethereum [Woo17, But14], each user controls one or more accounts, which stores the number of coins owned by the user – similar to the models used by traditional banks. The amount of sent/received coins in a transaction is then simply subtracted/added to the account's balance.

Smart Contracts

Transactions update the state of the distributed ledger. The simplest form of state update is typically a payment: a transfer of ownership of coins from one user account to another. Most cryptocurrencies, including Bitcoin, support the creation of *conditional* payments, i.e., allow to specify conditions which must be met before a transaction is included in the blockchain. These conditions are encoded in programs which are also referred to as *smart contracts* since their execution is enforced by the entirety of the network's consensus.

Smart contracts can take various forms. Systems like Bitcoin exhibit very limited scripting functionality, allowing users to construct simple commit-reveal schemes at most. Newer systems, such as Ethereum [But14, Woo17] or Polkadot [Woo15] provide near-Turing complete scripting functionality, enabling users to create more complex contracts and even reflect real-world financial agreements.

Tokens, Colored Coins, and Overlay Protocols

In the majority of cases, blockchain-based distributed ledgers introduce their own, native digital currency as part of the incentive mechanisms, including Bitcoin and Ethereum. However, deploying a new distributed ledger is not ultimately necessary to bootstrap a new cryptocurrency:

using the scripting support of distributed ledgers, it is possible to simulate the functionality of a digital currency on top of an existing system. The most prominent example are the fungible ERC-20 tokens on Ethereum [VB15], which are encoded via standardized smart contracts and are supported by essentially all Ethereum applications.

While tokens are mainly observed on blockchains with smart contract functionality, they have been first introduced as so-called *overlay protocols* for Bitcoin [ZSJ+18]: additional data was included in transactions and interpreted by applications, e.g. to replicate the functionality of tokens [Ros12].

2.2 Related Work

2.2.1 Formalization of Cross-Chain Communication

To the best of our knowledge, this work constitutes the first formalization of cross-chain communication and the derived impossibility result negates common assumptions about interoperability of previous works. Existing surveys on blockchain interoperability mainly provide iterative summaries of interoperability implementations, or focus on subsets of this space, supporting our study.

The first work discussing cross-chain communication, excluding forum discussions, is a technical report by Back et al. [BCD+14]. The authors introduce the term "sidechain" and present how assets can be transferred between two chains using a committee of custodians or SPV proofs in a homogeneous security model. A later report by Buterin discusses how cross-chain exchanges can be achieved via custodians, escrows, HTLCs, and cross-chain state verification, and provides a high-level discussion of possible failures in cross-chain communication [But16]. Belchior et al. [BVGC21] provide another, more recent, iterative overview of cross-chain projects, discussing numerous community-driven projects in-depth, yet without clear taxonomy or classification.

Siris et al. [SDF+19] provide an iterative overview of protocols for atomic cross-chain swaps and "sidechains", focusing mostly on community-driven efforts, rather than academic pub-

lications. Similarly, Johnson et al. discuss open-source interoperability projects related to Ethereum [JRB19], while Robinson evaluates Ethereum as a coordination platform for communication among other blockchains [Rob20]. Bennik et al. [BGDE18] and similarly Miraz et al. [MD19] summarize technical details of HTLC atomic cross-chain swaps. Avarikioti et al. provide a thorough formal study of blockchain sharding protocols, although their focus does not lie on the communication between shards [AKKW19].

Other works [XWS+17, TT17, CZDK17, ZJ18, VTPM18], mention the importance of blockchain interoperability, e.g. for bootstrapping new blockchains, but do not provide classifications or technical details.

2.2.2 Interoperability via Cryptocurrency-Backed Assets

Until 2019, the only mechanism believed to perform a trustless cross-chain transfer was *atomic cross-chain swaps* (ACCS) based on hashed timelocks [Tie16, Bit21a, Bit20a, Her18]. However, ACCS are *interactive*, i.e., they rely on all parties being online and monitoring the blockchain throughout the exchange to ensure security. Each swap thereby incurs long waiting periods to prevent fraud through exploiting blockchain reorganizations, which substantially hinders performance and involves synchronizing clocks between independent blockchains. Moreover, ACCS are vulnerable to packet and transaction memory-pool sniffing, allowing an adversary to exploit blockchain race conditions to steal funds. Finally, ACCS rely on a pre-established out-of-band communication channel between parties, required to exchange security-critical *revocation transactions* [BJZ+17, TS15]. Xclaim not only avoids these problems, as discussed throughout this work but is also more efficient in terms of execution costs and time.

The deployment of asset migration protocols opened a new way to exchange assets across blockchains: cryptocurrency-backed assets (CbAs). However, most existing approaches towards CbA systems require trust in intermediaries.

The most widely adopted CbA protocol today is wBTC ("wrapped Bitcoin") [Kyb19b], a fully custodial and centralized version of Bitcoin on Ethereum. Users entrust institutional service

providers with custody over their BTC and are required to undergo KYC procedures before interacting with the system. Similar approaches have been copied by prominent cryptocurrency exchanges such as Binance [Bin19] and Huobi [Tea20]. RSK [Ler15] relies on merged-mining with Bitcoin [JZS+17] and aims to leverage Bitcoin miners for moving BTC to its ledger - yet, as of this writing, relies on an independent, centralized set of custodians. Summarizing, these systems utilize External Custodians (cf. CCC classification in Section 3.3) and resemble centralized systems, upon which XCLAIM introduces significant improvements in terms of security and decentralization (cf. Section 4.2).

Liquid [DPW+16] is a permissioned blockchain that uses Bitcoin as its native asset via CBAs. Thereby, Liquid uses its consensus participants as custodians to hold migrated BTC using multisignatures (i.e., Consensus Custodians, cf.3.3). While arguably more robust to custodians external to the involved blockchains, this approach still exhibits trust and centralization issues. Since users rely on the custodians to not steal their BTC, they have no remedy in case of theft, and cannot become custodians themselves. Numerous other approaches have implemented similar mechanisms, including Proof-of-Stake sidechains [GKZ19a] and PoA Network[PoA18].

In the tBTC protocol, currently deployed between Bitcoin and Ethereum, External Custodians construct a jointly controlled deposit public key on Bitcoin via ECDSA threshold signatures [GGN16], and insure users against theft by locking up ETH collateral - following a similar approach to XCLAIM. However, at the time of writing, the implemented threshold signature scheme does not support fault attribution, i.e., it is impossible to distinguish between honest and malicious members of the threshold signature committee in case of failures. As a result, if an adversary is able to subvert a signing committee and commit theft by controlling the majority of signers, tBTC seizes the collateral of *all* signing committee members to reimburse users, even those who behaved honestly. To this end, tBTC, at the time of writing, utilizes a static and restricted committee of signers, failing to achieve the decentralization (and censorship-resistance) properties exhibited by XCLAIM. RenVM [Ren20] follows a similar design as tBTC, yet aims to replace threshold signatures with distributed key generation via secure multi-party computation [Gol98]. However, to the best of our knowledge, RenVM utilizes centralized custodians [Beh20] at the time of writing.

Bentov et al. describe how to tokenize exiting cryptocurrencies via trusted execution environments (TEEs) [BJZ+17]. TEEs, however, are known to be vulnerable to a wide range of side-channel attacks [XCP15, WKPK16, GESM17] and require trust in the hardware manufacturer. As a standalone solution, TEEs expose users to the same trust assumptions as CentralClaim. TEEs, however, can be used as an additional layer of security of Xclaim Vaults, improving the robustness of the implementation.

Dogecoin [TSB19], published online subsequently to Xclaim, describes a CbA protocol following the Xclaim design, yet under the assumption of constant exchange rates. As a result, Dogecoin would have to adopt similar collateral balancing mechanisms as Xclaim to maintain the security of user funds under the fluctuating exchange rates of cryptocurrency assets observed in practice.

Finally, *synthetic assets* are an alternative solution to cross-chain asset migration, relying purely on collateralization. Synthetix [Syn20] allows users to create assets pegged to the price of existing, backing assets, such as BTC, by locking a significantly higher (500%) amount of collateral using the Synthetix native token. The resulting sBTC token exhibits similar properties to Xclaim CbAs in terms of decentralization since any user can become their own "Vault" by contributing collateral to the Synthetix smart contract. The main difference between physically pegged Xclaim CbAs and synthetic assets is that CbAs can be "physically" redeemed for the backing asset and "cash" redemption through collateral is used only in case of failure, while synthetic assets can only be "cash" redeemed. This difference can be of relevance in catastrophic failure scenarios, e.g. mass liquidation and user exits in case of severe exchange rate fluctuations. Nevertheless, the similarities in terms of game-theoretic security and incentives suggest synergies between Xclaim and purely synthetic mechanisms and represent an interesting avenue for future research.

2.2.3 Chain Relays and Light Clients

Most existing cross-chain light clients, so-called chain relays [Con17a, Kyb17, Con17b], are implemented in accordance with the design of naive SPV clients, storing all block headers of

the verified blockchain, which can incur substantial cost on the blockchain they are deployed on. In contrast to SPV clients implemented as blockchain nodes, data stored in chain relays, which are typically implemented as smart contracts on top of another blockchain, cannot be easily removed. Furthermore, while bandwidth is typically assigned no financial cost when evaluating SPV clients, bandwidth is charged by the byte in smart contract environments such as Ethereum.

New proposals for efficient light clients, leveraging concepts such as NiPoPoW [KMZ17, KZ19] or FlyClient [BKLZ20] significantly reduce verification costs as they require to retrieve only a (poly-)logarithmic number of block headers by implementing random sampling heuristics. While the original works focus on single-chain applications, NiPoPoWs have been recently studied in the cross-chain setting. Daveas et al. [DKKZ20] showcase how an efficient Bitcoin light client can be implemented as an Ethereum smart contract, refining the NiPoPoW protocol and suggesting the "hash-and-resubmit" design pattern which greatly reduces processing costs for the Ethereum virtual machine. Interestingly, both NiPoPoWs and FlyClient require soft- or hard forks to be securely deployed on Bitcoin [KPZ20] and similar blockchains, contrary to the assumptions made in the original works. As a result, chain relays to this date implement naive SPV light client techniques.

An alternative approach to random sampling is the use of the compression properties of non-interactive zero-knowledge proof (ZKP) systems such as SNARKs [BCCT12], STARKs [BSBHR18] or Bulletproofs [BBB+18] for light client verification. A concrete technique is zkRelay [WE20a], which makes use of the Zokrates [ET18] framework to generate SNARK proofs for efficient light client verification of Bitcoin.

The verification process of light clients can also be outsourced to users or dedicated operators [TR19, TSB19], with disputes handled via interactive games. However, existing schemes have been shown to currently suffer security challenges [KGC+18].

Finally, deploying chain relays in trusted execution environments (TEEs) (e.g. Intel SGX [Int22]) may present a cheap and scalable approach to cross-chain state verification, at the cost of trusting hardware manufacturers. However, recent vulnerabilities detected in well-known TEE im-

plementations, in particular to side-channel attacks [XCP15, WKPK16, GESM17, VBMW+18], highlight existing security risks and raise questions about current applicability to financial systems. Furthermore, deploying blockchain clients in trusted execution environments may require modifications to the original implementation, increasing the long-term maintenance costs and potentially introducing compatibility issues with protocol upgrades, e.g. hard forks.

Chapter 3

Cross-Chain Communication: Formalization, Impossibility, Analysis

In this chapter, we set out to systematize the field of blockchain interoperability. We propose a formal model of the underlying problem of cross-chain communication, prove its impossibility by reduction from well-known problems in computer science, and construct a comprehensive guide for designing protocols bridging the numerous distributed ledgers available today. The aim of this work is to facilitate clearer communication between academia, community, and industry – enabling users, developers and researchers to gain confidence about the security of both new and existing solutions.

Contribution

In summary, the contribution of this chapter is as follows:

- We formalize the problem of Correct Cross-Chain Communication (CCC) (Section 3.1), tracing CCC back to existing research on distributed systems and outlining a generic CCC protocol encompassing existing solutions. We then relate CCC to the Fair Exchange problem and show that contrary to common beliefs in the blockchain community, CCC is *impossible without a trusted third party* (Section 3.2).

• With the impossibility result in mind, we introduce a framework to design new and evaluate existing CCC protocols, focusing on the inherent trust assumptions thereof (Sections 3.3).

• We apply our framework to classify the field of CCC protocols to date (Section 3.4), highlighting similarities and key differences.

• Finally, we outline general observations on current developments, provide an outlook on the challenges of CCC research, and discuss the implications of interoperability on the security and privacy of blockchains (Section 3.5).

3.1 The Cross-Chain Communication Problem

In this section, we relate cross-chain communication to existing research, introduce the model for interconnected distributed ledgers, provide a formal definition of the Correct Cross-Chain Communication (CCC) problem, and sketch the main phases of a generic CCC protocol.

3.1.1 Historical Background: Distributed Databases

The need for communication among distributed processes is fundamental to any distributed computing algorithm. In databases, to ensure the atomicity of a distributed transaction, an agreement problem must be solved among the set of participating processes. Referred to as the Atomic Commit problem (AC) [BHG87], it requires the processes to agree on a common outcome for the transaction: commit or abort. If there is a strong requirement that every correct process should eventually reach an outcome despite the failure of other processes, the problem is called Non-Blocking Atomic Commit (NB-AC) [BT93]. Solving this problem enables correct processes to relinquish locks without waiting for crashed processes to recover.

As such, we can relate the core ideas of communication across distributed ledgers to NB-AC. The key difference hereby lies within the security model of interconnected systems. While in classic distributed databases all processes are expected to *adhere to protocol rules* and, in the

worst case, may crash, distributed ledgers, where consensus is maintained by a committee, must also consider and handle *byzantine failures*.

3.1.2 Distributed Ledger Model

We use the terms *blockchain* and *distributed ledger* as synonyms and introduce some notation, based on [GKZ19b] with minor alterations. Thereby, it is assumed the majority[1] of consensus participants in both X and Y are honest, namely, that they follow the designated protocol.

Ledgers and State Evolution

We assume the data structure underlying a distributed ledger X is a *blockchain* (or *chain*), i.e., an append-only sequence of blocks, where each block contains a reference to its predecessor(s). The mechanism for maintaining a sequence of transactions is referred to as a *ledger* and we denote the ledger of a system X as L_x. We define the *state* of a ledger L as the dynamically evolving sequence of included transactions $\langle \mathrm{TX}_1, ..., \mathrm{TX}_n \rangle$. We assume that the evolution of the ledger state progresses in discrete *rounds* indexed by natural numbers $r \in \mathbb{N}$. At each round r, a new set of transactions (included in a newly generated block) is written to the ledger L. We use $\mathsf{L}^P[r]$ to denote the state of L at round r, i.e., after applying all transactions *written* to the ledger since round $r-1$, according to the view of some party P. A transaction can be written to L only if it is consistent with the system's consensus rules, given the current ledger state $\mathsf{L}^P[r]$. This consistency is left for the particular system to define, and we describe it as a free predicate $\mathsf{valid}(\cdot)$ and we write $\mathsf{valid}(\mathrm{TX}, \mathsf{L}_x^P[r])$ to denote that TX is valid under the consensus rules of L_x at round r according to the view of party P. To denote that a transaction TX has been *included* in/successfully written to a ledger L as position r, we write $\mathrm{TX} \in \mathsf{L}^P[r]$. While the ordering of transactions in a block is crucial for their validity, for simplicity, we omit the position of transactions in blocks and assume correct ordering implicitly.

[1] In case of Proof-of-Work or Proof-of-Stake blockchains, the majority pertains to computational power [Nak08] or stake [KRDO17] respectively.

Persistence and Liveness

Each participant P adopts and maintains a local ledger state $\mathsf{L}^P[t]$ at time t, i.e., her current view of the ledger. The views of two distinct participants P_1 and P_2 on the same ledger L may differ at time t (e.g., due to network delay): $\mathsf{L}^{P_1}[t] \neq \mathsf{L}^{P_2}[t]$. However, eventually, all honest parties in the ledger will have the same view. This is captured by the persistence and liveness properties of distributed ledgers [GKL16]:

Definition 1 (Persistence)**.** *Consider two honest parties P_1, P_2 of a ledger L and a persistence (or "depth") parameter $k \in \mathbb{N}$. If a transaction* TX *appears in the ledger of party P_1 at time t, then it will eventually appear in the ledger of party P_2 at a time $t' > t$ ("stable" transaction). Concretely, for all honest parties P_1 and P_2, we have that $\forall t \in \mathbb{N} : \forall t' \geq t + k : \mathsf{L}^{P_1}[t] \preccurlyeq \mathsf{L}^{P_2}[t']$, where $\mathsf{L}^{P_1}[t] \preccurlyeq \mathsf{L}^{P_2}[t']$ denotes that L^{P_1} at time t is a (not necessarily proper) prefix of $\mathsf{L}^{P_2}[t']$ at time t'.*

As parties will eventually come to an agreement about the blocks in their ledgers, we use the notation $\mathsf{L}[t]$ to refer to the ledger state at time t shared by all parties; similarly, we use the notation $\mathsf{L}[r]$ for the shared view of all parties at round r. This notation is valid when t is at least k time units in the past.

Definition 2 (Liveness)**.** *Consider an honest party P of a ledger L and a* liveness delay parameter *u. If P attempts to write a transaction* TX *to its ledger at time $t \in \mathbb{N}$, then* TX *will appear in its ledger at time t', i.e., $\exists t' \in \mathbb{N} : t' \geq t \wedge \text{TX} \in \mathsf{L}^P[t']$. The interval $t' - t$ is* upper bounded *by u.*

In our model, even honest parties are not guaranteed to be online. Instead they may experience temporary offline periods during which they cannot send or receive messages. This corresponds, for example, to temporary power/network outages. More specifically, the adversary can choose to "suspend" any honest party P at any time, but must eventually "resume" that party at some later time. While suspended, the Persistence and Liveness properties do not hold for that party, indicated by a boolean "error variable" f_P. For the sake of counting corruptions, a suspended party is still considered honest, not corrupted.

Notion of Time

The state evolution of two distinct ledgers L_x and L_y may progress at different *time* intervals: In the time that L_x progresses *one* round, L_y may, for example, progress *forty* rounds (e.g., as in the case of Bitcoin [Nak08] and Ethereum [But14]). To correctly capture the ordering of transactions across L_x and L_y, we define a clock function τ which maps a given round on any ledger to the time on a global clock $\tau : r \to t$. We use this conversion implicitly in the rest of this chapter. For conciseness, we will use the notation $\mathsf{L}^P[t]$ to mean the ledger state in the view of party P at the round $r = \tau^{-1}(t)$ which corresponds to time t, namely $\mathsf{L}^P[\tau^{-1}(t)]$.

Transaction Model

A transaction TX, when included, alters the state of a ledger L by defining operations to be executed and agreed upon by consensus participants $P_1, ..., P_n$. The expressiveness of operations is thereby left for the particular system to define and can range from simple payments to execution of complex programs [Woo17]. For generality, we do not differentiate between specific transactions models (e.g. UTXO [Nak08] or account-based models [Woo17]).

3.1.3 Cross-Chain Communication System Model

When speaking of CCC, we consider the interaction between any two distributed systems X and Y with underlying ledgers L_x and L_y that have Persistence and Liveness, as defined in Section 3.1.2. We assume X and Y may employ different agreement protocols and make no assumptions on the respective composition of consensus participants, i.e., we do not assume that X and Y have the same participants. While the agreement protocol implemented by X will require a (semi-) synchronous communication *between participants of* X to ensure Persistence and Liveness [FLP85] (respectively for Y)[2], we make no such assumptions for the communication channels across participants of any two different systems X and Y. Specifically, we assume

[2]We note that, in the anonymous blockchain setting, more synchrony requirements are imposed than in the byzantine setting.

no bounds on message delay or deviations between local clocks for communication channels between participants $P_1, ..., P_n$ of X and participants $Q_1, ..., Q_m$ of Y.

We assume a *closed* system model as in [Lam89] with two participants P and Q, who can be any autonomous system, e.g. a process or even a human sitting in front of a computer. We assume P is a participant of distributed system X and Q is a participant of $chainY$. Both P and Q can influence the state evolution of the underlying system by (i) writing a transaction $\textsc{tx}$ to the underlying ledger L (commit), or (ii) by stopping to interact with the system (abort). We assume that P possesses a transaction $\textsc{tx}_P$, which can be written to L_x, and Q possesses $\textsc{tx}_Q$, which can be written to L_y. A function *desc* maps a transaction to some "description" which can be compared to an expected description value, e.g., specifying the transaction value and recipient (the description differs from the transaction itself in that it may not, for example, contain any signature). P possesses a description d_Q which characterizes the transaction $\textsc{tx}_Q$, while Q possesses d_P which characterizes $\textsc{tx}_P$. Informally, P wants $\textsc{tx}_Q$ to be written to L_y and Q wants $\textsc{tx}_P$ to be written to L_x. Thereby, $d_P = desc(\textsc{tx}_P)$ implies $\textsc{tx}_P$ is valid in X (at time of CCC execution), as it cannot be written to L_x otherwise (analogous for d_Q).

We assume P and Q know each other's identity, e.g. public key, and no (trusted) third party is involved in the communication between the two processes.

3.1.4 Formalization of Correct Cross-Chain Communication

The goal of cross-chain communication can be described as the synchronization of processes P and Q such that Q *writes* $\textsc{tx}_Q$ *to* L_y *if and only if* P *has written* $\textsc{tx}_P$ *to* L_x. Thereby, it must hold that $desc(\textsc{tx}_P) = d_P \wedge desc(\textsc{tx}_Q) = d_Q$. The intuition is that $\textsc{tx}_P$ and $\textsc{tx}_Q$ are two transactions that must either both, or neither, be included in L_x and L_y, respectively. For example, they can constitute an exchange of assets that must be completed atomically.

To this end, P must prove to Q that it created a transaction $\textsc{tx}_P$ which was included in L_x. Specifically, process Q must verify that at given time t the ledger state $\mathsf{L}_x[t]$ contains $\textsc{tx}_P$. A cross-chain communication protocol that achieves this goal, i.e., is correct, must hence exhibit

the following properties:

Definition 3 (Effectiveness)**.** *If both P and Q behave correctly and TX_P and TX_Q match the expected descriptions (and are valid), then TX_P will be included in L_x and TX_Q will be included in L_y. If either of the transactions is not as expected, then both parties abort.*

$$(desc(\text{TX}_P) = d_P \wedge desc(\text{TX}_Q) = d_Q \wedge f_P = f_Q = \bot \implies \text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y)$$
$$\wedge\, (desc(\text{TX}_P) \neq d_P \vee desc(\text{TX}_Q) \neq d_Q \implies \text{TX}_P \notin \mathsf{L}_x \wedge \text{TX}_Q \notin \mathsf{L}_y)$$

Definition 4 (Atomicity)**.** *There are no outcomes in which P writes TX_P to L_x at time t but Q does not write TX_Q before t', or Q writes TX_Q to L_y at t' but P did not write TX_P to L_x before t.*

$$\neg((\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \notin \mathsf{L}_y) \vee (\text{TX}_P \notin \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y))$$

Definition 5 (Timeliness)**.** *Eventually, a process P that behaves correctly will write a valid transaction TX_P, to its ledger L.*

From Persistence and Liveness of L, it follows that eventually P writes TX_P to L_x and Q becomes aware of and verifies TX_P.

Definition 6 (Correct Cross-Chain Communication (CCC))**.** *Consider two systems X and Y with ledgers L_x and L_y, each of which has Persistence and Liveness. Consider two processes, P on X and Q on Y, with to-be-synchronized transactions TX_P and TX_Q. Then a* correct *cross-chain communication protocol is a protocol which achieves $\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y$ and has Effectiveness, Atomicity, and Timeliness.*

Summarizing, Effectiveness, and Atomicity are safety properties. Effectiveness determines the outcome if transactions are not as expected or both transactions match descriptions and both processes are behaving correctly. Atomicity globally restricts the outcome to exclude behaviors that place a disadvantage on either process. Timeliness guarantees eventual termination of the protocol, i.e., is a liveness property.

3.1.5 The Generic CCC Protocol

We now describe the main phases of a generic CCC protocol, which can represent the transfer of goods, assets, or objects, between any two blockchain-based distributed systems X and Y. A visual representation is provided in Figure 3.1.

1) Setup. A CCC protocol is parameterized by the involved distributed systems X and Y and the corresponding ledgers L_x and L_y, the involved parties P and Q, the transactions TX_P and TX_Q as well as their descriptions d_P and d_Q. The latter ensure the validity of TX_P and TX_Q and determine the application-level specification of a CCC protocol. For example, in the case of an exchange of digital assets, d_P and d_Q define the asset types, transferred value, time constraints, and any additional conditions agreed by parties P and Q. Typically, the setup occurs out-of-band between the involved parties and we hence omit this step hereby.

2) (Pre-)Commit on X. Upon successful setup, a publicly verifiable commitment to execute the CCC protocol is published on X: P writes[3] transaction TX_P to its local ledger L_X^P at time t in round r. Due to Persistence and Liveness of L_x, all honest parties of X will report TX_P as stable ($\text{TX}_P \in \mathsf{L}_x$) in round $r + u_x + k_x$.

3) Verify. The correctness of the commitment on X by P is verified by Q checking (or receiving a proof from P) that (i) $d_P = desc(\text{TX}_P)$ and (ii) $\text{TX}_P \in \mathsf{L}_x$ hold. From Persistence and Liveness of X we know the latter check will succeed at time t' which corresponds to round $r + u_x + k_x$ on X, if P executed correctly.

4a) Commit on Y. Upon successful verification, a publicly verifiable commitment is published on Y: Q writes transaction TX_Q to its local ledger L_Y^Q at time t' in round r' on Y. Due to Persistence and Liveness of L_y, all honest parties of Y will report TX_Q as stable ($\text{TX}_Q \in \mathsf{L}_y$) in round $r' + u_y + k_y$, where u_y is the liveness delay and k_y is the "depth" parameter of Y.

4b) Abort. If the verification fails and/or Q fails to execute the commitment on Y, a CCC protocol can exhibit an abort step on X, i.e., "reverting" the modifications TX_P made to the

[3] In off-chain protocols [GMSR+19], the commitment can be done by exchanging pre-signed transactions or channel states, which will be written to the ledger at a later point.

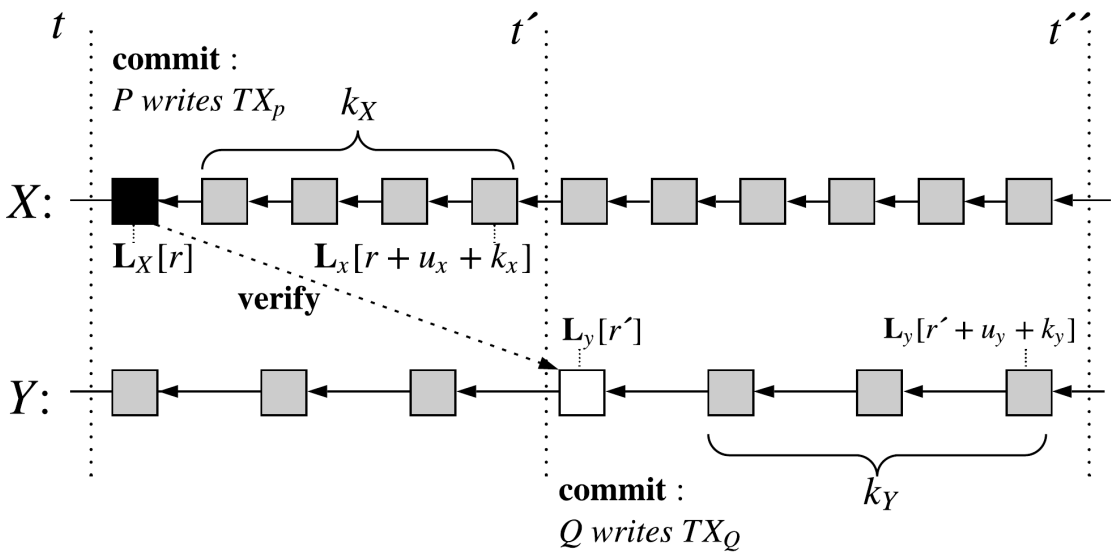


Figure 3.1: CCC between X and Y. Process Q writes TX_Q only if P has written TX_P. We set exemplary persistence delays for X and Y as $k_X = 4$ and $k_Y = 3$, and liveness delays as $u_x = u_y = 0$. We omit the optional the abort phase.

state of L_x. As blockchains are append-only data structures, reverting requires broadcasting an additional transaction $\text{TX}_{P'}$ which resets X to the state before the commitment of TX_P.

It is worth noting that some CCC protocols, specifically those facilitating *exchange* of assets, follow a two-phase commit design. In this case, steps 2 and 4a are executed in parallel, followed by the verification and (optional) abort steps on *both* X and Y. A further observation is that a CCC protocol necessarily requires a *conditional state transition* to occur on Y, given a state transition on X. As such, we do *not* consider (oracle) protocols which merely relay data across distributed ledgers [TE17, BCG15a, BGB17, Con17a, But16], as CCC protocols by themselves.

3.2 Impossibility of CCC without a Trusted Third Party

In this section, we show that CCC is impossible without a trusted third party by providing a reduction from the Fair Exchange problem [Aso98, PG99]. We first recall the definition of Fair Exchange and discuss the notion of trusted third parties. We then relate CCC to Fair Exchange, presenting a concrete instance of a cross-chain fair exchange protocol, and provide an outlook on incentives and economically rational behavior of parties.

3.2.1 Strong Fair Exchange Definition

On a high level, an exchange between two (or more) parties is considered fair if either both parties receive the item they expect, or neither do [ASW98b]. Fair exchange can be considered a sub-problem of fair secure computation [BK14], and is known to be impossible without a trusted third party [PG99, Yao86, EY80, Eve82]. In the following, we recall the definition of Fair Exchange.

Fair Exchange considers two processes (or parties) P and Q that wish to exchange two items (or asset): a_P owned by P against a_Q owned by Q. There exists a function *desc* that maps any exchangeable item (or asset) to a string describing it in "sufficient" detail (e.g. the value and recipient of a payment). The inputs of P to a Fair Exchange protocol are an item a_P and a description d_Q of the desired item. Analogous, the inputs for Q are a_Q and d_Q. To indicate that P is dishonest, an (boolean) error variable m_P is introduced (analogous, m_Q for Q) [Gär98]. A successful Fair Exchange is shown in Table 3.1 below.

Table 3.1: A successful Fair Exchange, as defined in [Aso98, ASW98a, ASW98b, PG99].

P		Q
Input : a_P, d_Q, Q		**Input** : a_Q, d_P, P
	fair exchange $\longleftrightarrow$	
Output : $a_Q(desc(a_Q) = d_Q)$		**Output** : $a_P(desc(a_P) = d_P)$
	or	
aborted		*aborted*

A successful Fair Exchange protocol must thereby fulfill the following properties:

Definition 7 (Effectiveness)**.** *If both P and Q behave correctly, i.e., $m_P = m_Q =$ false, and the items a_P and a_Q match the expected descriptions, i.e., $desc(a_Q) = d_Q \wedge desc(a_P) = d_P$, then P will receive a_Q and Q will receive a_P. If the items are not as expected, i.e., $desc(a_Q) \neq d_Q \vee desc(a_P) \neq d_P$, then both parties will abort the exchange.*

Definition 8 (Timeliness)**.** *Eventually P will transfer a_P to Q or abort, and Q will transfer a_Q to P or abort.*

Definition 9 (Strong Fairness)**.** *There are no outcomes in which Q receives a_P but P does not receive a_Q (Q aborts), or P receives a_Q but Q does not receive a_P (P aborts).*

Effectiveness determines the outcome of the exchange if P and Q are willing to perform the exchange and the items match the expected descriptions, or the items do not match the expected descriptions. (Strong) Fairness restricts the outcomes of the exchange such that neither party is left at a disadvantage. Timeliness ensures the eventual termination of the exchange protocol. Note: we do not provide a definition for "Non-repudiability" as this property is not a critical requirement for Fair Exchange protocols, but only becomes relevant in disputes after an exchange [ASW98a, PG99].

3.2.2 What is a Trusted Third Party?

Numerous recent works use *a single* distributed ledger such as Bitcoin and Ethereum to construct (optimistic) fair exchange protocols [BK14, And15, KB16, KZZ16, DEF18, KKAS+18]. They leverage smart contracts (i.e., programs or scripts), the result of which is agreed upon and enforced by consensus participants, to ensure the correctness of the exchange. These protocols thus use the consensus of the distributed ledgers as an abstraction for a trusted third party. If the majority of consensus participants are honest, correct behavior of processes/participants of the fair exchange is enforced – typically, the correct release of a_Q to P if Q received a_P.

A CCC protocol aims to achieve synchronization between *two* such distributed ledgers, both of which are inherently trusted to operate correctly. As we show below, a (possibly additional) TTP can be used to:

(i) Determine the outcome of the CCC protocol, i.e., confirm to the consensus participants of Y that $\textsc{tx}_P$ was included in L_x (and vice-versa);

(ii) Directly enforce correct behavior of Q (P), such that $\textsc{tx}_Q \in \mathsf{L}_y$ ($\textsc{tx}_Q \in \mathsf{L}_y$).

In the first case, we observe similarities to the concept of *failure detectors* [CT96], a construction used to introduce an implicit notion of time into distributed systems to solve consensus. In the

context of cross-chain communication, a failure detector - a trusted third party to the protocol - indicates whether a participant has failed on their commitment.

Similar to the abstraction of TTPs used in fair exchange protocols, in CCC, it does not matter how exactly the TTP is implemented, as long as it enforces the correct behavior of the participants. Strictly speaking, from the perspective of CCC there is little difference between a TTP consisting of a single individual and a committee where N out of M members must agree to take action (even though a committee is, without question, more resilient against failures) – contrary to the common assumptions made by the blockchain community.

3.2.3 Relating CCC to Fair Exchange.

We proceed to show that Correct Cross-Chain Communication is impossible without a trusted third party (TTP), under the system model of distributed ledgers, by reducing CCC to Fair Exchange [ASW98b, Aso98, PG99]. We recall, a fair exchange protocol must fulfill three properties: *Effectiveness*, *(Strong) Fairness* and *Timeliness* [PG99, Aso98].

Lemma 1. *Let M be a system model. Let C be a protocol that solves* CCC *in M. Then there exists a protocol S which solves Fair Exchange in M.*

Proof (sketch). Consider that the two processes P and Q are parties in a fair exchange. Specifically, P owns an item (or asset) a_P and wishes to exchange it against an item (or asset) a_Q owned by Q. Assume TX_P assigns ownership of a_P to Q and TX_Q transfers ownership of a_Q to P (specified in the "descriptions" d_P of TX_P and d_Q of TX_Q). Then, TX_P must be included in L_x and TX_Q must be included in L_y to correctly execute the exchange. In other words, if $\text{TX}_Q \in \mathsf{L}_y$ and $\text{TX}_P \in \mathsf{L}_x$, then P receives desired a_Q and Q receives desired a_P, i.e., P and Q fairly exchange a_P and a_Q.

We observe the definition of Timeliness in CCC is equivalent to the definition of Timeliness in fair exchange protocols, as defined in [PG99]. Effectiveness in fair exchange states that if P and Q behave correctly and do not want to abandon the exchange (i.e., $m_P = m_Q = \perp$), and

items a_P and a_Q are as expected by Q and P, then at the end of the protocol, P will own the desired a_Q and Q will own the desired a_P [PG99]. It is easy to see Effectiveness in CCC achieves exactly this property: if P and Q behave correctly and $desc(\text{TX}_P) = d_P$ and $desc(\text{TX}_Q) = d_Q$, i.e., TX_P transfers a_P to Q and TX_Q transfers a_Q to P, then P will write TX_P to L_y at time t and Q will write TX_Q to L_x before time t'. From Persistence and Liveness of L_x and L_y we know both transactions will eventually be written to the local ledgers of P and Q, consequently, all other honest participants of X will report $\text{TX}_P \in \mathsf{L}_X$ and honest participants of Y will report $\text{TX}_Q \in \mathsf{L}_Y$. From our model, we know that honest participants constitute majorities in both X and Y. Hence, P will receive a_Q and Q will receive a_P.

Strong Fairness in fair exchange states that there is no outcome of the protocol, where P receives a_Q but Q does not receive a_P, or, vice-versa, Q receives a_P but P does not receive a_Q [PG99]. In our setting, such an outcome is only possible if $\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \notin \mathsf{L}_y$ or $\text{TX}_P \notin \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y$, which contradicts the Atomicity property of CCC. □

We construct a protocol for Fair Exchange using CCC in Algorithms 1 - 4. Specifically, P and Q exchange assets a_P and a_Q across chains X and Y, if transaction TX_P is written to L_x and transaction TX_Q is written to L_y.

Algorithm 1 Fair Exchange using a CCC protocol

Result: $\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y$ (i.e.,P has a_P, Q has a_Q) **or** $\text{TX}_P \notin \mathsf{L}_x \wedge \text{TX}_Q \notin \mathsf{L}_y$ (i.e., no exchange)

```
setup(L_x,L_y,TX_P, TX_Q, d_P, d_Q)  if m_P = false then
    commit(TX_P, L_x); // P transfers a_P to Q

end

if (verify(TX_P, L_x, d_P) = true) ∧ m_Q = false then
    commit(TX_Q, L_y); // Q transfers a_Q to P

else
    abort(TX_Q, L_y); // Q does not transfer a_Q to P

end

if verify(TX_Q, L_y, d_Q) = false then
    abort(TX_P, L_x); // P recovers a_P

end
```

Algorithm 2 Commit(TX, L)

```
if valid(TX, L) then
  | Write TX to L;
end
```

Algorithm 3 Verify(TX, L, d)

```
if TX ∈ L ∧ desc(TX) = d then
  | return true
end
return false
```

Algorithm 4 Abort(TX, L)

```
if TX ∈ L then
  | Revert TX; // e.g. using a new transaction
else
  | //do nothing
end
```

It is left to show that CCC is defined under the same model as Fair Exchange. The distributed ledger model [GKL16] used in CCC assumes the same asynchronous (explicitly) and deterministic (implicitly) system model (cf. Section 3.1.3) as [PG99, FLP85]. Since P and Q by definition can stop participating in the CCC protocol at any time, CCC exhibits the same crash failure model as Fair Exchange [ASW98a, PG99] (and in turn Consensus [FLP85]). Hence, we conclude:

Theorem 1. *There exists no asynchronous* CCC *protocol tolerant against misbehaving nodes.*

Proof. Assume there exists an asynchronous protocol C which solves CCC. Then, due to Lemma 1 there exists a protocol that solves strong fair exchange. As this is a contradiction, there cannot exist such a protocol C. □

Our result currently holds for the closed model, as in [PG99, FLP85]. In the open model, P and Q can be forced to make a decision by the system (or environment), i.e., transactions

can be written on their behalf if they crash [KKJG+18]. In the case of CCC, this means that distributed system Y, or more precisely, the consensus of Y, can write TX_Q to L_y on behalf of Q (if P wrote TX_P to L_x). We observe that the consensus of Y becomes the TTP in this scenario: both P and Q must agree that the consensus of Y enforces correct execution of CCC. In practice, this can be achieved by leveraging smart contracts, similar to blockchain-based fair exchange protocols, e.g. [DEF18]. As such, we can construct a smart contract, the execution of which is enforced by consensus of Y, that will write TX_Q to L_y if P includes TX_P in L_x, i.e., Q is allowed to crash.

However, it remains the question of how the consensus participants of Y become aware that $\text{TX}_P \in \mathsf{L}_x$. In practice, a smart contract can only perform actions based on some input. As such, before writing TX_Q the contract/consensus of Y must observe and verify that TX_P was included in L_x. A protocol achieving CCC must hence make one of the following assumptions. Either, there exists a TTP that will ensure correct execution of CCC; or the protocol assumes P, or Q, or some other honest, online party (this can again be the consensus of Y) will always deliver a proof for $\text{TX}_P \in \mathsf{L}_x$ to Y within a known, upper-bounded delay, i.e., the protocol introduces some form of *synchrony* assumption. As argued in [PG99], we observe that *introducing a TTP and relying on a synchrony assumption are equivalent*:

Remark 1. *When designing a* CCC *protocol,one must chose between introducing a trusted third party, or, equivalently, assuming some synchrony on the network.*

The intuition behind this result is as follows. If we assume that process P does not crash and hence submits the necessary proof to the smart contract on Y, and that this message is delivered to the smart contract within a know upper bound, then we can be sure that CCC will occur correctly. Thereby, P intuitively represents its own trusted third party. However, if we cannot make assumptions on when the message will be delivered to the smart contract, as is the case in the asynchronous communication model between P (a participant of X) and the participants of Y, a trusted third party is necessary to determine the outcome of the CCC: the TTP observes $\text{TX}_P \in \mathsf{L}_x$ and informs the smart contract or directly enforces the inclusion of TX_Q in L_y. This illustrates how a TTP can be leveraged to enforce synchrony, i.e., timely

delivery of messages, in CCC protocols. While the two models yield equivalent results, the choice between a TTP and network synchrony impacts the implementation details of a CCC protocol.

Special Case: $P = X$ and $Q = Y$

So far, we considered P and Q as participants of X and Y respectively. A special case worthy of closer analysis is one where P represents the entirety of the consensus participants of X and Q the consensus participants of Y. While Theorem 1 holds even in this setting, the reduction from Fair Exchange may appear less intuitive: since both X and Y have Persistence and Liveness we can assume that neither P $(= X)$ nor Q $(= Y)$ will crash. Here, the challenge of establishing CCC arises from ensuring that Y correctly verifies the state transition of X, i.e., the majority of participants of Y obtain, validate and agree on a local view of L_x. While we defer a formal treatment of this case to future work, we provide an intuition below.

The key to relating this case to the Fair Exchange impossibility is to distinguish between communication models within and outside the system bounds of X and Y. Stemming from the Persistence and Liveness properties, the communication channels *within* X and Y respectively are (semi-)synchronous. However, CCC makes no such assumption beyond system bounds, i.e., there are no synchrony assumptions for communication channels between participants of X and participants of Y.

As such, while it appears possible to create a CCC protocol between X $(= P)$ and Y $(= Q)$ by requiring Y to collect and validate the state of X, (or vice-versa) it falls outside of the CCC model. Specifically, such a protocol requires (semi-)synchronous communication channels between participants of X and Y since the state validation of X is now subject to Persistence and Liveness of Y. This assumption is outside of the CCC model and, in fact, in line with our result. Intuitively, the existence of such a protocol between X and Y implies that consensus participants of X and Y overlap or one is a subset of the other, which in turn contradicts the CCC system model: arguably X and Y can no longer be considered two distinct systems in this setting.

For generality, we can further dissect the composition of a distributed system's participants into consensus participants and node operators. While both have (semi-)synchronous channels to other participants, maintain a local view of the ledger and can modify the latter, only consensus participants can finalize state transitions, i.e., a network participant will only consider transactions "stable" when signed off by a consensus participant [4]. It becomes clear that for Y to validate and agree on the state of X (requirement for CCC), there needs to be at least one honest and online party that is a node operator (not necessarily a consensus participant) of both X and Y and broadcasts the necessary the necessary state transition proof of L_x to Y.

3.2.4 Incentives and Rational CCC

Several workarounds to the fair exchange problem, including gradual release mechanisms, optimistic models, and partially fair secure computation [ASW98b, CC00, KL12, BK14], have been suggested in the literature. These workarounds suffer, among others, from a common drawback: they require some form of trusted party that does not collude with the adversary. Further, in case of an adversary-caused abort, honest parties must spend extra efforts to restore fairness, e.g., in the optimistic model, the trusted server must be contacted each time fairness is breached.

First suggested in the context of rational exchange protocols [Syv98], the economic dimension of blockchains enabled a shift in this paradigm: Rather than forcing an honest user to invest time and money to achieve fairness, the malicious user is economically punished when breaching fairness and the victim is reimbursed. This has paved the way to design *economically trustless* CCC protocols that follow a game-theoretic model under the assumption that actors behave rationally [ZHL+19]. We remark that malicious/altruistic actors can nevertheless breach CCC properties: even if there is no economic damage to parties P or Q, the correct execution of the communication protocol itself is not guaranteed.

[4] We consider accessing a node operated by someone else via a publicly exposed interface equivalent to being a node operator

3.3 The CCC Design Framework

With the impossibility result 3.2 and CCC model (Section 3.1.2) in mind, we now introduce a new framework for creating and evaluating CCC protocols.

A generic CCC protocol consists of three main phases: commit (on X), verify (and commit on Y), and an optional abort. The main challenge of designing a CCC protocol is hence to determine the necessary trust model for each phase, from one of the following: (*i*) relying outright on a TTP, (*ii*) relying on an explicit synchrony assumption, or (*iii*) a hybrid approach, where a TTP is only involved if synchrony is breached. The framework introduced below is structured as follows: for each CCC phase (subsection), we systematize the three possible trust models (TTP, synchrony, hybrid), outlining possible implementations and reasoning about practical considerations.

3.3.1 (Pre-)Commit Phase

The commit phase(s) of a CCC protocol typically involves the locking and unlocking of assets on chains X and Y, determined by the outcome of the protocol.

Model 1: Trusted Third Party (Coordinators) A *coordinator* is a TTP that is tasked with ensuring the correct execution of a CCC protocol. We classify coordinator implementations attending to two criteria: *custody of assets* and *involvement in blockchain consensus*. A coordinator (committee) can thereby be *static* (pre-defined) or *dynamic* (any user can join). And, finally, a CCC protocol can utilize *collateral* to incentivize correct behavior. We first introduce the classification criteria and then detail possible implementations of coordinators.

- *Custody of Assets.* Custody determines with whom the control over assets of (honest) participants resides. We differentiate between *custodians* and *escrows*. *Custodians* receive *unconditional* control over the participant's funds and are thus *trusted* to release them as instructed by the protocol rules. *Escrows* receive control over the participant's funds *conditional* to certain

prearranged constraints being fulfilled. Contrary to custodians, escrows can fail to take action, e.g. freeze assets, but cannot commit theft.

• *Involvement in Consensus.* Coordinators can optionally also take part in the blockchain consensus protocol. *Consensus-level* coordinators refer to TTPs that are additionally consensus participants in the underlying chain. This is the case, for example, if the commit step is performed on chain X and enforced directly by the consensus participants of X, e.g. through a smart contract or directly a multi-/threshold signature. *External* coordinators, on the other hand, refer to TTPs which are not represented by the consensus participants of the underlying blockchain. This is the case if (i) the coordinators are external to the chain X, e.g, the consensus participants of chain Y or other parties, or (ii) less than the majority of consensus participants of chain X are involved.

• *Election.* An important distinction to make is between *static*, i.e., unchanged over time (usually permissioned), and *dynamic* coordinators. A dynamic coordinator can be chosen by CCC participants for each individual execution or can be sampled by a pre-defined mechanism, as e.g. studied in [DW14, KJG+16, KK19, PS16] for Proof-of-Work and in [Mic16, KRDO17, DGKR18, BPS16] for Proof-of-Stake blockchains. We consider CCC protocols where any user can become a coordinator as *unrestricted* [ZHL+19], while protocols that require coordinators to register with some third party (or e.g. first acquire a token) as *restricted* [Kee19].

• *Incentives and Collateralization.* Instead of following a *prohibitive* approach, i.e., technically preventing or limiting coordinators from deviating from protocol rules, a CCC protocol can follow a *punishable* approach. That is, ensure misbehavior can be proven and penalized retrospectively. In the latter case, a coordinator will typically be required to lock collateral that can be *slashed* and allocated to (financially) damaged CCC participants.

Coordinator Implementations. We now detail the different coordinator types according to the aforementioned criteria and how they are implemented in practice.

• *External Custodians (Committees).* Instead of relying on the availability and honest behav-

ior of a single external coordinator, trust assumptions can be distributed among a set of N committee members. Decisions require the acknowledgment (e.g. digital signature) of at least $M \leq N$ members, whereby consensus can be achieved via Byzantine Fault Tolerant (BFT) agreement protocols such as PBFT [CL+99, KJG+16]. External custodians can be both static or dynamic, and collateralization can be added on involved blockchains to incentivize honest behavior.

- *Consensus-level Custodians (Consensus Committee)* are identical to external custodians, except that they are also responsible for agreeing on the state of the underlying ledger. This model is typically used in blockchain sharding [KKJG+18, ABSB+18], where the blockchain X on which the commit step is executed runs a BFT consensus protocol, i.e., there already exists a *static* committee of consensus participants that must be trusted for the correctness of CCC (Persistence and Liveness of X). Collateralization of Consensus Custodians is best handled on another blockchain, i.e., where the coordinators have no influence on consensus.

- *External Escrows (Multisignature Contracts).* External Escrows are a special case of External Custodians, where the coordinator is transformed from Custodian to Escrow by means of a *multisignature contract.* Multisignature contracts require signatures of a subset (or majority) of committee members *and* the participant P (e.g., the asset owner), i.e., $P + M, M \leq N$. The committee can thus only execute actions pre-authorized by the participant: it can at most freeze assets, but not commit theft.

- *Consensus-level Escrow (Smart Contracts)* are programs stored in a ledger that are executed and their result agreed upon by consensus participants [But14, C+16]. As such, trusting in the correct behavior of a smart contract is essentially trusting in the secure operation of the underlying chain, making this a useful construction for Escrows. Contrary to Consensus-level Custodians, who must actively follow the CCC protocol and potentially run additional software, with smart contracts consensus participants typically are not directly involved in the CCC protocol: interaction with the CCC smart contract is, by default, treated like any other state transition and no additional software/action is required. CCC protocols which rely on smart contracts typically involve cross-chain state verification (cf. Appendix A) to enforce correct

execution on both X and Y, and typically do not need additional collateralization, except as potential insurance against software bugs [KM20].

Model 2: Synchrony Assumptions (Lock Contracts) An alternative to coordinators consists in relying on synchronous communication between participants and leveraging locking mechanisms that harvest security from cryptographic hardness assumptions. In practice, so-called *lock contracts* are typically used in CCC protocols that facilitate asset exchanges and implement two-phase commit, where the same (*symmetric*) locks are created on both chains and released atomically.

• *Hash Locks.* A protocol based on hash locks relies on the *preimage resistance* property of hash functions: participants P and Q transfer assets to each other by means of transactions that must be complemented with the preimage of a hash $h := H(r)$ for a value r chosen by P – the initiator of the protocol – typically uniformly at random [Bit13, Tie13, Her18, MMK+17].

• *Signature-based Locks.* Protocols based on hash locks have limited interoperability as they require that both cryptocurrencies support the same hash function within their script language. Unfortunately, this assumption does not hold in practice (e.g., Monero does not even support a scripting language). Instead, P and Q can transfer assets to each other by means of transactions that require to solve the discrete logarithm problem of a value $Y := g^y$ for a value y chosen uniformly at random by P (i.e., the initiator of the protocol). In practice, it has been shown that it is possible to embed the discrete logarithm problem in the creation of a digital signature, a cryptography functionality used for authorization in most blockchains today [BN00, BBBF18, MMSS+18, TMSM19, EMSM19, Poe17, MSRL+19].

• *Timelock Puzzles and Verifiable Delay Functions.* An alternative approach is to construct (cryptographic) challenges, the solution of which will be made public at a predictable time in the future. Thus, P and Q can commit to the cross-chain transfer conditioned on solving one of the aforementioned challenges. Concrete constructions include timelock puzzles and verifiable delay functions. Timelock puzzles [RSW96] build upon inherently sequential functions where the result is only revealed after a predefined number of operations are performed. Verifiable

delay functions [BBBF18] improve upon timelock puzzles in that the correctness of the result for the challenge is publicly verifiable. This functionality can also be simulated by releasing parts of the preimage of a hash lock interactively bit by bit until it can be brute forced [BJZ+17].

Model 3: Hybrid (Watchtowers) Instead of fully relying on coordinators being available or synchrony assumptions among participants holding, it is possible to employ so-called *watchtowers*, i.e., service providers which act as a fallback if CCC participants experience crash failures. We observe strong similarities to optimistic fair exchange protocols [ASW98b, ASW98a, CC00]. Specifically, watchtowers take action to enforce the commitment, if one of the parties crashes or synchrony assumptions do not hold, i.e., after a pre-defined timeout [KNW19, ALS+18, MBB+18, AKW19]. This construction was first introduced and applied to off-chain payment channels [GMSR+19].

3.3.2 Verification Phase

The verification phase, during which the commitment on X is verified on Y (or vice-versa), can similarly be executed under different trust models, as detailed in the following. Thereby, it is also of relevance what exactly is being verified: the possibility or the consensus agreement on a state, a specific state transition (e.g. a transaction), or actual validation of a state under underlying consensus rules (we direct the interested reader to Appendix A for a detailed classification).

Model 1: Trusted Third Party (Coordinators). The simplest approach to cross-chain verification is to rely on a trusted third party (also referred to as *validators* [Woo15]) to handle the verification of the state changes on interlinked chains during CCC execution.

- *External Validators.* A simple approach is to outsource the verification step to a (trusted) third party, external to the verifying ledger (in our case Y), as in [TS15, Kyb19b]. The TTP can then be the same as in the commit/abort steps.

- *Consensus Committee / Smart Contracts.* Alternatively, the verification can be handled by

the consensus participants of the verifying chain [KKJG+18, DPW+16, Ler15], leveraging the assumption that misbehavior of consensus participants indicates a failure of the chain itself.

- *Verification Games.* Finally, rather than fully trusting coordinators, they can be used as a mere optimistic performance improvement by introducing dispute handling mechanisms to the verification process: users can provide (reactive) fraud proofs [ASB18] or accuse coordinators of misbehavior requiring them to prove correct operation [TR19, HB18, KGC+18].

Model 2: Synchrony Assumption. Instead of explicitly relying on a TTP, the verification phase can be implemented using:

- *Direct Observation.* Similar to the commit phase of CCC, one can require all participants of a CCC protocol to execute the verification phase individually: i.e., to run (fully validating) nodes in all involved chains. This is often the case in exchange protocols, such as atomic swaps using symmetric locks such as HTLCs [Bit13, Her18], but also in parent-child settings where one chain by design verifies or validates the other [BCD+14, GKZ19b, Ler18]. This relies on a synchrony assumption, i.e., requires CCC participants to observe commitments and act within a certain time, in order to complete the CCC.

- *Smart Contracts (Chain Relays).* The verification process can be encoded in smart contracts capable of verifying the commitment on X, so-called *chain relays*, as in the case of BTC-Relay [Con17a] – a smart contract on Ethereum which tracks the Bitcoin main chain and verifies BTC payments. Recently, chain relays capable of verifying succinct proofs of knowledge [ET18, WE20b] have been proposed, which can (theoretically) enable full validation of commitments on X (i.e., similar properties as full nodes, see Appendix A).

Model 3: Hybrid (Watchtowers). Just like in the commit phase, synchrony and TTP assumptions can be combined in the verification phase, such that a CCC protocol initially relies on a synchrony assumption, but can fall back to a TTP (*watchtowers*, c.f Section 3.3.1) to ensure correct termination if messages are not delivered within a pre-defined period.

3.3.3 Abort Phase

The abort of a CCC protocol is optional and is encountered typically in exchange protocols. Most other CCC protocols assume that once a commit is executed on X, no abort will be necessary.

Model 1: Trusted Third Party (Coordinators) Similar to the commit phase, an abort can be handled by a trusted third party and the possible implementations are the same as in Section 3.3.1. If a TTP was introduced in the commit phase, the abort phase will be typically handled by the exact same TTP.

Model 2: Synchrony Assumptions (Timelocks) Alternatively, it is possible to enforce synchrony by introducing timelocks, which abort the protocol after expiry. Specifically, to ensure that assets are *not locked up indefinitely* in case of a crash failure of a participant or misbehavior of a TTP entrusted with the commit step, all commit techniques can be complemented with *timelocks*: after the expiry of the timelock, assets are returned to their original owner. We differentiate between two types of timelocks:

- *Absolute timelocks*, where a transaction becomes valid only after a certain point in time, defined by a timestamp or a block (ledger at index i, $\mathsf{L}[i]$) located in the future.

- *Relative timelocks*, where a transaction TX_2 becomes valid only after a given time value or number of *confirmations* [bit18] have elapsed since the *inclusion* of another transaction TX_1 in the underlying ledger. Typically, TX_1 and TX_2 are related as TX_2 spends assets transferred in TX_1 [PD16]. Although more practical than absolute timelocks (no need for external clock), we are not aware of schemes allowing the creation of relative timelocks *across* ledgers.

Model 3: Hybrid (Watchtowers) As an additional measure of security, TTPs can be introduced as a fallback to timelocks in case CCC participants experience crash failures, e.g. in form of a watchtower [KNW19, ALS+18, MBB+18, AKW19] that recovers otherwise potentially lost assets. This is specifically useful in the case of atomic swaps using Hashed Timelock Contracts (HTLCs) [Bit13, Her18, Bit21a, PD16], when either party crashes after the hashlock's secret has been revealed.

Table 3.2: Classification of existing of Cross-Chain Communication protocols, in consideration of the selected TTP model (cf. Section 3.3) at each protocol step (commit, verify, abort). Notation for non-binary TTP values: ● uses a TTP, ○ fully relies on synchrony and availability of participants, ◑ hybrid. We also highlight if the TTP (committee) is static or changes dynamically, and whether collateral is utilzed to incentivize correct behavior of TTPs. We use the following abbreviations: **EC** for External Custodian, **CC** for Consensus Custodian, **EE** for External Escrow, **SC** for Smart Contract, **EV** for External Validator, **CM** for Consensus Committee, and **DO** for Direct Observation.

		Protocol	Trust Model at each CCC Protocol Phase							
			Commit on chain X				Verify & Commit on chain Y		Abort on chain X (optinal)	
			TTP	Dynamic?	Collateral?	Type	TTP	Type	TTP	Type
Exchange Protocols	(Atomic Swaps)	**Traditional Custodial Exchanges (e.g., [Bin22, BJZ+17])**	●	✗	✗	EC (single, restricted)	●	EV	●	EC (single, restricted)
		A2L [TMSM19]	◑	✗	✓	EE (multisig + signature Lock)	○	DO	◑	EE + Timelock
		Arwen [HLG19]	◑	✗	✗	EE (multisig + Hash Lock)	○	DO	◑	EE + Timelock
		Notarized HTLC Atomic Swaps [TS15]	○	-	-	Hash Lock	●	EV	◑	EE + Timelock
		HTLC Atomic Swaps [Bit13, Her18, Tie13, TS15]	○	-	-	Hash Lock	○	DO	○	Timelock
		ECDSA/DLSAG Atomic Swaps [MMSS+18, MSRL+19]	○	-	-	Signature Lock	○	DO	○	Timelock
		SPV Atomic Swaps [eth15, KZ18, ZHL+19, HLS19]	○	-	-	Standard payment	○	SC(chain relay)	○	Timelock
Migration Protocols	Cryptocurrency-backed Assets	**(Bidirectional) Chain Relays [KZ18, GKZ19b]**	○	-	-	SC	○	SC (chain relay)	-	-
		XCLAIM [ZHL+19], Dogethereum [TSB19]	●	✓	✓	EC (single, unrestricted)	○	SC (chain relay)	-?	-
		tBTC [Kee19]	●	✗	✓	EC (committee, restricted)	○	SC (chain relay)	-?	-
		Custodial Wrapped Assets (e.g., [Kyb19b, Ren20, Pto20])	●	✗	✗	EC (single, restricted)	●	EV	●	EC (single, restricted)
	Side-chains	**Federated Sidechains/Pegs [BCD+14, DPW+16, GKZ19b]**	●	✗	✗	EC (consensus of Y)	●	CM	-?	-
		RSK [Ler15, Ler18]	●	✗	✗	EC (consensus of Y)	●	CM	-?	-
	Sharding	**ATOMIX[KKJG+18],SBAC[ABSB+18], Fabric Channels[ACDCKK18]**	●	✗	✗	CC (shard X)	●	CM	●	CC (shard X)
		Rapidchain [ZMR18]	●	✗	✗	CC (shard X)	●	CM	-?	-
		XCMP [BCC+20]	●	✗	✗	EC (parent consensus)	●	CM	-?	-
	Boot-strapping	**Proof-of-Burn (Federated) [Ste12, KKZ20]**	○	-	-	SC / Burn address	●	CM	-	-
		Proof-of-Burn (SPV) [KKZ20]	○	-	-	SC / Burn address	○	SC (chain relay)	-	-
		Merged Mining/Staking [JZS+17, GKZ19b]	●	✗	✗	CC (consensus of X)	●	CM	-	-

? While not explicitly considered by the protocol, the TTP used for the commitment on X can, at its discretion, abort the CCC protocol manually/out-of-band in case of failure on Y.

3.4 Classification of Existing CCC Protocols

We now apply the CCC Design Framework introduced in Section 3.3 to classify existing CCC protocols. All CCC protocols observed in practice follow the Generic CCC Protocol model (cf. Section 3.1.5). For each protocol, we hence study and reason about the trust model (TTP, synchrony, hybrid) selected for each phase of the CCC process, and summarize our classification in Table 3.2.

In addition to applying the CCC Design Framework, we split existing proposals into two protocol families, based on their design rationale and use case, which has direct implications on the design choices: (i) *exchange* protocols, which synchronize the exchange of assets across two ledgers (Section 3.4.1), and (ii) *asset migration* protocols, which allow moving an asset or object to a different ledger (Section 3.4.2).

3.4.1 Exchange Protocols

Exchange protocols synchronize an atomic exchange of digital goods: x on chain X against y on Y. In practice, such protocols implement a *two-phase commit* mechanism, where parties first pre-commit to the exchange and can *explicitly abort* the protocol in case of disagreement or failure during the commit step.

(Pre-)Commit. Trivially, the commit phase can be handled by External Custodians: traditional, centralized exchanges require to deposit (commit) assets with a TTP before trading.

The longest-standing alternative to centralized solutions are atomic swaps via *symmetric locks* which rely on synchrony and cryptographic hardness assumptions. Counterparties P and Q lock (pre-commit) assets in on-chain contracts with identical release conditions on X and Y: spending from one lock releases the other, ensuring Atomicity of CCC. The first and most adopted implementation of symmetric locks are *hashed timelock contracts* (HTLCs) [Tie13, Bit13, Her18, TS15], where the same secret (selected by P) is used as a pre-image to identical Hash Locks on X and Y. To improve cross-platform compatibility, Hash Locks, which require

both chains to support (the same) hash functions, can be replaced with Signature Locks e.g., using ECDSA [MMSS+18] or group/ring signature schemes [MSRL+19].

On blockchains that support (near) Turing complete programming languages (e.g., Ethereum [But14]) the commitment on X can exhibit more complex locking conditions via smart contracts. In SPV atomic swaps [eth15, KZ18, ZHL+19, HLS19], assets of a party P are locked in a smart contract on X which is capable of verifying the state of chain Y (chain relay, cf. Section 3.3.2) - and unlocked only if counterparty Q submits a correct proof for the expected payment (commitment) on Y. The smart contract can be further extended to support collateralization and penalties for misbehaving counterparties (e.g., to mitigate optionally and improve fairness [HLY19, ZHL+19]).

Both symmetric and SPV atomic swaps suffer from usability challenges impeding adoption: they require users to be online and execute commitments in a timely manner to avoid financial damage (built-in abort mechanisms will be discussed later). Hybrid protocols seek to combine symmetric locks with TTP models to mitigate usability issues while avoiding full trust in a central provider. In Arwen [HLG19], parties P and Q commit to-be-exchanged assets into on-chain multisignature contracts on X and Y, establishing shared custody with an External Escrow (EE). Trades are executed similar to HTLC swaps, yet utilize the escrow to ensure correct and timely execution. A2L [TMSM19] follows a similar multisignature setup but utilizes adaptor signatures [AEE+20] to ensure Atomicity of trades: the escrow only forwards P's assets to Q if Q solves a cryptographic challenge, for which Q needs the help of P. Both Arwen and A2L require a complex on-chain setup process (similar to payment channels [PD16]) and rely on pre-paid fees (Arwen) or collateral (A2L) to protect the escrow from griefing attacks [CCLM09] - yielding them inefficient for one-time exchanges.

Verify. Contrary to traditional exchanges, where the custodial (operator) is also responsible for the verification phase, symmetric atomic swap protocols (including Arwen and A2L) require users to *directly observe* all chains involved in the CCC to verify the correct execution of the (pre-) commit phase. Notarized atomic swaps (e.g., as in InterLedger [TS15]) remove the online requirement for users by entrusting an External Validator (EV) e.g., a set of notaries,

with the verification of (and timely reaction to) the commitment on X- at the risk of the EV colluding with the counterparty to commit theft. A more robust approach, implemented in SPV atomic swaps, is the use of *chain relays*: the verification of the commit on X and the correct finalization of the CCC protocol (commit on Y) is executed by a smart contract on Y, enforced by the consensus of Y.

Abort. Exchange CCC protocols typically add timelocks to the release conditions of the commitments of X and Y to ensure an automatic abort of the CCC protocol after a pre-defined delay. This is to prevent indefinite lock-up of assets, should a party crash or misbehave. However, CCC protocols implementing timelocks impose strict online requirements on participants and expose them to race conditions. The initiator P of e.g., an HTLC swap can defraud counterparty Q by recovering assets on X if they remain unclaimed upon expiry of the timelock (e.g., if Q crashed). Some protocols, including A2L and Arwen, partially outsource this responsibility to TTPs [HLG19, TMSM19].

3.4.2 Asset Migration Protocols

Asset migration protocols temporarily or permanently move digital goods from one blockchain to another. Typically, this is achieved by obtaining a "write lock" on an asset x on chain X, preventing any further updates to x on chain X, and consequently creating a *representation* $y(x)$ on Y. The state of x can only be updated by modifying its "wrapped" version $y(x)$ on Y– comparable to the concept of *mutual exclusion* in concurrency control [Dij01]. The state changes of $y(x)$ will typically be reflected back to chain X by locking or destroying ("burning") $y(x)$ and applying the updates to x when it is unlocked.

Migration protocols only require to execute CCC synchronization across X and Y twice: creating and destroying $y(x)$. The "wrapped" representation $y(x)$ typically exhibits the same properties as "native" assets y, allowing seamless integration with applications on Y. For comparison, Exchange protocols require to set up and execute CCC for *each trade*. The main drawback of Migration protocols is the requirement of giving up custody over x, in the majority of cases to a TTP (cf. Table 3.2).

In practice, we identify four main use cases for Migration protocols: (i) *cryptocurrency-backed assets* used for transfers across heterogeneous blockchains (e.g., "wrapped" Bitcoin on Ethereum), (ii) communication across homogeneous chains (shards) in *sharded* blockchains, (iii) *sidechains* where a child chain is "pegged" to a parent for feature extensions, and (iv) *bootstrapping* of new block-chains using existing systems.

(Pre-)Commit. The simplest implementation of a Migration protocol (e.g., for cryptocurrency-backed assets) relies on a single, static TTP which receives unrestricted custody over the to-be-migrated assets during the commit phase (External Custodian) – for example, as implemented by wBTC [Kyb19b], a custodial platform for migrating Bitcoin to Ethereum.

Instead of relying on a single TTP, most CCC rely on a TTP committee to improve robustness against failures. Protocols connecting heterogeneous blockchains via cryptocurrency-backed assets, notably tBTC [Kee19], utilize a set of External Custodians (EC). In the tBTC protocol, currently deployed between Bitcoin and Ethereum, ECs construct a jointly controlled deposit public key on X via (ECDSA) threshold signatures [GGN16], to which users send (commit) to-be-migrated assets. The ECs must thereby lock up collateral on Y which is used to reimburse users in case the EC committee commits theft or crashes. At the time of writing, the implemented threshold signature scheme does not support fault attribution, i.e., it is impossible to distinguish between honest and malicious committee members when slashing collateral, requiring the EC set to be static and restricted. RenVM [Ren20] aims to replace threshold signatures with distributed key generation via secure multi-party computation [Gol98] but implements a centralized approach at the time of writing.

Sidechains [BCD+14, DPW+16, GKZ19b] establish a parent-child relationship between X and Y: the consensus committee of X (Consensus Custodian, CC) or Y (External Custodian, EC) is responsible for handling the correct deposit (commit) of x on X. In practice, implementations follow a similar approach to the heterogeneous setting: users deposit assets x to a public key with shared control among committee members, implemented e.g., via threshold / multisignature [IN83] schemes. Liquid [BCD+14, DPW+16], which coined the "sidechain" terminology, maintains an 11-of-15 multisignature, controlled by its consensus participants, to

migrate (lock/unlock) Bitcoin to and from the Liquid blockchain. RSK [Ler15, Ler18], a merge-mined [JZS+17] Bitcoin sidechain, currently follows the same approach as Liquid but envisions a Bitcoin protocol upgrade enabling miners to vote on migrating assets to RSK.

Similarly, *sharded blockchains*, which consist of a set of homogeneous shard-chains with a homogeneous, shared security model, utilize the consensus committee(s) available within the system for securing cross-shard migrations. While often considered as a separate topic in research, sharded blockchains exhibit built-in CCC protocols [AKKW19]: Migrated assets x are locked with the consensus of X (Consensus Custodian, CC) during the commit phase. A novelty compared to heterogeneous systems is the explicit consideration of n-to-m CCC protocols, such as ATOMIX [KKJG+18], SBAC [ABSB+18], and Fabric Channels [ACDCKK18], which require an explicit abort step as part of the two-phase commit design.

Recently, a new family of protocols following a permissionless design was introduced. XCLAIM [ZHL+19] and Dogethereum [TSB19] allow anyone to become a TTP and accept deposits (commits) of x on X, establishing a dynamic and unrestricted set of coordinators (External Custodians, ECs). The only requirement for registering as an EC is to lock collateral y on Y– the amount of y locked thereby determines the amount of x deposits (and hence minted $y(x)$) an EC can accept. While Dogethereum assumes a constant exchange rate between migrated x (equiv. $y(x)$) and collateral asset y, XCLAIM utilizes a multi-stage over-collateralization scheme to rebalance the economic value of committed x and locked collateral y. To enable ECs to join and leave the system at any point in time, XCLAIM implements a replacement/auction mechanism via cross-chain SPV atomic swaps, where collateral y can be exchanged for committed x held in custody.

In cases where X and Y support smart contracts, specifically chain relays, *bidirectional* chain relays [KZ18, GKZ19b] can be utilized, enabling non-custodial commitments on X and Y: locking of x and unlocking/minting of $y(x)$ is handled exclusively by smart contracts under the assumption of synchrony.

Proof-of-Burn [Ste12, KKZ20] follows a similar design, yet implements a unidirectional protocol: instead of being locked, x is provably destroyed ("burned"), and newly minted as $y(x)$ on Y. As

such, Proof-of-Burn is mostly used for bootstrapping new blockchains. Merged mining [JZS+17] was the first CCC protocol deployed in practice (2011 in Namecoin) and is used explicitly for bootstrapping purposes. Miners (stakers) of X can reuse PoW solutions (stake) to progress consensus on Y by including a commitment to Y's state in the ledger of X.

Verify. Asset migration protocols - with the exception of centralized, custodial services - rely on the consensus of chain Y to correctly verify the commitment on X. We observe two main implementation techniques: (i) under synchrony assumptions by using *chain relay* smart contracts, which cryptographically verify the correctness of the commitment on X, or (ii) by requesting the consensus committee of Y to explicitly sign off on the CCC execution. XCMP [BCC+20], a cross-shard protocol, adds an additional verification step: cross-shard transfers are verified by and included in a hierarchically "superior" parent chain – which in turn is verified by the target shard Y before commitment.

Abort. We observe that Asset migration protocols generally do not implement an explicit abort phase. Instead, they assume that if the commitment on X is executed correctly it will eventually be verified by chain Y, which in turn will result in a correct commitment on Y. An exception hereof are n-to-m transfers in sharded blockchains (e.g., ATOMIX [KKJG+18] and SBAC [ABSB+18]) which require an explicit abort phase. Such transfers follow a two-phase-commit protocol: assets on all source shards $X_1, ..., X_n$ are pre-committed and verified on all target shards $Y_1, ..., Y_n$, which in turn execute a pre-commitment. If a single target shard fails to reply with a pre-commitment (within some period), the CCC protocol is aborted on all other source and target shards.

3.4.3 Insights and General Observations

An interesting, yet expected insight is that performance and usability outweigh security considerations from a user's perspective. Decentralized and non-custodial CCC solutions have been proposed as early as 2013 (symmetric swaps [Tie13]) and 2015 (SPV swaps [eth15]), yet centralized providers remain the dominant cross-chain asset exchange facilitator. The recent rise of decentralized exchanges, which mostly operate within a single chain [Coi19], has boosted the

adoption of cryptocurrency-backed assets, although predominantly via custodial approaches: at the time of writing, 99% of "wrapped" Bitcoin on Ethereum has been issued through trusted, custodial services [Pul22a].

Decentralized CCC protocols still suffer from practical drawbacks hindering adoption. Symmetric atomic swaps impose strict online requirements on users. SPV atomic swaps, and similarly migration protocols such as XCLAIM and tBTC, make use of chain relays which are only feasible if Y supports smart contracts and the cryptographic primitives used in X. Orthogonal, collateralization, which allows to protect users from financial damage (cf. Section 3.2), incurs high capital requirements and opportunity cost – leading most users to resort to trusted, centralized solutions.

An interesting observation hereby is that sharded systems and sidechains do not necessarily benefit from decentralized CCC protocols. In fact, due to the homogeneous nature of the security models of X and Y in this setting, the use of the consensus committee(s) of X or Y as TTP for CCC does not introduce any additional (external) trust assumptions to the underlying systems.

3.5 CCC Challenges and Outlook

In this section, we provide an outlook on the (open) problems faced by CCC protocols and interesting avenues for future work.

3.5.1 Heterogeneous Models and Parameters Across Chains

Problems. Different blockchains leverage different system models and parameterizations, which, if not handled correctly by CCC protocols, can lead to protocol failures. For instance, the absence of a global clock across chains requires CCC participants to either agree on a trusted third party as means of synchronization or to rely on a chain-dependent time definition (e.g., block generation rates [GKL16]) which are often non-deterministic and hence unsafe for strictly

time-bound protocols [GKL16, ZHL+19]. A practical example hereof are race-condition attacks on symmetric exchange protocols such as HTLC atomic swaps, discussed in Section 3.4.

Another consideration are the security models of interconnected chains: while X and Y may exhibit well-defined security models, these are typically independent and not easily comparable (with the exception of sharding) – especially when combined within a CCC protocol. For instance, X may rely on PoW and thus assume that adversarial hash rate is bounded by $\alpha \leq 33\%$ [ES14, GKW+16, SSZ15]. On the other hand, Y may utilize PoS for consensus and similarly assume that the adversary's stake in the system is bound by $\beta \leq 33\%$. While similar at first glance, the cost of accumulating stake [GKR18, FKO+18] may be lower than that of accumulating computational power, or vice-versa [Bon16]. Since permissionless ledgers are not Sybil resistant [Dou02], i.e., provide weak identities at best, quantifying adversary strength is challenging even within a single ledger [AKWW19]. This task becomes nearly impossible in the cross-chain setting: not only can consensus participants (i) "hop" between different chains [MCJ17, KKSK19], destabilizing involved systems, but also (ii) be susceptible to bribing attacks executed cross-chain, against which there currently exist no countermeasures [MHM18, JSZ+19].

Following from different security models, the lack of homogeneous finality guarantees [SKK20] across blockchains poses another challenge for CCC. Consider the following: X accepts a transaction as valid when confirmed by k subsequent blocks e.g., as in PoW blockchains [GKL16]; instead, Y deems transactions valid as soon as they are written to the ledger ($k = 1$, e.g. [AGM18]). A CCC protocol triggers a state transition on Y conditioned on a transaction included in X, however, later an (accidental) fork occurs on X. While the state of X is reverted, this may not be possible on Y according to consensus rules – likely resulting in an inconsistent state on Y and financial damage to users.

Outlook. Considering the plethora of blockchain designs in practice, it is safe to assume a heterogeneous ecosystem for at least the near future. Protocol designers must hence carefully evaluate and consider the specifics of each interlinked chain when implementing CCC schemes: introduction of conservative lower bounds on transaction (commit) finality (hours/days rather

than minutes), analysis of computation and communication capabilities of consensus participants, and accounting for peer-to-peer network delays when utilizing a trusted third party as a global clock.

3.5.2 Heterogeneous Cryptographic Primitives Across Chains

Problems. Interconnected chains X and Y may rely on different cryptographic schemes or different instances of the same scheme. CCC protocols, however, often require compatible cryptographic primitives: a CCC protocol between a system X using ECDSA [JMV01] as its digital signature scheme and a system Y using Schnorr [Sch91] is only seamlessly possible if both schemes are instantiated over the same elliptic curve [MMSS+18]. This is one of the reasons Ethereum uses the same secp256k1 curve as Bitcoin [But20].

Similarly, CCC protocols using Hash Locks, e.g. HTLC swaps, require that the domain of the hash function has the same size in both X and Y– otherwise, the protocol is prone to *oversize preimage attacks* [Ja19], i.e., an attack where a transaction cannot be accepted by a chain because the representation of the preimage requires more bits than those previously allocated to store it.

Outlook. A design challenge in CCC protocols is thus the interoperability of chains in terms of (cryptographic) primitives as required in CCC protocols. In cases where interlinked chains implement different elliptic curves, zero-knowledge proofs may provide a workaround, yet at the cost of increased protocol complexity, as well as computation and communication costs [Noe20]. Our observations suggest that this is one of the main reasons for the lack of interoperability across current blockchain networks.

3.5.3 Collateralization and Exchange Rates

Problems. In recent works [ZHL+19, TSB19, KZ18, Kee19], we observe a trend towards collateralizing coordinators to prevent financial damage to users and incentivize correct behavior

of TTPs. Thereby, it is crucial to ensure that the provided collateral has sufficient value to outweigh potential gains from misbehavior. However, in the cross-chain setting, where insured assets and collateral are typically different, collateralized CCC protocols are forced to (i) implement measures against exchange rate fluctuations such as over-collateralization incurring capital inefficiencies for participants, and (ii) rely on (typically centralized) price oracles.

Outlook. Current CCC protocols, if at all, only provide minimal protection against exchange rate fluctuations, such as over-collateralization. An interesting avenue for future research is hence the design of dynamic collateralization e.g., based on the volatility of the locked/collateral assets. Decentralized price oracles already are an active field of research [RWG+18, ABV+18, TE17, EJS17, ZMM+19], yet as of this writing, oracles remain single points of failure in collateralized CCC protocols. Cryptocurrency-backed assets traded on decentralized exchanges, where trading data is available on-chain, may thereby provide a valuable source of information for cross-verification with centralized providers [ZHL+19].

3.5.4 Lack of Formal Security Analysis

Problems. While numerous CCC protocols have been deployed and used in practice, handling value transfers worth millions, most lack formal and rigorous security analysis. This lack of formal security guarantees opens the door to possible security threats. For instance, *replay attacks* on state verification, i.e., where proofs are re-submitted multiple times or on multiple chains, can result in failures such as double spending [MHM17] or counterfeited cryptocurrency-backed assets [ZHL+19]. Another security issue arises with *data availability*. Protocols employing cross-chain verification via chain relays typically rely on the timely arrival of proofs and metadata (block headers, transactions, ...). However, if an adversary can withhold this data from the verifying chain [ASB18], such protocols not only become less efficient but potentially vulnerable to double-spending and counterfeiting.

Outlook. This state of affairs calls for a rigorous and formal security analysis of existing CCC protocol – least those deployed in practice. In the meantime, ad-hoc solutions to the aforementioned security threats have been discussed in the community. For instance, protections

against replay attacks involving the use of sequence numbers, or chains keeping track of previously processed proofs [MHM17, SBABD19, But18]. Similarly, first attempts to mitigate the data availability problem via erasure coding have been suggested in [ASB18, AB19, YSL+19] – yet at the cost of protocol complexity and communication overhead.

3.5.5 Lack of Formal Privacy Analysis.

Problems. Privacy is a crucial property of financial transactions and hence applies to CCC protocols. Ideally, it should not be possible for an observer to determine which two events have been synchronized across chains (e.g., which assets have been exchanged and by whom). Unfortunately, CCC protocols deployed in practice lack formal privacy analysis, and numerous privacy issues have already been detected. For instance, existing works [MMK+17, GM16] leverage the fact that the same hash value is used on both chains involved in symmetric HTLC atomic swaps to trivially link exchanged assets and accounts. Other de-anonymization techniques enabled by CCC protocols include miner address clustering via blocks merge-mined across different cryptocurrencies [JZS+17], cross-linking of miner and user accounts cross-chain by analyzing blockchain forks [HH18, SSJ+19], and using public exchange datasets to trace cross-ledger trades [YKM19].

Outlook. The academic community has developed formal frameworks that permit rigorous analysis of the privacy properties in the context of exchange protocols [GM16, HAB+16, TMSM19, MMK+17, MMSS+18]. First techniques towards privacy-preserving CCC Exchange protocols via asymmetric and unlinkable locking techniques have been studied in [MMK+17, MMSS+18, RNS14, DH20], yet, at the time of writing, we are not aware of privacy enhancements for the more-widely adopted Migration protocols – an interesting avenue for future research.

3.6 Conclusion

Our systematic analysis of cross-chain communication as a new problem in the era of distributed ledgers allows us to relate (mostly) community-driven efforts to established academic research in database- and distributed systems research. We formalized the cross-chain communication problem and show it cannot be solved without a trusted third party – contrary to the assumptions often made in the blockchain community. Following this result, we introduced a framework for evaluating existing and designing new cross-chain communication protocols, based on the inherent trust assumptions thereof. We then provide classification and comparative evaluation, taking into account both academic research and the vast number of online resources, allowing us to better understand the similarities and differences between existing cross-chain communication approaches. Finally, by discussing implications and open challenges faced by cross-chain communication protocols, as well as the implications of interoperability on the security and privacy of blockchains, we offered a comprehensive guide for designing protocols, bridging multiple distributed ledgers.

Chapter 4

XCLAIM: Trustless, Interoperable Cryptocurrency-backed Assets

Blockchain-based cryptocurrencies enable secure and trustless transactions between parties. As a result, they have gained widespread adoption and popularity in recent years; there are currently over 12 000 different cryptocurrencies in operation, with a total market cap of USD 1.7 trillion [Coi22]. However, despite a growing and thriving ecosystem, cryptocurrencies continue to operate in complete isolation from one another: blockchain protocols provide no means by which to communicate or exchange data with external systems. Hence, achieving interoperability between blockchains remains an open challenge.

Centralized exchanges thus remain the preferred route to execute fund transfers and exchanges across blockchains. However, these services require trust and therefore undermine the very nature of the cryptocurrencies on which they operate, making them vulnerable to attacks [Moo13, Bal, Pag14, ABC16]. To overcome this, *decentralized exchanges* [Aur19, Eth22a, WB17, Air22, Kyb22] (*DEXs*) have recently emerged, removing the need to trust centralized intermediaries for blockchain transfers. However, the vast majority of DEXs only enable the exchange of *cryptocurrency-assets* within a single blockchain, i.e., they do not operate across blockchains (*cross-chain*). As such, it is only a handful of platforms [Kom19, Dec17] that actually support cross-chain exchanges through the use of *atomic cross-chain swaps* (ACCS) [Tie16, Bit20a,

BCD+14, Her18].

ACCS enable secure cross-chain exchanges, e.g. using *hashed timelock contracts* (HTLCs) [Bit21a, Chr15]. At present, they are the only mechanism to do this without necessitating trust. Unfortunately, they require several strong assumptions to maintain security, thus limiting their practicality: they are interactive, requiring all parties to be online and actively monitor all involved blockchains during execution; they require synchronizing clocks between blockchains and rely on pre-established secure out-of-band communication channels. In addition, they also incur long waiting periods between transfers and suffer the limitation that for every cross-chain swap, four transactions need to occur, two on each blockchain. This makes them expensive, slow, and inefficient.

We therefore present XCLAIM (pronounced *cross-claim*): the first generic framework for achieving trustless cross-chain exchanges using *cryptocurrency-backed assets*. In XCLAIM, blockchain-based assets can be securely constructed and one-to-one backed by other cryptocurrencies, for example, Bitcoin-backed tokens on Ethereum. Through the secure issuance, swapping, and redemption of these assets, users can perform cross-chain exchanges in a trustless and non-interactive manner, overcoming the limitations of existing solutions.

To achieve this, XCLAIM exploits publicly verifiable *smart contracts* to remove the need for trusted intermediaries and leverages *chain relays* [But16, Con17a, KZ19, BCD+14] for cross-chain state verification. Using these building blocks, XCLAIM constructs a publicly verifiable audit log of user actions on both blockchains and employs *collateralization* and *punishments* to enforce the correct behavior of participants. Thereby, XCLAIM follows a *proof-or-punishment* approach, i.e., participants must proactively prove adherence to system rules.

Due to its simple and efficient design, XCLAIM enables several novel applications, such as (i) *smart contract and application access* for legacy cryptocurrencies, (ii) *cross-chain payment channels*, where users can exchange payments *off-chain* across different blockchains in a trustless manner; (iii) *temporary transaction offloading*, where users temporarily tokenize their cryptocurrency on other blockchains to overcome network congestion and high fees; and (iv) *N-way and multi-party atomic swaps* allowing efficient and complex atomic swaps. Finally, as XCLAIM

maintains compatibility with existing standardized asset interfaces [VB15, Dex17], the issued assets are tradeable via existing decentralized exchanges, enabling these exchanges to operate cross-blockchain.

Contribution

In summary, the contributions of this chapter are the following:

- We define the notion of cryptocurrency-backed assets for blockchains and formulate goals for security and functionality (Section 4.1). We then present XCLAIM, a practical and secure system to construct cryptocurrency-backed assets without trusted intermediaries (Section 4.2-4.3).

- We provide a formal protocol specification for XCLAIM and analyze in detail the requirements for the underlying blockchains (Section 4.4). While the blockchain used to issue cryptocurrency-backed assets must support smart contracts, XCLAIM requires only base-ledger functionality on the backing side, supporting practically all cryptocurrencies.

- We implement XCLAIM(BTC,ETH), to the best of our knowledge, the first system for trustlessly issuing, transferring, swapping, and redeeming Bitcoin-backed tokens on Ethereum (Section 4.6). In our prototype, it costs USD 0.47 to issue, USD 0.04 to transfer, USD 0.19 to atomically swap, and USD 0.49 to redeem any given amount of cross-chain tokens[5]. We compare performance and costs to HTLC atomic swaps and show XCLAIM is 95.7% faster and 65.4% cheaper for 1000 swaps.

- Finally, we present and describe several novel applications enabled exclusively by XCLAIM, such as cross-chain payment channels and efficient N-way and multi-party atomic swaps (Section 4.7).

[5] According to exchange rates as of 30 November 2018.

4.1 System Overview

In this section, we first define cryptocurrency-backed assets. We then present the system model and actors in XCLAIM, as well as the network and threat models. Finally, we present XCLAIM's system goals.

4.1.1 Cryptocurrency-backed Assets (CbA)

Definition. We define *cryptocurrency-backed assets* (CBAs) as assets deployed on top of a blockchain I that are backed by a cryptocurrency on blockchain B. We denote assets in I as i and cryptocurrency on B as b. We use $i(b)$ to further denote when an asset on I is backed by b.

We extend the definition of assets by Androulaki et al. [ACDCKK18] and describe a CBA through the following fields:

- *issuing blockchain*, the blockchain I on which the CBA $i(b)$ is issued.
- *backing blockchain*, the blockchain B that backs $i(b)$ using cryptocurrency b.
- *asset value*, the units of the backing cryptocurrency b used to generate the asset $i(b)$.
- *asset redeemability*, whether or not $i(b)$ can be redeemed on B for b.
- *asset owner*, the current owner of $i(b)$ on I.
- *asset fungibility*, whether or not units of $i(b)$ are interchangeable.

We define a CBA as *symmetric* if the total amount of backing units b is equivalent to the total amount of issued units $i(b)$, i.e., $|b| = |i(b)|$, and as *asymmetric* if the CBA exhibits an alternate backing rate, i.e., $|b| \neq |i(b)|$. In XCLAIM, we restrict CBAs to be symmetric cryptocurrency-backed assets. Moreover, CBAs can be divided and merged back together as necessary. We defer the analysis of alternate CBAs, such as asymmetric and non-fungible CBAs, to future work.

4.1.2 System Model and Actors

XCLAIM operates between a backing blockchain B of cryptocurrency b and an issuing blockchain I with underlying CBA $i(b)$. To operate CBAs, XCLAIM further differentiates between the following actors in the system:

- **CbA Requester.** Locks b on B to request $i(b)$ on I.
- **CbA Sender.** Owns $i(b)$ and transfers ownership to another user on I.
- **CbA Receiver.** Receives and is assigned ownership over $i(b)$ on I.
- **CbA Redeemer.** Destroys $i(b)$ on I to request the corresponding amount of b on B.
- **CbA Backing Vault (*vault*).** A (non-trusted) intermediary liable for fulfilling redeem requests of $i(b)$ for b on B.
- **Issuing Smart Contract (iSC).** A public smart contract responsible for managing the correct issuing and exchange of $i(b)$ on I. The iSC ensures the correct behavior of the *vault*.

To perform these roles in XCLAIM, actors are identified on a blockchain using their public/private key pairs. As a result, the *creator*, *redeemer*, and *vault* must maintain key pairs for both blockchains B and I. The *sender* and *receiver* only need to maintain key pairs for the issuing blockchain I. iSC exists as a publicly verifiable smart contract on I.

4.1.3 Distributed Ledger Model

We use the terms *distributed ledger* and *blockchain* as synonyms and adapt the distributed ledger model based on [ZABZ+19, GKL15], and as introduced in Section 3.1.2.

We consider two distributed systems B and I that each consist of a set of participants and employ a consensus protocol to agree on a sequence of transactions. We assume the security

model (threat, network, and cryptographic assumptions) of the consensus protocol holds: the fraction of consensus participants f or computational power α corrupted by an adversary is bounded by the threshold necessary to ensure the security of the consensus protocol. For Proof-of-Work blockchains, we therefore assume $\alpha \leq 33\%$ [ES14, GKW+16, SSZ15].

In this work, we only consider distributed systems that maintain robust transaction ledgers [GKL15], meaning they satisfy persistence and liveness with high probability to the security parameters.

We refer to Section 3.1.2 for a detailed discussion of assumptions and notation.

4.1.4 Network Model

For the underlying network, we make the same assumptions as in prior work [KRDO17], [KKJG+18], namely that (i) honest nodes are well connected and (ii) communication channels between these nodes are (semi-)synchronous, i.e., messages sent between honest participants will be received within a known maximum delay Δ (Δ^B for B and Δ^I for I). For simplicity, we assume the liveness delay parameter u is hidden in Δ. Additionally, we assume that all participants are aware of the smart contract iSC, and hence of the values publicly stored in it.

4.1.5 Threat model

We assume that the cryptographic primitives of B and I are secure and that adversaries are computationally bounded. Adversaries are fully adaptive, i.e, can freely choose controlled/corrupted participants for each time window. Adversaries may also perform arbitrary actions to maximize their economic value, such as delay or withholding transactions on protocol-level, reading unconfirmed transactions in the network, and performing Sybil attacks. Note that under these assumptions (and the persistence and liveness properties of I) the adversary cannot tamper with the correct execution of the smart contract iSC.

To keep track of and react to exchange rate fluctuations between i and b, we assume an oracle $\mathcal{O}$ provides the iSC with the exchange rate $\varepsilon(i, b) \in \mathbb{R}_{\geq 0}$. We further assume that there exists a

lower bound $\Omega(\varepsilon(i,b))$ for the exchange rate of i to b, below which adhering to protocol rules no longer represents the equilibrium strategy of rational adversaries [Nas51, NRTV07]. Therefore, in case of extreme devaluation of i relative to b, we assume a delay $\Delta_{\Omega(\varepsilon(i,b))}$ before this lower bound is reached such that $\Delta_{\Omega(\varepsilon(i,b))} < \Delta^I$, i.e., an honest user can include a transaction in I before $\varepsilon(i,b)$ drops below $\Omega(\varepsilon(i,b))$.

4.1.6 System Goals

Under the blockchain, network, and threat models specified above, in Sections 4.1.2-4.1.5, we derive the following desirable *security properties* for XCLAIM with regards to CBAs:

- **Auditability.** Any user with read-access to blockchains B and I can audit the operation of XCLAIM and detect protocol failures.
- **Consistency.** No CBA units $i(b)$ can be issued without the equivalent amount of backing currency b being locked, i.e., that $|b| = |i(b)|$.
- **Redeemability.** Any user can redeem CBAs $i(b)$ for backing currency b on B, or be reimbursed with equivalent economic value on I.
- **Liveness.** Any user in XCLAIM can issue, transfer and swap CBAs without requiring a third party, i.e., liveness relies only on the secure operation of B and I.
- **Atomic Swaps.** Users can atomically swap XCLAIM CBAs against other assets on I or the native currency i.

Furthermore, we derive the following desirable *functional properties* for XCLAIM:

- **Scale-out.** The total amount of CBAs available for circulation increases with the total amount[6] of backing currency locked up in blockchain B. Any user can contribute to this amount by assuming the role of the *vault*.

[6]Specifically, locked collateral. To become a *vault*, a user must provide at a pre-defined minimal amount of collateral in i; cf. Section 4.3.5.

- **Compatibility.** XCLAIM does not rely on a single cryptocurrency implementation with a set of specific features. Instead, it allows issuing assets $i(b)$ on any blockchain I that supports smart contracts[7], backed by any blockchain B that supports only basic fund transfers between parties. This enables XCLAIM to maintain backward compatibility with existing blockchains that do not provide smart contract support, such as Bitcoin.

4.2 Strawman Solution and Design Roadmap

In this section, we present a strawman solution, CENTRALCLAIM, that outlines how a CBA-based system with the actors defined in Section 4.1.2 might operate. We use CENTRALCLAIM to highlight the challenges faced by XCLAIM in achieving the goals from Section 4.1.6. Finally, we lay out a design roadmap for the secure design of XCLAIM. We present XCLAIM in Section 4.3.

4.2.1 Strawman Solution

CENTRALCLAIM proposes the use of a single trusted intermediary on the backing blockchain B that takes the role of the *vault*. The iSC is a smart contract deployed on the issuing blockchain I. The *vault* is registered with the iSC, i.e., the iSC can verify the *vault*'s digital signature and knows the *vault*'s public key. As defined in Section 4.1.2, I is responsible for managing the correct issuing and exchange of $i(b)$ on I.

We assume a user Alice controls units of b on a blockchain B, while a user Dave controls units of i on a blockchain I. Alice wishes to create B-backed assets $i(b)$ and transfer them to Dave on I. Dave, at some later point in time, wishes to redeem his units of $i(b)$ for the corresponding amount of b.

To achieve this, CENTRALCLAIM offers four protocols: *Issue*, *Transfer*, *Swap* and *Redeem*. For simplicity, we omit any processing fees charged by the *vault* or the iSC for the use of the service. We also omit the cost of transaction fees on the underlying blockchains B and I.

[7]Turing completeness is not required, as discussed in Section 4.4.3.

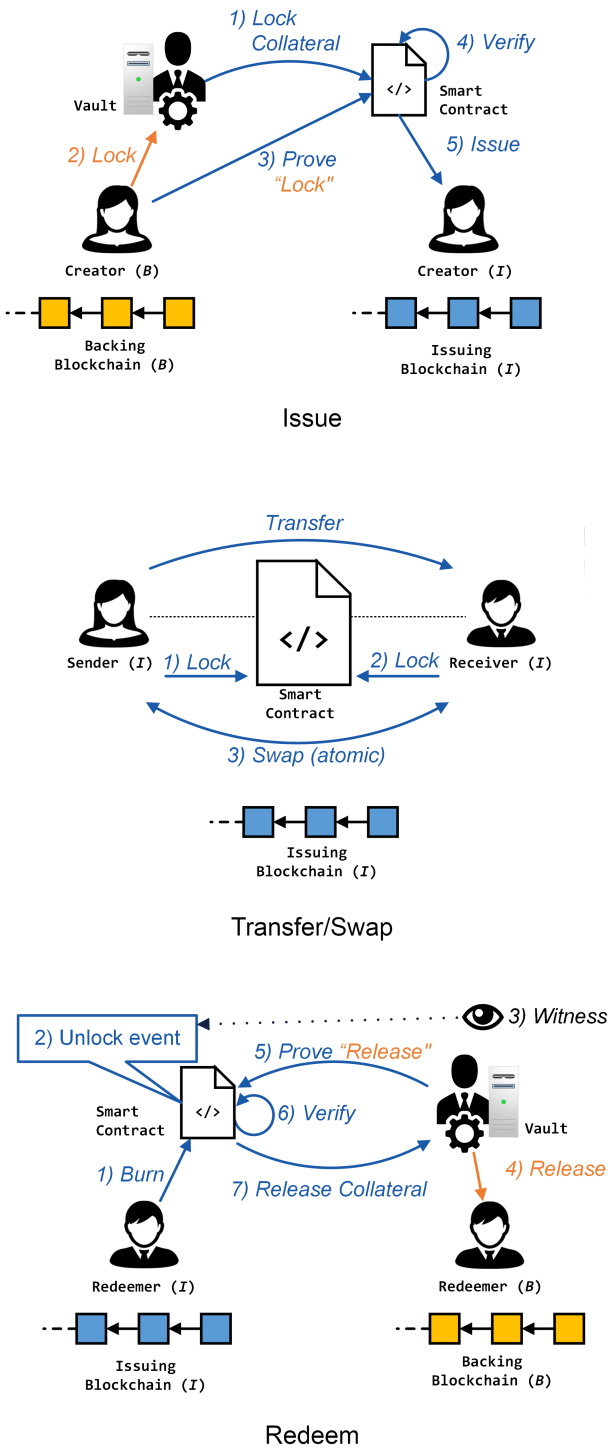


Figure 4.1: High-level overview of the *Issue*, *Swap* and *Redeem* protocols in XCLAIM's (under successful execution). All parties interact with the iSC, creating a publicly verifiable audit log. Correct behavior is enforced by (i) over-collateralizing the *vault* and (ii) cross-chain transaction inclusion proofs. When issuing, the *creator* proves the correctness of the lock making *Issue* *non-interactive*. Safety is ensured by forcing the *vault* to *proactively* prove the correctness of the *Redeem* process. As a result, XCLAIM enforces *Transfer* and *Swap* occur consistently on the backing (B) and issuing (I) blockchains.

Protocol: Issue. Alice (*creator*) locks units of b with the *vault* on B to create $i(b)$ on I:

1. *Setup.* First, Alice verifies the iSC smart contract is available on chain I, i.e., the issuing blockchain, and identifies the single backing intermediary on B, i.e., the *vault*.

2. *Lock.* Alice generates a new public/private key pair on I and locks funds b with the *vault* on B in a publicly verifiable manner, i.e., by sending b to the *vault*. As part of locking these funds with the *vault*, Alice also specifies where the to-be-generated $i(b)$ should be sent, i.e., Alice associates her public key on I with the transfer of b to the *vault*.

3. *Create.* The *vault* confirms to the issuing smart contract iSC via a signed message that Alice has correctly locked her funds and forwards Alice's public key on I to the iSC. The iSC verifies the *vault*'s signature, then creates and sends $i(b)$ to Alice, such that $|i(b)| = |b|$.

Protocol: Transfer. Alice (*sender*) transfers $i(b)$ to Dave (*receiver*) on I:

1. *Transfer.* Alice notifies the iSC that she wishes to transfer her $i(b)$ to Dave (public key) on I. The state of the iSC is updated and Dave becomes the new owner of $i(b)$.

2. *Witness.* The *vault* witnesses the change of ownership on I through iSC, and no longer allows Alice to withdraw the associated amount of locked b on B. The process for any further transfers from Dave to other users is analogous.

Protocol: Swap. Alice (*sender*) atomically swaps $i(b)$ against Dave's (*receiver*) i on I:

1. *Lock.* Alice locks $i(b)$ with the iSC.

2. *Swap.* If Dave locks the agreed-upon units of i (or any other asset on I) with the iSC within delay Δ_{swap}, the iSC updates the balance of Dave, making him the new owner of $i(b)$, and assigns Alice ownership over i.

3. *Revoke.* If Dave does not correctly lock i with the iSC within Δ_{swap}, the iSC releases locked $i(b)$ to Alice.

4. *Witness.* If the swap is successful, the *vault* witnesses the change of ownership of $i(b)$ and no longer allows Alice to redeem the associated amount.

Protocol: Redeem. Dave (*redeemer*) locks $i(b)$ with the iSC on I to receive b from the *vault* on B; $i(b)$ is then destroyed:

1. *Setup.* Dave creates a new public/private key pair on B.
2. *Lock.* Next, Dave locks $i(b)$ with the iSC on I and requests the redemption of $i(b)$. Thereby, Dave also specifies his new public key on B as the target for the redeem.
3. *Release.* The *vault* witnesses the locking and redemption request of $i(b)$ on I and releases funds b to Dave's specified public key on B, such that $|b| = |i(b)|$.
4. *Burn.* Finally, the *vault* confirms with the iSC that b was redeemed on B, and the iSC destroys, or *burns*, the locked $i(b)$ on I.

4.2.2 Strawman Limitations and Properties

While CentralClaim, as presented in Section 4.2.1, already provides sufficient functionality for issuing, transferring, swapping and redeeming CbAs, it does not achieve all the goals defined in Section 4.1.6. Namely, it does not achieve **Consistency**, **Redeemability** and **Liveness**. This is because CentralClaim is inherently centralized around a single *vault*, and trusts the *vault* to behave correctly. This is fundamentally insecure, however, as the *vault* is economically rational and therefore incentivized to misbehave.

For example, the *vault* is trusted to monitor the backing chain B for newly created locks of b and notify the iSC via a signed transaction on I. Should the *vault* fail to do this, it can steal the locked funds and violate **Consistency**. Similarly, the *vault* is trusted to release the correct amount of b on B when a *redeemer* requests the redemption of $i(b)$. Failing to do this allows the *vault* to steal the locked b and break **Redeemability**. Finally, CentralClaim also inherently violates **Liveness**; it exhibits a single point of failure, as backing-funds are locked with a single

intermediary, the *vault*. The *vault* is therefore assumed to be interactive, i.e., always online. As such, even in the case that the *vault* behaves honestly, CentralClaim can fail to achieve **Liveness**, e.g. due to denial-of-service and eclipse attacks [HKZG15] on the *vault*.

Surprisingly, however, CentralClaim already exhibits significant advantages over centralized systems offering digital tokens backed by real-world assets, e.g. the US dollar [Tet16]. Specifically, CentralClaim achieves **Auditability**, allowing users to detect if any actors misbehave, and **Atomic Swaps**, enabling secure swaps of assets and cryptocurrency.

It is easy to see how CentralClaim achieves **Auditability**: for successful execution, *Issue*, *Transfer*, *Swap* and *Redeem* all require secure transaction inclusion in blockchains B and I with security parameters k^B and k^I. Users can therefore detect both crash and Byzantine failures if incorrect transactions are published or transactions are missing from each blockchain. As such, an adversary could only interfere with this by (i) preventing transaction inclusion in B and I; or (ii) stopping a user from receiving messages broadcast by other nodes on B and I. Both attack vectors are not possible under the blockchain and (semi-)synchronous network models.

Likewise, it is easy to see how CentralClaim achieves **Atomic Swaps**: the *Swap* protocol is exclusively executed by the iSC on I. Specifically, to initiate *Swap*, the *sender* locks $i(b)$ in the iSC via a transaction on I. By construction, the iSC will only release $i(b)$ to the receiver if the *receiver* locks the correct amount of i with the iSC within Δ_{swap}. Otherwise, $i(b)$ is released back to the *sender*. An adversary cannot prevent the atomicity of *Swap*: this would require tampering with the iSC, which is not possible under the assumptions of the blockchain and threat models.

Finally, CentralClaim also provides **Compatibility**, as the only operation executed on the backing chain B is a simple transfer of funds to the *vault*. A detailed overview of operational requirements is provided in Section 4.4.3.

4.2.3 XCLAIM Design Roadmap

To address the security challenges and limitations of CentralClaim, we outline the design roadmap for Xclaim and introduce the building blocks used in its construction:

1. In Section 4.3.2, we remove the trust required by the *vault* during the issuing of CbAs, and make the issuing process non-interactive, thus achieving **Consistency** and **Liveness.** For this, we use *chain relays* to allow programmatic verification of transaction inclusion proofs for B on I and require all parties to *proactively* prove correct behavior.

2. In Section 4.3.3, we show how to incentivize the correct behaviour of the *vault* during CbA redemption through the introduction of *collateralization* and *punishments*, enforcing a *proof-or-punishment* model. We highlight race conditions during *Issue* due to collateralization, and present two effective mitigations: (i) *deferred collateral withdrawal* and (ii) *collateralized issue commitments.* Hence, we achieve **Redeemability** under a *fixed* exchange rate $\varepsilon(i, b)$.

3. In Section 4.3.4, we show how to prevent collateral deterioration due to exchange rate fluctuations by introducing (i) *over-collateralization*, (ii) *collateral adjustment* and (iii) *automatic liquidation.* As a result, Xclaim achieves **Redeemability** under *non-constant* exchange rates.

4. Finally, in Section 4.3.5, we achieve **Scale-Out** by removing single points of failure in CentralClaim. We do this by making Xclaim a multi-*vault* system where *any user* can assume the role of the *vault*.

4.3 XCLAIM Secure Design

This section presents the secure design of Xclaim. We first provide the high-level overview and intuition behind Xclaim, and then follow the technical roadmap outlined in Section 4.2 to provide a detailed system description.

4.3.1 XCLAIM Overview

XCLAIM overcomes the limitations of CENTRALCLAIM through three primary techniques: (i) constructing secure audit logs on both the backing blockchain B and the issuing blockchain I to trace all actions in the system; (ii) transaction inclusion proofs via chain relays to prove correct behavior on the backing blockchain B to the iSC; and (iii) collateralization to incentivize correct behavior through *proof-or-punishment*. We provide a brief overview and intuition for XCLAIM below. Figure 4.1 illustrates the *Issue*, *Transfer*, *Swap* and *Redeem* protocols in XCLAIM, while the design of the issuing smart contract iSC is shown in Figure 4.2.

Similar to CENTRALCLAIM, in XCLAIM, funds on the backing blockchain B are secured by backing intermediaries, *vaults*. The *vaults* store locked coins b on blockchain B and handle issue and redeem requests. To avoid necessitating trust in the *vault* however, XCLAIM uses collateral to incentivize behavior; XCLAIM requires actors, such as the *vault*, to deposit collateral on blockchain I, owned by the iSC. Every action in XCLAIM is then logged securely via the iSC and misbehaving actors, such as the *vault*, are punished by *slashing* collateral belonging to them, and reimbursing wronged actors. XCLAIM ensures deposited collateral is always sufficient, even in case of exchange rate fluctuations between b and i.

For the iSC to ensure that correct behaviors have taken place on blockchain B, where the iSC does not have direct visibility, XCLAIM uses *chain relays*. Chain relays provide external blockchain data, such as the transactions in blockchain B, to the iSC executing on I. As such, the iSC can trace every action by every actor in the system across blockchains. Actors in XCLAIM therefore proactively prove their honest behavior to the iSC via the chain relay; failure to do so results in punishment. By combining secure audit logs, chain relays, and collateralization in this way, XCLAIM can overcome the limitations of CENTRALCLAIM, and achieve the properties defined in Section 4.1.6.

4.3.2 Chain Relays: Cross-Chain State Verification

As outlined in Section 4.3.1, XCLAIM employs chain relays [But16, Con17a, KZ19] to provide data from the backing blockchain B to the iSC on the issuing blockchain I. We use chain relays to make the issuing of assets $i(b)$ on I non-interactive. For this, XCLAIM introduces a chainRelay component to the smart contract iSC (cf. Figure 4.2). The chainRelay is capable of interpreting the state of the backing blockchain B and provides functionality comparable to an SPV or light client [Bit19, BCD+14, Wik20, Tec21]. That is, a chainRelay stores and maintains block headers from blocks in B on I, and provides two functionalities to the iSC: *Transaction inclusion verification* and *Consensus verification*:

- **Transaction inclusion verification.** The chainRelay stores every block header in the backing blockchain B on I. Each block header in chainRelay contains the root of the Merkle tree [Mer87] containing all transactions (or their identifiers) for that block. To verify the correct inclusion of a transaction in a block in B, it is sufficient to provide the Merkle tree path from the root to the leaf containing the transaction (identifier) and the transaction data itself. This verification can then be performed in a non-interactive manner by the chainRelay as part of the iSC.

- **Consensus verification.** The chainRelay can also verify that any given block header is part of the backing blockchain B, i.e., has been agreed upon by the majority of consensus participants. In XCLAIM, consensus verification depends on the consensus mechanism used by the backing blockchain B. For Nakamoto consensus [Nak08], the chainRelay must (i) know the difficulty adjustment policy and (ii) verify that the received headers are on the chain with the most accumulated Proof-of-Work [Con17a, KZ19]. For Proof-of-Stake blockchains, e.g Ouroboros [KRDO17], the chainRelay must (i) be aware of the protocol/staking epochs and (ii) verify the signature membership of elected leader(s) for the threshold/multi-signatures of block headers [GKZ19a]. For permissioned (Proof-of-Authority) systems, the verification is analogous, or simpler, if the consensus participants are pre-defined [Vuk15]. We provide a formal definition for the necessary functionality of Proof-of-Work chain relays in Appendix B.

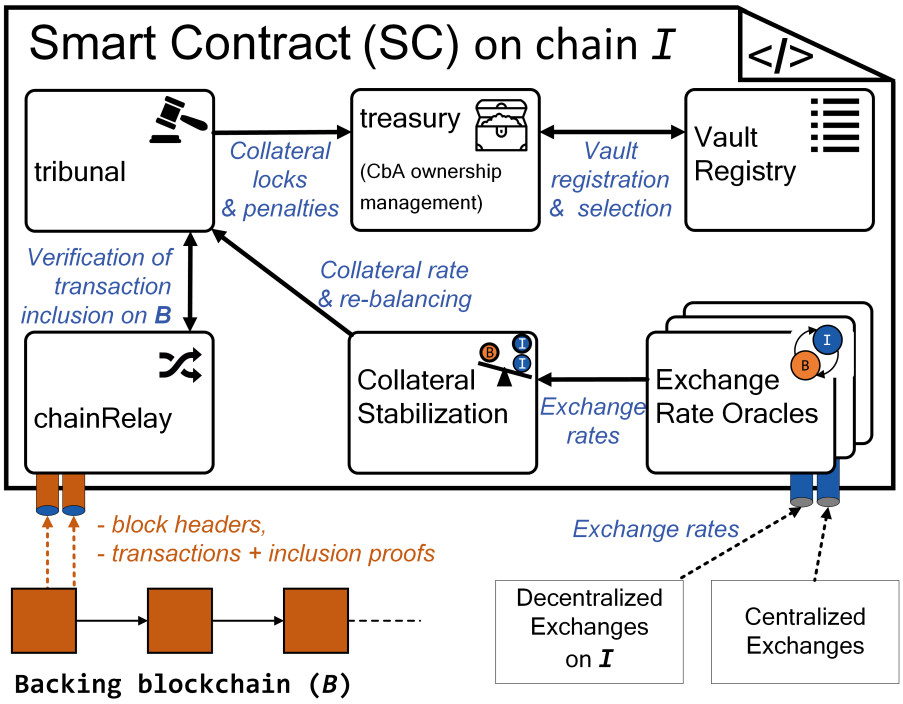


Figure 4.2: High level overview of the architecture of the XCLAIM smart contract (iSC) and the interactions between its components. References to sections introducing each component are provided. The treasury refers to the basic ledger functionality of I.

XCLAIM uses the chainRelay to modify the *Issue* protocol presented in CENTRALCLAIM (Section 4.2.1): after locking funds b with the *vault*, the *creator* must prove to the chainRelay that funds were locked correctly by presenting the transaction generated when sending b to the *vault* in B. The chainRelay can then verify that the given transaction has been securely included in B and the funds were locked correctly. If successful, the chainRelay triggers the automatic issuing of the corresponding amount of $i(b)$ in the iSC.

Similarly, XCLAIM also modifies the *Redeem* protocol: upon a redeem request being made by the user, the *vault* is required to prove that (i) the funds b were released to the *redeemer* and (ii) the released amount corresponds to the burnt CBA, i.e., $|b| = |i(b)|$. This is done by presenting the chainRelay with the transaction that sends b to the *redeemer* within a maximum delay Δ^{I}_{redeem}. Should the *vault* fail to comply, it incurs a financial penalty, and the iSC guarantees reimbursement to the *redeemer*; we discuss this in Sections 4.3.3 and 4.3.4.

We note that to correctly verify inclusion proofs, the chainRelay must be up to date with the block headers of B. As XCLAIM makes timing assumptions on inclusion proof submission during

Redeem, we must define an upper bound Δ_{relay} for the delay between generation of a block containing transaction TX^B on B and the submission of (i) the block header and (ii) the inclusion proof for TX^B to the iSC via the chainRelay. Hence, we define $\Delta_{relay} = \Delta^B + \Delta_{submit} + 2\Delta^I$, where Δ_{submit} is the delay before a transaction submitting a B block header to the chainRelay is broadcast. If batched submission of n block headers within a single transaction on I is possible, the upper bound for the delay is increased to $\Delta_{submit} + n(\Delta^B + 2\Delta^I)$. This also applies to compact proofing techniques, e.g. NiPoPoWs [KMZ17] or FlyClient [BKLZ20].

Security Arguments for Liveness

The chainRelay makes the *Issue* protocol non-interactive: instead of trusting the *vault* to confirm the lock of b, the iSC accepts a transaction inclusion proof provided by the *creator*. To prevent the *creator* from executing *Issue*, an adversary hence must control all funds i on I and/or prevent inclusion of transactions in B or I. As *Transfer* and *Swap* only require interaction with the iSC, to interfere, an adversary must modify the behavior of the iSC or prevent transaction inclusion in I. This, however, is not possible under the assumptions of the blockchain and threat models. Hence, XCLAIM achieves **Liveness**.

Security Arguments for Consistency

By construction, the iSC only issues $i(b)$ if the provided transaction inclusion proof shows that the correct amount on b was locked on B, i.e., $|b| = |i(b)|$. From the blockchain and threat models, we know an adversary cannot tamper with the iSC. Hence, XCLAIM achieves **Consistency.**

We note that for the *vault* to have a realistic time window to provide a proof, we must consider the security parameters for B and I, as well as the block generation rates when parameterizing Δ_{redeem}, i.e., it must hold that $\Delta_{redeem} > \Delta^B + \Delta_{relay}$.

4.3.3 Tribunal: Incentives via Collateralization

We next modify CentralClaim by introducing *collateral* as a means to incentivize honest behavior in Xclaim and impose punishment on misbehaving parties through the iSC. We refer to this component of the iSC as the tribunal (cf. Figure 4.2). Specifically, we modify CentralClaim by requiring the *vault* to lock up units of i as collateral when registering with the iSC, which we denote as i_{col}. If the *vault* fails to prove correct execution of the *Redeem* protocol, the collateral is automatically used by the iSC to compensate the *redeemer* and to pay an additional punishment fee.

For collateralization of the *vault* to be effective in terms of maintaining incentives to behave honestly, the collateral must be at least equal to the funds locked on backing chain B. One challenge faced by this approach in Xclaim is that the *vault*'s collateral is locked in currency i, while the value it is balanced against is measured in currency b. To this end, we must ensure $i_{col} \geq b_{lock} \cdot \varepsilon(i, b)$ holds, where b_{lock} refers to the units of b locked with the *vault* on B and $\varepsilon(i, b)$ is the exchange rate provided by oracle $\mathcal{O}$. For ease of explanation, at this point, we assume the exchange rate $\varepsilon(i, b)$ is constant. We discuss challenges of non-constant exchange rates and mitigation thereof in Section 4.3.4.

To ensure **Redeemability**, users must only initiate the *Issue* protocol, if sufficient collateral is provided by the *vault* in the iSC. However, the naive *Issue* protocol of CentralClaim exhibits vulnerabilities to *race conditions*: (i) the *vault* can attempt to withdraw collateral before the *creator* can finalize the issuing process, i.e., provide the transaction inclusion proof to the chainRelay, and (ii) multiple *creators* can attempt to simultaneously issue for the same amount of the *vault*'s collateral, triggering a race where the loser's locked funds b_{lock} are not secured by collateral. We present mitigations for the above attacks in Xclaim:

- **Deferred Collateral Withdrawal**. The *vault* may exploit race conditions due to network latency, delays Δ^B and Δ^I or DoS attacks against the *creator* to attempt collateral withdrawal *after* a lock on B is executed, committing unpunished theft. We derive a simple announce-delay-withdraw scheme to prevent such attacks. Specifically, we require

the *vault* to announce collateral withdrawal publicly via the iSC. The iSC allows users to finalize (in theory also to initiate new) issue processes within a delay $\Delta_{withdraw}$, after which the *vault* may withdraw the remaining unused collateral. Thereby, the lower bound for $\Delta_{withdraw}$ is the upper bound on transaction inclusion proofs Δ_{relay} defined in Section 4.3.2, i.e., $\Delta_{withdraw} > \Delta_{relay}$.

- **Collateralized Issue Commitments**. To prevent multiple *creators* from concurrently locking funds b using the same amount of the *vault*'s collateral, we introduce a registration step to the setup phase of the *Issue* protocol. Specifically, a *creator* must register an issue request for $i(b)$ with the iSC, which temporarily locks the corresponding amount of the *vault*'s collateral. Within the following delay $\Delta_{commit} > \Delta^B + \Delta_{relay}$ the *creator* can then safely execute the remaining steps of the *Issue* protocol. The iSC therefore only accepts pre-registered issuing attempts. To avoid *griefing* attacks by malicious *creators*, i.e., continuous locks of the *vault*'s collateral, we require the *creator* to commit to issuing by providing collateral herself. The latter is used to reimburse the *vault* in case of failure. We note that multiple collateralized commitments can be created in parallel, of which only a single one will be accepted by the iSC, on a first-come-first-served basis. In this worst-case scenario, the losses faced by *creators* are hereby limited to a transaction fee on I.

Security Arguments for Redeemability under constant $\varepsilon(i, b)$

By introducing collateralization in XCLAIM we ensure that an economically rational *vault* has no incentive to misbehave. Specifically, by construction, the iSC only accept issue requests if collateral $i_{col} \geq b \cdot \varepsilon(i, b)$ is locked by the *vault*. During the *Redeem* protocol, the *vault* is required to include a transaction in B, sending b to the *redeemer* such that $|b| = |i(b)|$ and provide an inclusion proof to the iSC. If the *vault* misbehaves, it will lose collateral i_{col}, which the iSC uses to reimburse the *redeemer*, and miss out on fees for honest behavior. Meaning, a *vault* gains negative utility from misbehaving and does not execute its equilibrium strategy.

Deferred collateral withdrawal and collateralized issue commitments, therefore, prevent the

vault from exploiting network-related race conditions to defraud users. It is also easy to see that collusion of malicious *vault* and *redeemer* yields no benefits, as issuing and redeeming in XCLAIM is a zero-sum game. In fact, transaction fees on B and I lead to negative utility in such scenarios. Hence, under the economically rational adversaries as per our threat model, XCLAIM achieves **Redeemability** under constant exchange rates.

4.3.4 Mitigating Exchange Rate Fluctuations

Until now, we have assumed that both the exchange rate $\varepsilon(i, b)$ and the collateral i_{col} provided by the *vault* remain unchanged. However, real-world observations show that the exchange rate $\varepsilon(i, b)$ between the two cryptocurrencies may be susceptible to strong fluctuations. To ensure **Redeemability** under *non-constant* exchange rates, we hence (i) over-collateralize the *vault*, (ii) enable adjustment of the *vault*'s locked collateral and (iii) introduce automatic liquidation to prevent financial loss in case of extreme devaluation of i.

Over-collateralization helps mitigate failures due to sudden drops of $\varepsilon(i, b)$. We over-collateralize the *vault* by a factor $r_{col} \in \mathbb{R}_{\geq 1}$, creating a buffer to account for possible exchange rate fluctuations. For secure operation, the following must hold for the lifecycle of XCLAIM:

$$i_{col} \geq b_{lock} \cdot (r_{col} \cdot \varepsilon(i, b)) \geq b_{lock} \tag{4.1}$$

As a result, the over-collateralization factor r_{col} becomes a security parameter in XCLAIM. The combination of r_{col} with the exchange rate $\varepsilon(i, b)$ then defines how many units of the backing cryptocurrency b a *creator* can safely lock with the *vault*, i.e., the maximum amount of safely issuable $i(b)$:

$$\max(i(b)) = \frac{i_{col}}{r_{col} \cdot \varepsilon(i, b)} \tag{4.2}$$

For clarity, we denote *blocked* collateral, i.e., collateral already used to securely issue $i(b)$, as $i^{-}_{col} = i(b) \cdot r_{col} \cdot \varepsilon(i, b)$ and *free* collateral as $i^{+}_{col} = i_{col} - i^{-}_{col}$.

While over-collateralization helps mitigate extreme fluctuations in the short term, it may be

insufficient to securely handle long-term issuing. To this end, we enable the adjustment of the *vault*'s collateral and introduce the notion of automatic liquidation of $i(b)$ by the iSC. We derive a simple multi-stage system for collateral i_{col}. The latter defines the behavior of the iSC, based on the observed collateral rate $r^*_{col} = \frac{i^-_{col}+i^+_{col}}{b_{lock}\cdot\varepsilon(i,b)}$ and the (parameterized) ideal rate r_{col}. Specifically, we introduce thresholds $r_{col} > r^{liq}_{col} > 1.0$. For ease of explanation, we assume an exemplary collateral rate $r_{col} = 2.0$. We define the multi-stage system for collateral as follows:

- **Secure Operation** : The *vault* has locked more collateral than necessary to ensure **Redeemability** in XCLAIM, i.e., new $i(b)$ can be issued correctly. Similarly, the available free collateral i^+_{col} can be withdrawn by the *vault*, as long as $r^*_{col} \geq r_{col}$ holds.

- **Buffered Collateral**: The collateral rate r^*_{col} has dropped below the ideal rate r_{col}, however, there is sufficient buffer to ensure secure operation of XCLAIM. However, as defined in Eq. 4.2, no new $i(b)$ can be issued.

- **Liquidation**: The collateral rate is critically close to the lower bound of 1.0 (e.g. $r^{liq}_{col} = 1.05$). If the *vault* does not re-balance r^*_{col} by increasing i_{col}, the iSC automatically initiates *Redeem* for all existing $i(b)$. The remaining collateral buffer $r^*_{col} - 1.0 > \varepsilon(i,b)$ is thereby used to cover transaction fees. This measure is necessary to prevent users from facing financial loss, should r^*_{col} drop below 1.0.

Security Arguments for Redeemability under non-constant $\varepsilon(i,b)$

Through over-collateralization of the *vault*, we use a buffer to tolerate sudden exchange rate drops. As the *vault* can update collateral, it can, in the optimistic case, maintain the secure operation of XCLAIM even if the buffer is depleted. Should the latter fail, the iSC ensures users do not face financial losses via automatic liquidation. Specifically, an economically rational *vault* will only misbehave if $i_{col} < b\cdot\varepsilon(i,b)$. By construction, the iSC automatically reimburses i_{col} to a *redeemer* before $i_{col} < b\cdot\varepsilon(i,b)$. Hence, misbehaving only becomes the equilibrium strategy of the *vault*, if it can alter the behavior of the iSC. As this is not possible under the assumptions

of the blockchain and threat model, it follows that XCLAIM achieves **Redeemability** under non-constant exchange rates $\varepsilon(i, b)$.

Note: from $\Delta_{\Omega(\varepsilon(i,b))} < \Delta^I$ it follows the *redeemer* can either initiate the *Redeem* protocol or, in case of automatic liquidation, withdraw i from the iSC before $\varepsilon(i, b) < \Omega(\varepsilon(i, b))$.

It is worth noting that while private information of a *vault* may suggest that the value of b will exceed the value of the liquidated collateral i_{col} in the future, this does not affect the security of XCLAIM. If *vault* decides to steal b in exchange for the collateral i_{col} while the value of the collateral exceeds the value of b, the *redeemer* can use the received i_{col} to acquire lost b on a secondary market, e.g. a centralized or decentralized exchange, avoiding financial damage.

4.3.5 Multi-*vault* System: Removing Single Points of Failure

Until now, we have assumed a single *vault*. However, the design of XCLAIM allows it to be easily extended to a multi-*vault* system. Hence, we allow *any* user to become a *vault* by registering with the iSC and providing collateral. The list of *vaults* is maintained in a public registry in the iSC. By allowing both *creators* and *redeemers* to freely choose which *vault* they wish to use for issuing and redeeming, we create a free market driven by charged fees and the observed collateral rate $r_{col}*$ of each *vault*. The availability of multiple *vaults* further allows a *redeemer*, upon a failed *Redeem* caused by a *vault*, to choose between: (i) being reimbursed from the slashed collateral i_{col} or (ii) retrying the *Redeem* using a different *vault*.

One challenge that arises from a multi-*vault* system is ensuring correct automatic liquidation. Deterioration of collateral of a single *vault* does not affect the entire system, but only the corresponding fraction of issued $i(b)$. In a first step, the iSC offers *beneficial* liquidation, i.e., redemption of $i(b)$ against the corresponding amount of b_{lock} and an additional small premium in i, deducted from the *vault*'s available collateral $(r^*_{col} - 1.0)$. Should insufficient users wish to execute *Redeem*, the iSC, as a final fallback, equally distributes the liquidation among all users of XCLAIM. Note: tracing CBAs back to the *vault* they were issued makes CBAs non-fungible, which is contrary to the desired functional properties.

Arguments for Scale-Out

By construction, any user can register as a *vault* with the iSC on I by locking collateral i_{col}, i.e., the set of *vaults* can change dynamically and is not pre-defined. It is easy to see any user can hence increase the total amount of safely issuable CbAs, $\max(i(b))$. We note that to prevent registration of new *vaults*, an adversary must: (i) control all funds i on I and/or (ii) prevent inclusion of transactions in I. Both scenarios are not possible under the assumptions of the blockchain and threat models. Hence, under secure operation of B and I, XClaim achieves **Scale-Out**.

4.3.6 Atomic *vault* Replacement

Until now, we have assumed a *vault* cannot lock funds b_{lock} on B and must remain in XClaim until the latter is fully redeemed by users. In a real-world scenario, the *vault* may wish to leave and transfer their role to another party earlier, or move funds to a different account on B for practical purposes. To this end, we describe *Replace*, a non-interactive atomic cross-chain swap (ACCS) protocol based on cross-chain state verification, which allows *vaults* to move funds on B without being punished by the iSC.

Protocol: Replace. *vault* migrates locked funds b to *vault*$'$, who replaces *vault*'s collateral in the iSC.

1. *Setup.* The *vault* submits a *replacement* request to the iSC and locks up collateral $i_{col}^{replace}$, sufficient to cover costs of a transaction on I.

2. *Lock.* A new candidate *vault*$'$ can lock the corresponding amount of collateral for a pre-defined period $\Delta_{replace}$ with the iSC on I, such that $|i_{col}^{vault}| = |i_{col}^{vault'}|$, providing their public key on backing chain B.

3. *Migrate.* Within $\Delta_{replace} > \Delta_{relay}$, the still active *vault* must migrate the locked b_{lock} to the public key of *vault*$'$ on B and submit the corresponding transaction inclusion proofs to the chainRelay on I.

4. *Release.* The chainRelay verifies the migration was executed correctly on B and the iSC releases the old *vault*'s collateral, i.e., both i_{col}^{vault} and $i_{col}^{replace}$. If the *vault* does not execute the migration on B within $\Delta_{replace}$, the iSC releases the new candidate's collateral, while using $i_{col}^{replace}$ to reimburse wasted transaction fees.

We can further use the *Replace* protocol to enable re-balancing of collateral among *vaults*. Assume a *vault*'s observed collateralization rate r_{col}^{*} has dropped significantly below ideal rate r_{col}. To prevent automatic liquidation, the *vault* must contribute additional collateral to the iSC on I. Alternatively, the *vault* can execute *Replace* to migrate a fraction of total locked coins b_{lock} to $vault'$, so as to re-balance her collateralization rate r_{col}^{*}. Should no single *vault* have sufficient free collateral to complete the re-balancing, the *Replace* protocol can be executed iteratively with multiple *vaults*, e.g. until $r_{col}^{*} \geq r_{col}$ holds. This procedure is specifically useful if a *vault* cannot provide additional collateral due to insufficient funds on I but intends to ensure secure operation of iSC.

4.4 Formal Protocol Specification

This section presents a formal specification XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols, as well as the requirements imposed on the backing and issuing blockchains.

Extended Notation

We differentiate between *state changing* and *non-state changing* operations in XCLAIM; state changing operations result in new transactions (T) in the underlying blockchain, while non-state changing operations, such as verifying operations, are "read-only", returning boolean values ($\top|\bot$). We use TX_{id}^{B} to refer to a transaction included in the ledger L_B of chain B with an identifier id ($\text{TX}_{id}^{B} \in \mathsf{L}_B$) and TX_{id}^{I} for transactions on I respectively ($\text{TX}_{id}^{I} \in \mathsf{L}_I$). The execution of an operation on input in that produces an output out is denoted as $\mathsf{operation}(in) \rightarrow out$. To indicate that an operation is executed by a user $user$, we write $user.\mathsf{operation}$. We identify

a user *user* in XCLAIM by her public key pk_u^X (with corresponding private key sk_u^X), where X can be either blockchain B or I. For readability, we often write $user^X$ when referring to pk_u^X. We use *cond* to refer to locking and unlocking conditions for funds on both B and I, e.g. a condition for a transaction may be the digital signature of user u on chain X with private key sk_u^X, denoted as σ_{user}^X. A summary of symbols is provided in Section 4.9.

We parameterize XCLAIM, using: (i) the blockchain security parameters k^B and k^I; (ii) block generation rates τ^B and τ^I; (iii) collateral rate r_{col} and the automatic liquidation threshold r_{col}^{liq}; and (iv) the delays Δ_{redeem}, $\Delta_{withdraw}$, Δ_{commit}, Δ_{swap} used in the *Issue*, *Redeem* and *Swap* protocols. The algorithms specifying XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols, as well as the necessary operations exhibited by XCLAIM on the underlying blockchains B and I are provided in Section 4.9.

4.4.1 XCLAIM Operations

First, we overview the operations exhibited by XCLAIM on the backing blockchain B and issuing blockchain I.

Backing Blockchain

For the backing blockchain B, operations are executed by the *creator* and the *vault*. We differentiate between the two:

Operations performed by the *creator*:

- $\mathsf{lockB}(b, cond) \rightarrow \text{TX}_{lock}^B$ which locks coins b on chain B under conditions *cond*.

Operations performed by the *vault*:

- $\mathsf{verifyIOp}(\mathsf{operation}, \text{TX}_{op}^I[, \Delta_{op}]) \rightarrow \top|\bot$ which verifies that $\mathsf{operation}$ was executed on I, i.e., that TX_{op}^I is securely included in I according to k^I and within optional delay Δ_{op} as per XCLAIM parameters.

- $\mathsf{redeemB}(b, user^B) \rightarrow \text{TX}^B_{redeem}$ that releases locked coins b to $user^B$.

- $\mathsf{transferB}(b, user^B) \rightarrow \text{TX}^B_{transfer}$ which transfers ownership of b to user $user^B$.

Issuing Blockchain

For the issuing blockchain I, operations are executed by the iSC:

- $\mathsf{lock}(x, cond) \rightarrow \text{TX}^I_{lock}$ which locks x under conditions $cond$ on I, where x can be $i(b)$ or i.

- $\mathsf{release}(x, user^I) \rightarrow \text{TX}^I_{release}$ that releases asset or coin x to $user^I$, where x can be $i(b)$ or i.

- $\mathsf{slash}(x, user^I, user^I_*) \rightarrow \text{TX}^I_{slash}$ that destroys or slashes collateral funds x of $user^I$ and reimburses them to $user^I_*$, where x can be $i(b)$ or i.

- $\mathsf{commit}(b, user^B, vault, i) \rightarrow \text{TX}^I_{commit}$ which commits $user^B$ to calling $\mathsf{lockB}(b, \sigma^B_{vault})$ within Δ_{commit}. Locks i as collateral conditioned on the iSC's signature via $\mathsf{lock}(i, \sigma^I_{SC})$.

- $\mathsf{verifyBOp}(\mathsf{operation}, \text{TX}^B_{op}[, \Delta_{op}]) \rightarrow \top|\bot$ which verifies operation was executed on B, i.e., securely included in B via TX^B_{op} according to k^B and within optional delay Δ_{op} as per XCLAIM parameters.

- $\mathsf{issue}(b, user^I) \rightarrow \text{TX}^I_{issue}$ which creates and allocates $i(b)$ to $user^I$, such that $|i(b)| = |b|$.

- $\mathsf{transfer}(x, user^I) \rightarrow \text{TX}^I_{transfer}$ which transfers ownership of x to user $user^I$, where x can be $i(b)$ or i.

- $\mathsf{swap}(x, user^I_x, y, user^I_y) \rightarrow \text{TX}^I_{swap}$ which transfers ownership of x to $user^I_y$ and y to user $user^I_x$ atomically; x and y can be $i(b)$ or i.

- $\mathsf{burn}(i(b)) \rightarrow \text{TX}^I_{burn}$ that destroys $i(b)$.

4.4.2 XCLAIM Protocols

In the following, we present the algorithms specifying XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols for cryptocurrency-backed assets.

Algorithm 5 XCLAIM *Issue* protocol

Result: *creator* locks units of b with the *vault* on B to create $i(b)$ on I

$vault.\mathsf{lock}(i_{col})$

$creator.\mathsf{commit}(b_{lock}, \mathsf{pk}^B_{creator}, \mathsf{pk}^I_{vault}, i^{commit}_{col})$

$creator.\mathsf{lockB}(b_{lock}, \sigma_{vault})$ /* $\rightarrow \mathrm{TX}^B_{lock}$ */

creator submits TX^B_{lock} to iSC calling verifyBOp

if $\mathsf{iSC.verifyBOp}(\mathsf{lockB}, \mathrm{TX}^B_{lock}, \Delta_{commit}) = \top$ **then**

  $\mathsf{iSC.issue}(b_{lock}, \mathsf{pk}^I_{creator})$

  $\mathsf{iSC.release}(i^{commit}_{col}, \mathsf{pk}^I_{creator})$

else

  $\mathsf{iSC.slash}(i^{commit}_{col}, \mathsf{pk}^I_{creator}, \mathsf{pk}^I_{vault})$

end

Algorithm 6 XCLAIM *Swap* protocol

Result: *sender* atomically swaps $i(b)$ against *receiver*'s i on I

$sender.\mathsf{lock}(i(b), \sigma^I_{SC})$

$receiver.\mathsf{lock}(i, \sigma^I_{SC})$ /* $\rightarrow \mathrm{TX}^I_{lock}$ */

if $\mathsf{iSC.verifyIOp}(\mathsf{lock}, \mathrm{TX}^I_{lock}, \Delta_{swap}) = \top$ **then**

  $\mathsf{iSC.swap}(i(b), \mathsf{pk}^I_{receiver}, i, \mathsf{pk}^I_{sender})$

else

  $\mathsf{iSC.release}(i(b), \mathsf{pk}^I_{sender})$

end

Algorithm 7 XCLAIM *Transfer* protocol

Result: *sender* transfers $i(b)$ to Dave *receiver* on I

sender calls $\mathsf{iSC.transfer}(i(b), \mathsf{pk}^I_{receiver})$

Implicit: $vault.\mathsf{verifyIOp}(\mathsf{transfer}, \mathrm{TX}^I_{transfer})$

Algorithm 8 XCLAIM *Redeem* protocol

Result: *redeemer* locks $i(b)$ with the iSC on I to receive b from the *vault* on B; $i(b)$ is then destroyed.

redeemer calls $\mathsf{iSC.lock}(i(b), \mathsf{pk}^B_{redeemer})$ /* $\rightarrow \text{TX}^I_{lock}$ */

iSC signals $|b| = |i(b)|$ is to be released to *redeemer* on B

if $vault.\mathsf{verifyIOp}(\mathsf{lock}, \text{TX}^I_{lock}) = \top$ **then**

  $vault.\mathsf{redeemB}(b_{lock}, \mathsf{pk}^B_{redeemer})$ /* $\rightarrow \text{TX}^B_{redeem}$ */

  vault submits TX^B_{redeem} to iSC calling verifyBOp

  if $\mathsf{iSC.verifyBOp}(\mathsf{redeemB}, \text{TX}^B_{redeem}, \Delta_{redeem}) = \top$ **then**

    $\mathsf{iSC.release}(i_{col}, \mathsf{pk}^I_{vault})$

  else

    $\mathsf{slash}(i_{col}, \mathsf{pk}^I_{vault}, \mathsf{pk}^I_{redeemer})$

  end

  $\mathsf{iSC.burn}(i(b))$

end

4.4.3 Blockchain Requirements for Implementing XCLAIM

Using the system operations performed by XCLAIM as defined in Section 4.4.1, we derive the requirements for the underlying blockchains B and I. We summarize our findings in Table 4.1 and provide examples for backing and issuing chains currently supported by XCLAIM. We note that neither B nor I require a Turing-complete instruction set.

Backing Blockchain (B)

On the backing blockchain B we need to lock and redeem funds based on conditions *cond*, i.e., a user's digital signature. *Boolean* operations are required to verify *cond* are either true or false. Moreover, conditions for locking and redeeming from different users require (i) a *stack* to store intermediary values, (ii) *read* and *write* operations for the current stack, and (iii) public-key encryption and signature verification. *Flow control* operations do not have to be available

to users and can be expressed by stack states (empty / not empty) [Bit21b]. In addition, while public-key encryption and signatures can be implemented using basic *arithmetic* and *bitwise* operations, both script complexity and execution cost can be reduced if *cryptographic* operations, including hash and signature verification functions, are supported by the scripting language as dedicated operations. Finally, B requires a method to store data, necessary to e.g. include the target public key of the *creator* for issuing on I.

Issuing Blockchain (I)

To support issuing of CBAs in a smart contract, the issuing chain I requires a method to create custom assets, which are part of the consensus protocol. This can be realized by *permanent storage*, i.e., *storage read* and *storage write* operations. In accordance to our definitions (cf. Section 4.1.1), the CBA attributes, *issuing chain*, *backing chain*, and *asset value* are represented as integers; integer balances are assigned to the *asset owner*'s public keys (or digests thereof). Modification of asset balances can be realized using *arithmetic* operations on integers, and the authorization of changes via *boolean* operations.

For transaction verification, the chainRelay requires (i) *permanent storage* to store block header, transaction, and proof data and (ii) *arithmetic*, *bitwise*, and *boolean* operations for proof verification. Verifying Merkle tree inclusion requires traversing the data structure (both of block headers and transaction lists), i.e., I must support *finite loops or recursion*. If I supports the same (or super-) set of *cryptographic* operations used on B (specifically hash functions and signature schemes), the verification may be executed at a lesser cost. Next, I requires more complex conditional locks than B, i.e., *flow control* must be supported (in addition to *arithmetic* and *boolean* operations). Since *Issue* and *Redeem* require checking delays (blocks or based on timestamps), I must allow access to XCLAIM parameters via *global parameters* (integers).

Table 4.1: Required operations on backing (B) and issuing (I) chains including application candidates.

Requirements	Operations: Arithmetic	Boolean	Bitwise	Cryptographic†	Flow control	Local stack	Finite loops or recursion	Permanent storage	Global parameters	Examples for supported blockchains
Backing chain B	✓	✓	(✓‡)	(✓)	✗⋆	✓	✗	✓	✗	Bitcoin[Nak08], ZCash[BSCG+14], Namecoin[Nam22], Litecoin[Lit21], Ethereum[But14]
Issuing chain I	✓	✓	✓	(✓)	✓	✓	✓	✓	✓	Ethereum[But14], Polkadot[Woo15], Cardano[Car22], Cosmos[KB15], Solana[Yak18], RSK[Ler15]

† Not strictly required, but reduces script complexity.
‡ Not necessary if native support for cryptographic operations (hash and signature verification) is available.
⋆ Can be represented as stack states (empty / not empty).

Discussion

One of the main challenges overcome by the design of XCLAIM is backward compatibility: any blockchain supporting the minimum requirement of transferring funds between users can act as backing blockchain B. However, there are cases where B may support a similar set of operations to I, i.e., B may also allow the deployment of a smart contract bSC. More specifically, programmatic verification of transaction inclusion proofs via chainRelay components is supported bilaterally, i.e., the verifyIOp operation in *Issue* and *Redeem* can be executed directly by bSC rather than by an intermediate *vault*. The rest of the system remains unchanged. The use of collateralized intermediaries in this scenario, while no longer necessary for secure operation, can act as a performance and cost improvement. Examples are Ethereum and Ethereum Classic, where inclusion proofs can be verified via PeaceRelay [Kyb17].

4.5 Security Analysis

In this section we provide an informal security analysis to supplement the design choices and security propositions presented in Sections 4.2 and 4.3. We discuss attack vectors, potential impacts, and their mitigations.

4.5.1 Chain Relay Poisoning

An adversary may attempt to *poison* the chainRelay with false information regarding blockchain B. Such attacks can be considered equivalent to successful *selfish mining* [ES14] attacks, as the adversary must trigger a chain reorganization according to the underlying consensus rules. Even though in our model we assume $f < n/3$ (or $\alpha < 33\%$), if the assumptions regarding data availability of the chainRelay do not hold (cf. Section 4.3.2), a poisoning attack can be successful well below this threshold. While an adversary cannot prevent the inclusion of transactions in I under our model, the lack of honest block headers submitted to the chainRelay may have alternate reasons, e.g. high cost. To mitigate relay poisoning due to temporary lack of block header data, we can introduce a *maturity period* $\Delta_{maturity}$ for newly generated CBAs [BCD+14], similar to that of newly minted coins in PoW blockchains. If a (correct) conflicting chain is submitted within $\Delta_{maturity}$, the pending CBAs are not issued.

4.5.2 Replay Attacks on Inclusion Proofs

Without adequate protection, inclusion proofs for transactions on B can be *replayed* by: (i) the *creator* to trick the iSC into issuing duplicate $i(b)$ and (ii) the *vault* to reuse a single transaction on B to falsely prove multiple redeem requests. A simple and practical mitigation is to introduce unique identifiers for each execution of *Issue* and *Redeem* and require transactions on B submitted to the chainRelay of these protocols to contain the corresponding identifier.

4.5.3 Counterfeiting

A *vault* which receives b_{lock} from a *creator* during *Issue* could use these coins to re-execute *Issue* itself, creating counterfeit $i(b)$, i.e., $|b_{lock}| < |i(b)|$. To this end, the iSC forbids *vaults* to move locked funds b_{lock} received during *Issue*. From **Auditability** we know that any user with read-access to B can detect misbehavior and can submit a transaction inclusion proof to the iSC showing the *vault* spent locked funds b_{lock}. To restore **Consistency**, the iSC slashes

the *vault*'s entire collateral and executes automatic liquidation, following the steps described in Sections 4.3.4 and 4.3.5, yielding negative utility for the *vault*. To allow economically rational *vaults* to move funds on B, XCLAIM utilizes the *Replace* protocol, a non-interactive atomic cross-chain swap (ACCS) mechanism based on cross-chain state verification, as described in Section 4.3.6.

4.5.4 Permanent Blockchain Splits

Permanent chain splits or *hard forks* occur where consensus rules are loosened or conflicting rules are introduced [ZSJ+18], resulting in multiple instances of the same blockchain. Thereby, a mechanism to differentiate between the two resulting chains (*replay protection*) is necessary for secure operation [MHM17].

Backing Chain

If replay protection is provided after a permanent split of B, the chainRelay must be updated to verify the latter for B (or B' respectively). If no replay protection is implemented, the chainRelay will behave according to the protocol rules of B for selecting the "main" chain. For example, it will follow the chain with the most accumulated PoW under Nakamoto consensus.

Issuing Chain

A permanent fork on the issuing blockchain results in two chains I and I' with two instances of the iSC identified by the same public key. To prevent an adversary exploiting this to execute replay attacks, both *creator* and *vault* must be required to include the public key of the iSC (or a digest thereof) in the transactions published on B as part of *Issue* and *Redeem* (in addition to the identifiers introduces in 4.5.2). Next, we identify two possibilities to synchronize $i(b)$ balances on I and I': (i) deploy a chain relay for I on I' and vice-versa to continuously synchronize iSC [Kyb17] and iSC$'$ states or (ii) redeploy the iSC on both chains and require users and *vaults* to re-issue $i(b)$, explicitly selecting I or I'.

4.5.5 Denial-of-Service Attacks

XCLAIM is decentralized by design, thus making denial-of-service (DoS) attacks difficult. Given that any user with access to B and I can become a *vault*, an adversary would have to target all *vaults* simultaneously. Where there are a large number of *vaults*, this attack would be impractical and expensive to perform. Alternatively, an attacker may try to target the iSC. However, performing a DoS attack against the iSC is equivalent to a DoS attack against the entire issuing blockchain or network, which conflicts with our assumptions of a resource-bounded adversary and the security models of B and I. Moreover, should an adversary perform a Sybil attack and register as a large number of *vaults* and ignore service requests to perform a DoS attack, the adversary would be required to lock up a large amount of collateral to be effective. This would lead to the collateral being slashed by the iSC, making this attack expensive and irrational.

4.5.6 Fee Model Security: Sybil Attacks and Extortion

While the exact design of the fee model lies beyond the scope of this work, we outline the following two restrictions, necessary to protect against attacks by malicious *vaults*.

Sybil Attacks

To prevent financial gains from Sybil attacks, where a single adversary creates multiple low collateralized *vaults*, the iSC can enforce (i) a minimum necessary collateral amount and (ii) a fee model based on issued volume, rather than "pay-per-issue". In practice, users can in principle easily filter out low-collateral *vaults*.

Extortion

Without adequate restrictions, *vaults* could set extreme fees for executing *Redeem*, making redeeming of $i(b)$ unfeasible. To this end, the iSC must enforce that either (i) no fees can be

charged for executing *Redeem* or (ii) fees for redeeming must be pre-agreed upon during *Issue*.

4.6 XCLAIM(BTC,ETH) Implementation and Evaluation

We instantiate XCLAIM as XCLAIM (BTC,ETH) to issue Bitcoin-backed tokens on Ethereum. Ethereum's virtual machine provides Turing completeness [Woo17], fulfilling the requirements (Section 4.4.3) for the issuing chain I. Bitcoin, due to the limited operation set of its Script language [Bit21b] only qualifies as a backing blockchain (Section 4.4.3). Both Bitcoin and Ethereum use ECDSA with the secp256k1 Koblitz curve [Kob91, BWQ99], providing native support for the corresponding cryptographic operations. Ethereum further makes the SHA-256 and RIPEMD-160 hash functions, which are used in Bitcoin, available as pre-compiled contracts [Woo17].

We implement the iSC smart contract on Ethereum in Solidity v0.4.24 [Eth22b] in approximately 820 lines of code. On Bitcoin, we use regular P2PKH [Bit20b] transactions. We use the existing Serpent [Eth17] implementation of BTCRelay [Con17a] for the chainRelay component of the iSC. Bitcoin-backed tokens per our implementation are compliant with the ERC20 token standard [VB15], and hence usable in most services on Ethereum, including *decentralized exchanges*. To evaluate the cost and performance of XCLAIM (BTC,ETH), we deploy the iSC on the Ethereum Ropsten test network [Eth22c]. For evaluations, we assume a *vault* is registered with the iSC and has locked in collateral on Ethereum.

4.6.1 Protocol Execution Costs

We define on-chain execution costs measured in USD as the amount of Bitcoin and Ethereum transaction fees required to execute each of the protocols: *Issue*, *Transfer*, *Swap*, *Redeem* and *Replace*. The costs are calculated using exchange rates at the time of implementation[8]. It is

[8]Storage and execution costs are in USD as per exchange rates of 30 Nov. 2018: BTC/USD 3717.38 and ETH/USD 105.71

worth noting that the USD costs are subject to change, in line with fluctuations of the Bitcoin and Ethereum USD exchange rates, as well as network utilization.

In Bitcoin, transaction fees are calculated based on the transaction size, i.e., the number of consumed inputs and generated outputs. To ensure transactions are included in the next generated block without delays, we calculate fees with 40 Satoshis (10^{-8} BTC) per byte [ear22]. In Ethereum, transaction fees are measured in *gas*, depending on both on the transaction size and the cost of executed smart contract operations [Woo17]. We assume a gas cost of 9 Gwei based on the network fees for fast transaction inclusion at the time of implementation (30 November 2018) [Sta22].

We summarize our measurement results in Table 4.2. In our measurements, we refer to the complete process of issuing (*Issue*), executing a trade (*Swap*), and then redeeming (*Redeem*) an arbitrary amount of Bitcoin-backed tokens on Ethereum as a *round*. Our experiments show a full protocol execution round only costs USD 1.15. The main cost factor thereby are Bitcoin transaction fees (53.9%). As such, transferring ownership over units of Bitcoin via XCLAIM Bitcoin-backed tokens on Ethereum costs only USD 0.04, in contrast to USD 0.31 if the transfer is executed natively on Bitcoin (87.1% cheaper). Additional costs are incurred for keeping BTCRelay up to date with the current state of the Bitcoin blockchain, amounting to approximately USD 27 per day (not included in the table). These costs are fixed and shared among all users of XCLAIM; each user's share decreases with higher adoption of XCLAIM. We further note this number constitutes an *upper bound* given current prices: (i) the existing implementation of BTCRelay is non-optimal[9] and (ii) our measurements consider the worst-case scenario where batched block header submissions are not available (cf. Section 4.3.2).

[9]The Serpent language is not actively maintained (last commit on 1 October 2017) and is not optimized to the current version of the EVM, resulting in higher execution costs. More efficient proofing techniques can further reduce costs [KMZ17, BKLZ20, TR19, KGC+18].

[10]Performance is measured in minutes and includes security parameters: 6 confirmations, each 10 minutes for Bitcoin (k^B); 12 confirmations, each 14 seconds for Ethereum (k^I).

Table 4.2: Overview of execution costs[8] and performance[10] for the *Issue*, *Transfer*, *Swap* and *Redeem* protocols in XCLAIM (BTC,ETH).

Protocols	**Transactions**	**Cost (USD)**			**Duration (minutes)**
		Ethereum	**Bitcoin**	**Total**	
Issue	2^{Eth} 1^{Btc}	0.16	0.31	0.47	75.98
Swap	2^{Eth}	0.19		0.19	5.98
Redeem	2^{Eth} 1^{Btc}	0.18	0.31	0.49	75.98
Total	6^{Eth} 2^{Btc}	0.53 (46.1%)	0.62 (53,9%)	1.15	157.94
Transfer	1^{Eth}	0.04		0.04	2.99
Replace †	3^{Eth} 2^{Btc}+	0.13	0.62	0.75	148.97+

†Duration and costs of *Replace* depend on the number of Bitcoin UTXOs which need to be migrated. Provided numbers are lower bounds.

4.6.2 Performance

We evaluate the performance of XCLAIM (BTC,ETH) with respect to the duration of the *Issue*, *Transfer*, *Swap*, *Redeem*, and *Replace* protocols (measurements provided in Table 4.2). Thereby, we adhere to the recommended security parameters regarding transaction confirmations based on our threat model ($\alpha \leq 33\%$): $k^B = k^{BTC} = 6$ and $k^I = k^{ETH} = 12$. Recall, we consider a transaction in the block at position j as securely included when the blockchain tip reaches position h, with $h - j \geq k$. At the time of writing, the average block time in Bitcoin amounts to 10 minutes, and in Ethereum to 14 seconds.

One execution round of issuing, atomically swapping, and redeeming of Bitcoin-backed token on Ethereum requires 2 Bitcoin transactions (1 user transaction and 1 *vault* transaction) and 6 Ethereum transactions (5 user transactions and 1 *vault* transaction). The end-to-end process is securely completed after 158 minutes; a *Swap* only takes 6 minutes, while *Issue* and *Redeem* account for the greater part of the delay due to Bitcoin transaction processing (96,2%). Note: we can use off-chain payment channels [MBKM17, KZF+18] on the issuing chain (Ethereum) to significantly reduce execution cost and make *Swap* real-time (see Section 4.7).

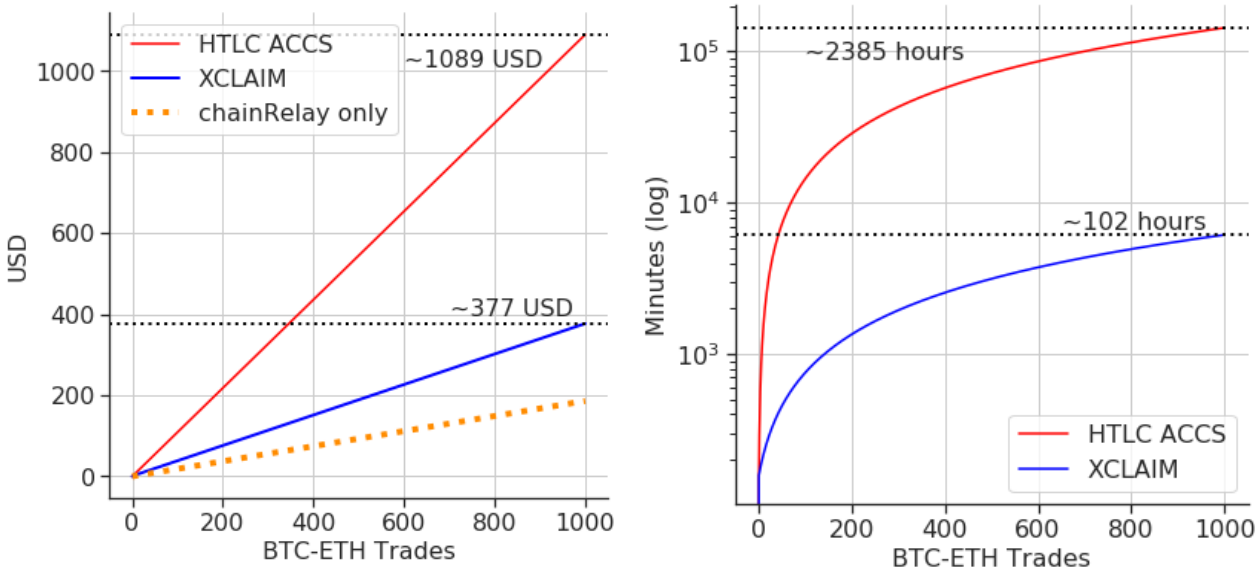


Figure 4.3: Comparison of BTC-ETH atomic swaps via XCLAIM and via HTLC ACCS for 1000 individual swaps. Storage and execution costs (Left) are in USD[8]; performance (Right, logarithmic y-axis) is measured in minutes[10]. We observe XCLAIM is 95.7% faster and 65.4% cheaper for 1000 swaps.

4.6.3 Comparison to HTLC Atomic Swaps

We compare the performance and execution costs of XCLAIM (BTC,ETH) to that of atomic cross-chain swaps (ACCS) based on hashed-timelock contracts (HTLCs) [Tie16, Bit20a, Bit21a]. Both implementations are tested under identical conditions, including fee calculation and security parameters. We visualize the results of our experiments in Figure 4.3.

Each interactive atomic swap requires users to create two transactions on both Bitcoin and Ethereum (4 in total). Including a minimum necessary delay to prevent race conditions, an atomic swap takes approximately 146.5 minutes to execute securely. Note: we omit the additional necessary time to establish out-of-band channels and exchange revocation transactions in ACCS; hence the ACCS measurements are *lower bounds*.

In XCLAIM, after an initial *Issue* process, each additional *Swap* requires only 2 Ethereum transactions. As such, using XCLAIM to atomically exchange BTC against ETH is already more efficient than ACCS after the second swap; for 1000 swaps XCLAIM is 95.7% faster. Similarly, XCLAIM (including BTCRelay fees) outperforms ACCS cost-wise after the second trade; for 1000 trades XCLAIM is 65.4% cheaper than ACCS. Note, that a significant cost

factor in XCLAIM (BTC,ETH) are the non-optimized BTCRelay maintenance fees, which e.g. account for 49.3% of the incurred cost for 1000 swaps.

4.7 Applications

This section provides a brief overview of several new and novel applications enabled by XCLAIM cryptocurrency-backed tokens. These applications illustrate how XCLAIM paves the way for usable and scalable cross-chain communication.

Smart Contract and Application Access for Legacy Blockchains

The primary application of cryptocurrency-backed assets is to provide "legacy" crypto*currencies* like Bitcoin with access to smart contract functionality on blockchain *platforms* like Ethereum. At the time of writing, the Etheruem blockchain exhibits a flourishing ecosystem of decentralized financial applications, enabling users to trade, borrow, lend, invest in structured financial products, and more without trusting centralized intermediaries. The total value locked across various applications on Ethereum is surpasses USD 50 billion [Pul22b], while another estimated 30 billion is spread across platforms such as Solana, Polkadot, Polygon, and Avalanche [Lla22].

The market size of all decentralized financial applications combined is, however, still dwarfed by the market capitalization of Bitcoin (USD 367 billion) [Coi22]. It is hence not surprising that centralized providers have already deployed versions of Bitcoin on Ethereum, termed "wrapped Bitcoin", including wBTC [Kyb19b], hBTC [Tea20] and renBTC [Ren20]. At the time of writing, these centralized Bitcoin-backed assets hold an estimated USD 6 billion worth of Bitcoin - requiring users to give away custody and fully trust the intermediaries.

XCLAIM offers users a decentralized alternative, enabling them to mint Bitcoin-backed assets secured by collateral insurance. When using centralized Bitcoin-backed assets, BTC can be lost due to a single party failing - either due to a security or regulatory incident - leaving users without remedy. In the case of XCLAIM, in the worst-case scenario, are reimbursed in the

collateral asset(s), bearing no or very limited financial damage. End-users are not the only risk bearers in this case: decentralized financial protocols integrating centralized Bitcoin-backed assets exhibit a long-term dependency on the security and reliability of the intermediaries holding Bitcoin in custody. For example, at the time of writing 14% of MakerDAO's DAI stablecoin [Mak14] is backed by centralized wBTC, amounting to USD 3.89 billion.

Cross-Chain Payment Channels

Cryptographic payment channels [Chr15, MBKM17, DEFM17, LEK+17, KZF+18] address the performance limitations of blockchain protocols [GKCC14, CDE+16] by avoiding the need to publish every transaction on the blockchain. Instead, transactions are executed directly between participants *off-chain*, and only the final balances of the participants are published on the blockchain. Despite improving transaction throughput and latency considerably, payment channels cannot execute payments across different blockchains. As such, users are required to set up and instantiate multiple channels, one for every blockchain they wish to participate in. With XCLAIM, however, payment channels deployed on a blockchain capable of issuing XCLAIM CBAs become cross-blockchain compatible automatically; users can transfer CBAs as per normal in a payment channel network, and later redeem those CBAs for native coins. As such, XCLAIM allows existing issuing blockchains, such as Ethereum, to process transactions of any backing cryptocurrency off-chain, without requiring changes to the underlying code. XCLAIM can therefore be used to provide novel contributions to state-of-the-art payment channels.

Temporary Transaction Offloading

The design of XCLAIM allows for both long-term and short-term issuance of CBAs. As such, during temporary periods of high network congestion [Blo22] or transaction fee spikes [Eth21], XCLAIM CBAs can be used to temporarily switch to another blockchain for secure payment processing. Moreover, users can temporarily leverage this technique in XCLAIM to exploit features or benefits that may be present in the issuing chain, but not on the backing chain. For example, Bitcoin users may temporarily switch to Bitcoin-backed tokens on Ethereum to avoid

long transaction processing times, or to leverage more complex transaction scripts and smart contracts in Ethereum. Once such periods have passed, users may exchange their CbA back to coins on the native blockchain securely, without requiring trust.

N-Way and Multi-Party Atomic Swaps

Xclaim is more efficient than atomic cross-chain swaps (ACCS), both in terms of performance and cost (see Section 4.6). In addition, Xclaim can be leveraged to perform more complex and intricate swap constructions. For example, Xclaim enables *N-way atomic swaps*: by extending the *Swap* protocol in Xclaim, users can swap multiple different units of cryptocurrencies for others, e.g. trading units of cryptocurrency x and y against units of w and z atomically within a single swap. Comparing this to ACCS, N-way swaps are impractical, as they would require the creation of N locking and spending transactions while monitoring all involved chains for failures.

In addition, Xclaim also enables *multi-party atomic swaps*. Assume Alice owns coin x and wants to acquire coin y owned by Bob. Bob, however, will only trade y for coin z owned by Carol. Attempting to resolve this situation with ACCS would require Alice to separately swap with Carol and then with Bob, i.e., this process would not be atomic, resulting in 8 transactions on 3 different chains [Bit20a]. However, with Xclaim, if x, y, and z are CbAs, Alice can construct a non-interactive multi-party swap via the iSC, where a single transaction can change the ownership of all 3 coins in iSC, atomically.

4.8 Conclusion

In this chapter, we formalized the notion of cryptocurrency-backed assets. We presented Xclaim, a system for issuing, transferring, swapping and redeeming cryptocurrency-backed assets between blockchains in a financially trustless manner. That is, users can always redeem their CbAs for the backing asset, or claim reimbursement in a collateral currency. This

approach allows us to work around the impossibility result presented in Chapter 3 in practice. We provided a detailed analysis of XCLAIM's design, present a formal protocol specification and identified requirements for the underlying blockchains. XCLAIM is general in design and supports many existing cryptocurrencies without modification. We implemented XCLAIM (BTC,ETH) to construct Bitcoin-backed tokens on Ethereum and evaluated the performance and execution costs; XCLAIM achieves a significant improvement over atomic cross-chain swaps. Finally, we outlined several novel and interesting applications enabled by XCLAIM.

4.9 Symbols and Notations

Table 4.3 provides an overview of symbols and variables used in this Chapter.

Table 4.3: Summary of used symbols and notations used for XCLAIM protocol descriptions.

Symbol	Description
B	Backing blockchain
L_B	Ledger of backing blockchain B
b	Cryptocurrency unit underlying the backing blockchain B
b_{lock}	Units of b locked by a *creator* with a *vault* during *Issue*
I	Issuing blockchain
L_I	Ledger of backing blockchain B
i	Cryptocurrency unit underlying the issuing blockchain I
i_{col}	Units of i locked by a user with the iSC
$i(b)$	Unit of a CBA on I backed by units of b on B
k^X	Security parameter of blockchain X. Defines how many confirmations are necessary for secure transaction inclusion
Δ^X	Maximum delay from transaction broadcast to secure inclusion in blockchain X
$\overline{r_\tau}$	Upper bound for deviations of expected and observed ratio between block generation rates of B and I.
Δ^X_{tx}	Maximum delay from transaction broadcast to receipt by honest consensus participants in X
$\varepsilon(i,b)$	Exchange rate of i to b, given by oracle $\mathcal{O}$
$\min \varepsilon(i,b)$	Lower bound for $\varepsilon(i,b)$ below which econ. rational adversaries are incentivized to misbehave
$\Delta_{\min(\varepsilon)}$	Delay before $\varepsilon(i,b)$ falls below lower bound $\min(\varepsilon(i,b))$
τ^X	Block generation rate of blockchain X
$(\mathsf{pk}^X_u, \mathsf{sk}^X_u)$	Public / private key pair of user u on blockchain X
σ^X_u	Digital signature of user u on chain X, i.e., via sk^X_u
TX^X_{id}	Transaction included in the ledger L_X of blockchain X with identifier id
$\mathsf{operation}(in) \rightarrow out$	Operation taking in as input an generating output out
$\mathsf{operation}$	Short for operation with default inputs and outputs
$cond$	Conditions used to lock coins, e.g. presenting a user's digital signature σ_{user}
Δ_{id}	Time delay introduced in XCLAIM's protocols with identifier id
r_{col}	Ideal (parameterized) collateralization rate of *vaults*
r^*_{col}	Observed collateralization rate of a *vault* in XCLAIM
r^{liq}_{col}	Collateralization threshold for automatic liquidation

Chapter 5

Cross-Chain Light Clients: Problem, Overview, and Efficiency Improvements

With decentralized cryptocurrencies finding more and more applications in industry, the need to deliver digital payments on resource-constrained devices, such as mobile phones, wearable- and Internet-of-things (IoT) devices, is steadily increasing. Even within the cryptocurrency ecosystem, the need for efficient payment verification is becoming imminent. One example are multi-currency wallets, which track the state of multiple cryptocurrencies and hence face high storage and bandwidth requirements. Another are the growing number of cross-cryptocurrency applications [ZHL+19, BCD+14, ZABZ+19]. Here, verification of correct payments happens cross-chain and is often executed by smart contracts, where storage and bandwidth are priced by the byte.

We present TxChain, a novel scheme to improve the efficiency of transaction verification, which improves upon recent work on optimized light clients [KMZ17, BKLZ20]. Thereby, we do not rely on complex cryptographic schemes, but rather leverage the security properties offered by the consensus of decentralized cryptocurrencies – making TxChain compatible with the majority of existing systems.

Blockchain and Light Clients (SPV)

Most widely-used cryptocurrencies, such as Bitcoin and Ethereum, maintain an append-only transaction ledger, the *blockchain*. The blockchain consists of a sequence of blocks chained together via cryptographic hashes. Each block thereby consists of a *block header* and a batch of valid transactions. The block header contains (i) a pointer to the previous block, (ii) a vector commitment over all included transactions, and (iii) additional metadata (e.g. timestamp, version, etc.). Each block is uniquely identified by a hash over its block header.

Vector commitments are employed by users to verify transactions without downloading the entire blockchain. For example, Simplified Payment Verification (SPV) clients in Bitcoin [Nak08] only maintain a copy of the block headers of the longest (valid) proof-of-work chain, where each header includes the root of a Merkle tree that contains the identifiers of the block's transactions as leaves. To verify a transaction is included in a block, an SPV client requires (i) the block header of the block that contains the transaction (to extract the Merkle root), and (ii) the Merkle tree path to the leaf containing the transaction identifier (given the Merkle root). The size of the Merkle path, i.e., the number of hashes, is thereby logarithmic in the number of transactions in the block.

Sublinear Light Clients

Recently, two proposals for so-called *sublinear* light clients were made: *non-interactive proofs of proof-of-work* (NIPoPoW) [KMZ17] and FlyClient [BKLZ20]. In contrast to naïve SPV clients, NIPoPoWs and FlyClient only require to download a fraction of the block headers to verify that a given chain is the valid chain[11]. Both mechanisms sample a subset of block headers *at random*, such that a fake chain produced by an adversary corrupting at most 33% of consensus participants or total computational power will be detected with overwhelming probability – and hence rejected.

NIPoPoWs [KMZ17] sample block headers which exceed the minimum Proof-of-Work target –

[11] The chain with the most accumulated Proof-of-Work in PoW blockchains.

the so-called *superblocks*. Due to the design of PoW, statistically, 1/2 of the generated blocks will exceed the minimum target (level-1 superblocks), 1/4 will exceed the target by a higher number (level-2 superblocks), etc. By only sampling superblocks, the number of block headers NIPoPoW clients need to download is polylogarithmic in the blockchain size. Unless deployed as a non-backward compatible hard fork [ZSJ+18], NIPoPoWs require block headers to contain an additional *interlink* data structure (pointers to previous superblocks) for secure verification of the valid chain.

FlyClient [BKLZ20] samples block headers based on an optimized heuristic, which takes as input a random number, e.g. generated using the latest PoW block hash. Similar to NIPoPoWs, a backward-compatible deployment of FlyClient requires additional data to be stored in block headers: the root of a Merkle Mountain Range commitment – an efficiently-updatable Merkle tree variant that supports logarithmic subtree proofs. The leaves of the MMR contain block hashes of all blocks generated so far.

Both protocols also provide mechanisms to verify that a block header, not sampled as part of the (poly)logarithmic valid chain proof, is indeed part of the valid chain. In NIPoPoWs, this is achieved via so-called *infix* proofs, which link the blocks in question to the sampled superblocks via the so-called "interlink" structure. In FlyClient, this is achieved by a Merkle tree path from the MMR root to the leaf containing the hash of the block in question. Note that additional block inclusion checks are not necessary in naïve SPV clients, since all block headers are already downloaded.

Probabilistic Sampling Dilemma

To the best of our knowledge, all sublinear light client verification protocols only reduce the block-header data submitted to the client, i.e., the protocols provide efficient valid chain proofs. The ultimate goal of light clients, however, is not only to efficiently determine the valid (or "main") chain but to verify the inclusion of transactions in the latter. As such, to prove the inclusion of n transactions in the blockchain, both super-block NIPoPoWs and FlyClient require n block headers and n Merkle tree membership proofs to be submitted to the client – on

top of the valid chain proof. Therefore, for large n, transaction inclusion verification becomes the performance bottleneck of sublinear light clients. Considering the additional data stored in block headers, performance may even be worse than that of naïve SPV clients for high transaction volumes. We term this problem the *Probabilistic Sampling Dilemma*.

Contribution

In summary, we make the following contributions:

- We introduce the Probabilistic Sampling Di-lemma and provide a formal analysis, deriving the expected overhead of payment verification in sublinear light clients (Section 5.2).
- We introduce TxChain as a new technique for compressing transaction inclusion proofs, leveraging the security assumptions of the underlying blockchain (Section 5.3). In particular, to prove the inclusion of n transactions, TxChain creates $\lceil \frac{n}{c} \rceil$ *contingent* (on-chain) transactions, where c is a constant dependent on the block/transaction size of the blockchain. Contingent transactions are only valid if each of the referenced n transactions exists in the blockchain. Proving the inclusion of a contingent transaction hence proves the inclusion of the n referenced transactions. To circumvent block size limitations, we further show how to construct hierarchies of contingent transactions. As a result, to prove the existence of n transactions, TxChain requires a single contingent transaction in the best case ($n \leq c$) and $\lceil \frac{n}{c} + log_c(n) \rceil$ in the worst case ($n > c$).
- We prove TxChain's security and formally analyze its efficiency (Section 5.4). Under high transaction volumes, TxChain reduces the number of downloaded block headers by up to a factor of 977x for FlyClient, and 973x for NIPoPoWs, for $c = 1000$ as in Bitcoin. In terms of transaction inclusion proofs, TxChain achieves an improvement of up to 1190x across all types of light clients.
- We show how TxChain can be deployed (i) in Bitcoin without requiring changes to the underlying protocol and as a soft fork, (ii) and as a hard fork in Ethereum. We

show TxChain's performance improvement when added as an extension to NIPoPoWs, FlyClient and even naïve (linear) SPV clients (Section 5.5.1 and 5.5.2).

- To demonstrate effectiveness in resource-constrained environments, we implement TxChain as a smart contract on Ethereum which efficiently verifies Bitcoin payments (Section 5.5.3)[12].

5.1 Model and Definitions

5.1.1 System Model

Our setting consists of three types of users: *miners*, full *nodes*, and light *clients*.

Miners participate in the consensus protocol that orders the blocks, e.g., in Proof-of-Work blockchains the miners are the users that create the blocks by solving the computationally difficult puzzles. The miners essentially determine which is the valid chain.

Full nodes verify and store a copy of the entire valid (honest) chain[13]. Since a blockchain is a distributed system, the valid chain is the one agreed by the honest miners. To verify that a blockchain is a valid chain, a user can download a copy of the entire chain from one full node (or miner, ideally multiple), and verify all blocks. However, this is quite costly, both in terms of space and computation.

Light clients allow for fast synchronization and transaction verification, under the assumption that the valid chain follows the rules of the network. Specifically, light clients only maintain the following: (i) the necessary data to verify chain validity, i.e., for SPV clients all block headers, while for sublinear light clients a (random) sample of block headers with cardinality polylogarithmic to the length of the valid chain, (ii) for each transaction to-be-verified, the corresponding block header to extract the vector commitment (and optionally a proof that this

[12]Open source implementation available at `https://github.com/txchain/txchain`
[13]Miners are also full nodes, while full nodes are miners with zero "voting power".

block header is indeed part of the valid chain), and inclusion proof, e.g., for Bitcoin the Merkle tree root and path.

Assumptions

We make the usual cryptographic assumptions: all users are computationally bounded; cryptographically-secure communication channels, hash functions, signatures, and encryption schemes exist. Further, we assume the underlying blockchain maintains a distributed transaction ledger that has the properties of persistence and liveness as defined in [GKL15]. Persistence states that once a transaction is included "deep" enough in an honest miner's valid chain it will be included in every honest miners' valid chain in the same block, i.e., the transaction will be "stable".

We assume persistence is parametrized by a "depth" parameter k, meaning that we assume the finality of transactions after k blocks. Liveness states that a transaction that is given as input to all honest miners for a "long" enough period will eventually become stable. Lastly, we note that TxChain does not require any synchrony assumptions since it is a non-interactive proof scheme. Hence, we assume the same network model of the underlying blockchain system. We note, however, that each client is assumed to be connected to at least one honest full node or miner and is hence not prone to eclipse attacks [HKZG15].

Threat Model

We assume a rushing and fully adaptive adversary, meaning that the adversary can reorder the delivery of messages, but cannot modify or drop them, and corrupt users on the fly. However, the proportion of corrupted miners (consensus participants) is bounded by the threshold necessary to ensure safety and liveness for the underlying system [Fuz08]. For Nakamoto consensus, we bound the fraction of computational power $\frac{\alpha}{1+\alpha}$ controlled by the adversary at any moment by $\frac{\alpha}{1+\alpha} \leq 1/3$ [GKL15], where α is a security parameter. In Byzantine fault-tolerant settings, e.g. Proof-of-Stake such as [KRDO17], the fraction of corrupted consensus participants f is

bounded by $f < 1/3$.

Blockchain Notation

We denote a block header, i.e., a block without the included transactions, at position i in chain C as C_i. The genesis block header is, therefore, C_0, while C_h denotes the block header at the tip of the chain, where h is the current "length" (or height) of chain C. Each block header includes (at least) a vector commitment over the set of transactions included in block and the hash of the previous block header in chain C. This hash acts as a reference to the previous block and thus the hash-chain is formed. The vector commitment, on the other hand, is a cryptographic accumulator over an ordered list of transactions or a *position binding* commitment, which can be opened at any position with a proof sublinear in the length of the vector.

We use TX_{id} to refer to a transaction with identifier id. Furthermore, we denote by $\gamma_{(\cdot,\cdot)}$ the inclusion proof of a transaction in a block. Specifically, $\gamma_{(i,id)}$ denotes an inclusion proof of transaction TX_{id} in the block at position i of the chain. If there exists a proof $\gamma_{(i,id)}$, we write $\text{TX}_{id} \in C_i$ Typically, the transaction inclusion proof employs the vector commitment on the block header. We define as $\beta_{(C_i,C)}$ the inclusion proof of the block header C_i in chain C. A naïve block inclusion proof is the entire hash chain C: the hash-chain that includes the block C_i points back correctly to the genesis block C_0 (ground truth). Lastly, we denote as $\pi_{(C,C_h)}$ a chain validity proof: a proof that a chain C at some round ending in a specific block C_h at position h (the tip of the chain) is the valid chain, i.e, the chain agreed by the honest miners. We denote by $|S|$ the cardinality of a set S. Further, we abuse the block header notation C_i to also refer to the block.

5.1.2 Protocol Goals

We use the prover–verifier model from [KMZ17]. In TxChain, the prover (full node) wants to convince the verifier (client) that a set of transactions T are included in the valid chain C. To do so, the prover(s) must provide three types of proofs to the verifier:

1. *Chain validity proof* $\pi_{(\cdot,\cdot)}$: A proof that chain C is the valid chain. Both NIPoPoW and FlyClient provide succinct proofs that the given chain is valid.

2. *Transaction inclusion proofs* Γ: For each transaction in T, a proof of inclusion in a specific block $\gamma_{(\cdot,\cdot)}$.

3. *Block inclusion proofs* $\mathcal{B}$: For each block that contains a transaction of T, a proof of inclusion $\beta_{(\cdot,\cdot)}$ that the block is in the valid chain C. The structure of this proof is specific to the protocol used to verify that chain C is the valid chain.

These proofs are not necessarily distinct, meaning that the data the prover sends to the verifier for all three proofs may overlap. For instance, in an SPV client, the proof of block inclusion requires no additional data since all block headers are stored and verified as part of the verification process of the chain validity. Therefore, if the block inclusion proof is already part of the chain validity proof, we do not send the data twice.

Desired Properties

Our goal is to design a protocol that is *secure* and *efficient*:

- *Security* in TxChain encapsulates the correctness of the protocol, meaning that a verifier only accepts the proofs, i.e., terminates correctly, if the prover is honest and knows the valid chain. In other words, the verifier will terminate correctly if all transactions in T are included in the valid chain C.

- *Efficiency* captures the storage cost of the protocol, i.e., how much data must be sent to the verifier as part of the verification steps (1-3). To evaluate the efficiency of TxChain, we calculate these storage costs and compare them against existing solutions for different sets of transactions (increasing cardinality).

5.2 Probabilistic Sampling: Cure or Curse?

In this section, we highlight the practical challenges of light clients based on probabilistic sampling. We demonstrate that these light clients offer only optimistic performance improvements when the transactions to-be-verified are many, and in the worst case, can perform worse than naïve SPV clients. We term this problem the *Probabilistic Sampling Dilemma*. We first provide an intuition, and then a formal analysis to measure efficiency.

5.2.1 Probabilistic Sampling Dilemma

Chain Validity Proof

Existing sublinear light clients, such as superblock NIPoPoWs [KMZ17] and FlyClient [BKLZ20] use probabilistic sampling to reduce the number of block headers necessary to prove knowledge of the valid chain (chain validity proof). FlyClient relies on a pre-defined heuristic for sampling blocks, while superblock NIPoPoWs sample headers of blocks that exceed the minimum PoW difficulty. Due to the nature of Proof-of-Work (and specifically hash functions modeled as random oracles), the appearance of such blocks is considered random. In both cases, the prover cannot predict upfront which blocks to provide to the verifier as part of the requested chain validity proof. This property yields the probability of the prover defrauding the verifier with respect to the chain validity proof negligible, within our model as described in Section 5.1.

Block Inclusion Proof

In naïve SPV clients, the block inclusion proof is trivial, as the verifier already has the hash-chain for the chain validity proof. However, this is not the case in sublinear light clients that use probabilistic sampling: For a given set of transactions, the prover must provide to the verifier (a) the block headers and block inclusion proofs for the chain validity proof ((poly)logarithmic in cardinality), and (b) for any block including a transaction of the input set that is *not sampled* for the chain validity proof, the corresponding block header, and block inclusion proof.

The reason for the additional block headers is that the probabilistic sample of block headers is independent of the transactions the client wants to verify. Therefore, in addition to the chain validity proof (e.g., NIPoPoW) and the transaction inclusion proof for every transaction, the prover must also persuade the verifier that the block header corresponds to the transaction inclusion proof of each transaction is part of the valid chain. This implies that the cost of the probabilistic NIPoPoWs is also *dependent on the number of transactions to-be-verified and how they are distributed in the blockchain.*

Probabilistic Sampling Dilemma

An additional overhead of probabilistic NIPoPoW is the increase of the block header size – especially if deployed in the blockchain without major modification to the underlying consensus rules. This results in the following phenomenon: the storage and bandwidth cost of both superblock NIPoPoWs and FlyClient can exceed that of naïve SPV clients for high transaction volumes (as shown in the experimental evaluations in Section 5.5.1). In particular, in probabilistic sampling clients, the cost is proportional to the number of different block headers (and block inclusion proofs) that are given to the verifier, multiplied by the block header size. If transactions are distributed across many different blocks of the chain, which are *not sampled* in the chain validity proof, the cost, i.e., the additive data for the three proofs (c.f. Section 5.1.2) sent to the verifier / light client, increases.

As a result, a dilemma arises for clients with constrained resources: Clients can either (a) anticipate a high transaction volume and use a naïve SPV client, accepting a higher cost for chain validity proofs, or (b) rely on a probabilistic sampling (NIPoPoWs, FlyClient), saving costs on downloaded block headers under low transaction volumes, but under high transaction volumes end up with overall higher storage and bandwidth costs. We call this the *Probabilistic Sampling Dilemma.*

5.2.2 Analysis

In this section, we show that given a set of transactions to-be-verified T, the cost of probabilistic sampling light clients grows proportionally to the number of transactions $n = |T|$ and sublinear to the length of the chain. As such, when the number of transactions is large, the costs of the protocol is dominated by the cost of the block inclusion proofs, instead of the chain validity proof.

To that end, suppose $C_1, \ldots, C_\sigma$ is the set of blocks sampled for the chain validity proof. The selected set is expressed via a random variable X which follows the probability distribution defined in the light client protocol – e.g. uniformly-random distribution with respect to the length of the chain in FlyClient. This means, that $X_i = 1$ if the block header C_i is chosen to be part of the chain validity proof. Now, suppose σ is the size of the probabilistic sample and h the length of the valid chain, then if X follows a discrete uniform distribution, it holds that $Pr[X_i = 1] = \frac{\sigma}{h}$, for all $i \in \{0, 1, \ldots, h-1\}$. As mentioned in Section 5.2, we assume the prover cannot influence or bias this random variable for security reasons.

On the other hand, we define the discrete random variable $Y_{i,j} = 1$ if transaction $\text{TX}_j \in T$ is included in block C_i. For the purpose of our analysis, we assume $Y_{i,j}$ follows a discrete uniform distribution on the length of the chain h as well. Thus, $Pr[Y_{i,j} = 1] = \frac{1}{h}$, for all $i \in \{0, 1, \ldots, h-1\}$ and $j \in \{1, 2, \ldots, n\}$. We further define the discrete random variable Y_i to express if a block contains at least one of the transactions in T; $Y_i = 1$ if for any $j \in \{1, 2, \ldots, n\}$, $Y_{i,j} = 1$. Each trial is independent as a transaction's inclusion in a block has no influence on which block will contain another transaction (for block size large enough) Therefore, $Pr[Y_i = 1] = 1 - Pr[Y_{i,1} = 0] \cdot Pr[Y_{i,2} = 0] \ldots Pr[Y_{i,n} = 0] = 1 - \left(1 - \frac{1}{h}\right)^n$.

For every block that includes at least one transaction from T, the prover must provide to the verifier the block header and a block inclusion proof, even if this block is not sampled for the chain validity proof. To determine the overhead on the cost, we have to count the number of blocks that include at least one transaction and are not sampled for the chain validity proof. To that end, we define $Z_i = 1$ if $Y_i = 1 \wedge X_i = 0$. Since Y_i and X_i are independent random

variables,

$$Pr[Z_i = 1] = Pr[Y_i = 1] \cdot Pr[X_i = 0] = \big(1 - (1 - \frac{1}{h})^n\big) \cdot \big(1 - \frac{\sigma}{h}\big).$$

Thus, the expected number of additional block headers are

$$\mathbb{E}(Z) = \mathbb{E}\Big(\sum_{i=0}^{h-1} Z_i\Big) = h \cdot Pr[Z_i = 1] = (h - \sigma) \cdot \Big(1 - (1 - \frac{1}{h})^n\Big) \geq \Big(1 - \frac{\sigma}{h}\Big) \cdot n$$

We observe that the smaller the sample for the chain validity proof, the larger the expected number of additional transactions. Furthermore, we notice that for a given chain length and sample size, the expected number of additional blocks grows with the number of transactions to be verified.

5.3 TxChain Design

In this section we present the design of TxChain. We first define the concept of contingent transactions and then present how this mechanism can be used to circumvent the Probabilistic Sampling Dilemma.

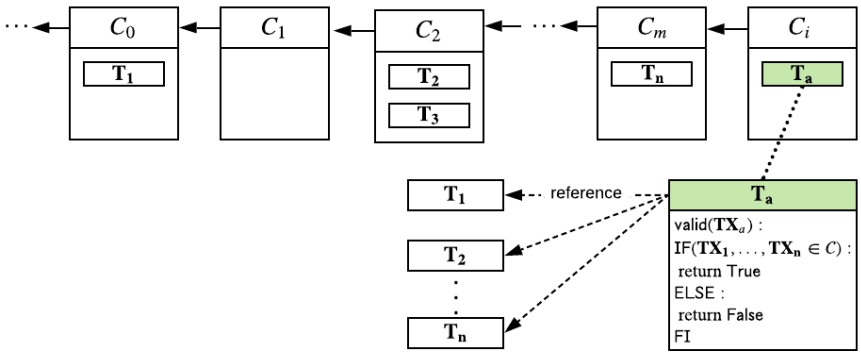


Figure 5.1: Visualization of TxChain: a contingent transaction TX_a is only valid and can hence be included in the valid chain C at index i if all referenced transactions $\text{TX}_1, \ldots, \text{TX}_n$ are included in C, and hence are valid. The inclusion proof $\gamma_{(i,a)}$ for TX_a is hence also proves inclusion of $\text{TX}_1, \ldots, \text{TX}_n$.

5.3.1 Contingent Transactions

Smart contracts in blockchains allow us to define under which conditions a transaction can be included in the underlying ledger, i.e., specify when the transaction becomes valid under the blockchain's consensus rules. In TxChain, we leverage a fairly simple type of smart contract: contingent payments (or transactions). Thereby, a transaction TX_a is constructed such that it becomes valid – and hence can be included in the underlying ledger – if and only if a set of transactions $T = \text{TX}_1, \ldots, \text{TX}_n$ was already included in the underlying ledger. Formally,

Definition 10 (Contingent Transaction)**.** *A transaction* TX_a *is contingent on a set of transactions* $T = \text{TX}_1, \ldots, \text{TX}_n$ *if* TX_a *can only be included in* C_i *iff* C *already contains* $\text{TX}_1, \ldots, \text{TX}_n$*. Formally:* $\text{TX}^a \in C_i \implies \forall j \in \{1, 2, \ldots, n\}\ \exists m \in \{0, \ldots, i\} s.t. \text{TX}_j \in C_m$.

When checking validity of a contingent transaction TX_a, *full nodes* look up the referenced transactions $\text{TX}_1, \ldots, \text{TX}_n$ in their local copy of the full valid chain, and only accept TX_a if all transactions were indeed found, as illustrated in Figure 5.1.

5.3.2 TxChain: Contingent Transaction Aggregation

We now apply the concept of contingent transactions to reduce the storage and bandwidth requirements of light clients when verifying n transaction inclusion proofs. Consider the following setting: A prover wants to convince a verifier that a set of transactions $T = \text{TX}_1, \ldots, \text{TX}_n$ was included in the valid chain C. The transactions are thereby distributed across h different blocks, $h \leq n$. In TxChain, the prover creates a *contingent transaction* TX_a, referencing transactions $\text{TX}_1, \ldots, \text{TX}_n$ and includes it in the blockchain at position i, i.e., $\text{TX}_a \in C_i$. Following Definition 10, by convincing the verifier that $\text{TX}_a \in C_i$, the prover also proves that for every $\text{TX}_1, \ldots, \text{TX}_n$ there is a block C_m ($m \in \{0, \ldots, h\}$) that includes the transaction, and all these blocks are part of the valid chain C (i.e., $\forall m \in \{0, \ldots, h\}\ \exists \beta_{(C_m, C)}$).

Specifically, the prover has a valid chain of $h + 1$ blocks $C_0, \ldots, C_h$ and receives a query for a set of transactions $T = \text{TX}_1, \ldots, \text{TX}_n$ from the verifier. The prover creates transaction TX_a,

contingent on T and includes it in the valid chain in block C_i ($i > h$). Once included, the prover computes the valid chain proof $\pi_{(C,C_{i+k})}$, the block inclusion proof $\beta_{(C_i,C)}$ and the transaction inclusion proof $\gamma_{(i,c)}$ and sends these proofs to the verifier, alongside TX_a. The verifier checks the provided proofs and that TX_a is indeed contingent on T. Once TX_a has k confirmations, the verifier accepts TX_a as proof that transactions T are in the valid chain.

Algorithm 9 TxChain Prover / Verifier Protocol for Inclusion Verification of n Transactions

Prover

1. Has valid chain of $h+1$ blocks $C_0, \ldots, C_h$
2. Receives query for transactions $T = \text{TX}_1, \ldots, \text{TX}_n$ from verifier
3. Creates transaction TX_a contingent on the set of transactions T
4. Includes it in the valid chain at position C_i, $i > h$
5. Waits k blocks until TX_a is stable
6. Computes:
   (a) the valid chain proof $\pi_{(C,C_{i+k})}$
   (b) the block inclusion proof $\beta_{(C_i,C)}$
   (c) the transaction inclusion proof $\gamma_{(i,c)}$
7. Sends $\pi_{(C,C_{i+k})}$, $\beta_{(C_i,C)}$, $\gamma_{(i,c)}$ and TX_a to the verifier

Verifier

1. Has transactions $T = \text{TX}_1, \ldots, \text{TX}_n$
2. Queries prover for a proof that transactions T are included in the valid chain
3. Receives proof $\pi_{(C,C_{i+k})}$, $\beta_{(C_i,C)}$, $\gamma_{(i,c)}$ and TX_a from the prover
4. Verifies
   (a) the valid chain proof $\pi_{(C,C_{i+k})}$
   (b) the block inclusion proof for $\beta_{(C_i,C)}$
   (c) the transaction inclusion proof $\gamma_{(i,c)}$
   (d) that transaction TX_a is contingent on transactions T
5. If everything checks out accepts the transaction inclusion proof for T

5.3.3 Hierarchical TxChain

So far, we have assumed that a single transaction TX_a can be contingent on an arbitrary number n of pre-existing transactions. Including references to $T = \{\text{TX}_1, \ldots, \text{TX}_n\}$ in TX_a, however, comes at a cost: each additional reference means additional data must be attached to TX_a. However, blockchains typically exhibit block or transaction size limits due to network latency concerns: the larger a transaction, the slower it will be propagated across the network, and the more susceptible it is to double-spending attacks [GRKC15].

Depending on the size of these identifiers, which in turn depends on the design of the underlying blockchain as well as the means of deployment of TxChain (c.f. Section 5.5), the number of transactions referenced by a single contingent transaction TX_a can be limited. We capture this by a constant $c > 1$. As long as $n \leq c$, verifying n transactions requires only a single contingent transaction.

Consider, however, a scenario where $n > c$, i.e., a prover wants to convince the verifier that a large number of transactions are included, but cannot reference them all within a single contingent transaction. To circumvent this problem, the prover splits transactions $\text{TX}_1, \ldots, \text{TX}_n$ across multiple contingent transactions $\text{TX}_{a(1)}, \ldots, \text{TX}_{a(n/c)}$. Next, the prover constructs an *hierarchical* dependency across the "first-layer" contingent transactions by creating transactions $\text{TX}_{a(n/c)}, \ldots \text{TX}_{a(n/c^2)}$. In simple terms, the prover creates a *N-arry* tree of contingent transactions, where each node is a contingent transaction acting as inclusion proof for c nodes (transactions) in that branch.

As a result, the prover can apply TxChain to an arbitrary number of transactions, at the cost of including in the blockchain and sending to the verifier $\frac{n}{c} + \lceil log_c(n) \rceil$ contingent transactions. For example, for $n = 1000$ and $c = 100$, the number of contingent transactions would be 11. This yields a 91x reduction in the required transaction and block inclusion proofs. If $c \geq n$ (e.g. $c = 1000$), the reduction in the example is 1000x. That is, the number of transactions c that can be referenced by contingent transactions directly impacts the improvement offered by TxChain.

5.4 Security and Efficiency Analysis

In this section we show how TxChain achieves the two protocol goals: *security* and *efficiency* (see Section 5.1.2).

5.4.1 Security Analysis

TxChain achieves security when the verifier terminates correctly if and only if the prover is honest.

[⇒] If the prover is honest then, all transactions are included in the valid chain C, and the proofs are generated according to the protocol specifications. Therefore, the verification of all proofs will be successful by the verifier and thus will terminate correctly.

[⇐] For the opposite direction, we will prove the statement by contradiction. Let us assume the verifier terminates correctly but the prover is malicious. This implies that the prover deviated from the protocol specification. Given that the verifier terminated, the verifier received the corresponding proofs from the prover. Since the security of the generation of the proofs is guaranteed by the underlying light client verification protocol, the prover must have deviated from the protocol during the creation of the contingent transaction. However, the verifier has the block inclusion proof for the contingent transaction and also the last k blocks headers of the chain; therefore, the prover can only deviate during the creation of the contingent transaction. However, during the verification of the transaction inclusion proof the verifier ensures that all requested transaction identifiers are tied to this transaction. Thus, the prover cannot create an incorrect contingent transaction. Contradiction. We conclude that TxChain achieves security.

Hierarchical TxChain

The security of the hierarchical TxChain construction follows the security analysis of TxChain. Now assume T encapsulates all to-be-proven transactions $\text{TX}_1, ...\text{TX}_n$, as well as the set of contingent transactions $\text{TX}_{a(1)}, ...\text{TX}_{a(x)}$ where x is upper-bound by $\frac{n}{c} + log_c(n)$, i.e.,

$T = \{\text{TX}_1, ...\text{TX}_n\} + \{\text{TX}_{a(1)}, ...\text{TX}_{a(x)}\}$. If the contingent transaction $\text{TX}_{a(x)}$, which is the root of the created N-arry tree of contingent transactions, is included in valid chain C, this means that the subset of contingent transactions $\{\text{TX}_{a(x-1-c)}, ...\text{TX}_{a(x-1)}\}$ was also included in C. The same holds for the predecessors of each transaction TX_j with $j \in \{x-1-c, ..., x-1\}$. We continue this process recursively until we observe that $\{\text{TX}_1, ...\text{TX}_n\}$ must also be included in C, for $\text{TX}_{a(x)}$ to be valid and hence included in C.

5.4.2 Efficiency Analysis

We now discuss how TxChain achieves *efficiency* by comparing the storage costs of naïve (SPV) and sublinear (NIPoPoWs and FlyClient) light clients *with* and *without* applying TxChain. We assume a secure hash function H and denote its size as $|H|$. We analyze the cost of each proof (see Section 5.1) below.

Valid Chain Proofs: The size of the valid chain proof in naïve SPV is linear in h. The size of the valid chain proof in sublinear light clients depends on two parameters: (i) λ, which defines the probability $2^{-\lambda}$ of a verifier terminating correctly on an invalid proof, (ii) α, which defines the strength of the adversary $\alpha/(1+\alpha)$, e.g. the hash rate in PoW blockchains , and (iii) the "depth" parameter k. The NIPoPoW $\pi_{(C,C_h)}$ size [KMZ17, BKLZ20] is given by

$$log_{1/\alpha}(2)\lambda \cdot ((log_2(h)+1) \cdot C + log_2(h) \cdot \lceil log_2(log_2(h)) \rceil \cdot |H|).$$

The FlyClient $\pi_{(C,C_h)}$ size [BKLZ20] is given by

$$\lambda log_{1/\alpha}(2) ln(h) \cdot (C + |H|).$$

Note the increased block header size due to the additionally required number of hashes $|H|$ in NIPoPoWs (interlink structure) and FlyClient (MMR root).

Block Inclusion Proofs ($\mathcal{B}$): Since naïve SPV clients store all block headers, no extra block inclusion proofs $\beta_{(\cdot,\cdot)}$ are required. Both NIPoPoW and FlyClient require block inclusion proofs

for blocks not sampled as part of $\pi_{(C,C_h)}$ – for both mechanisms, the size of $\beta_{(\cdot,\cdot)}$ is $log(h) \cdot |H|$ per block header.

Transaction Inclusion Proofs (Γ): A transaction inclusion proof $\gamma(i, id)$ is a list of hashes (Merkle tree path), logarithmic in the number of transactions contained in block C_i. Hence, the size of each proof is $log(t) \cdot |H|$, where t is the total number of transactions included in the block containing a transaction of T.

TxChain Efficiency

In Section 5.2, we determined the expected number of additional block headers and block inclusion proofs $\mathbb{E}(|\mathcal{B}|)$ required in NIPoPoW and FlyClient to verify the inclusion of n transactions for any given blockchain size h:

$$\mathbb{E}(|\mathcal{B}|) = (h - \sigma) \cdot (1 - (1 - \frac{1}{h})^n),$$

where σ is the number of blocks sampled for the chain validity proof. When applying TxChain to such probabilistic sampling clients, this number decreases to:

$$\mathbb{E}(|\mathcal{B}'|) = \frac{\mathbb{E}(|\mathcal{B}|)}{c} + log_c(\mathbb{E}(|\mathcal{B}|)).$$

We observe that the improvement achieved by TxChain is most significant for large c, since

$$lim_{c \to \infty} \mathbb{E}(|\mathcal{B}'|) = 1.$$

To evaluate the theoretical improvement we can achieve in TxChain, we apply TxChain as an extension to both NIPoPoW and FlyClient. Figure 5.2 overviews the expected number of (a) additional block inclusion proofs (and hence block headers) and (b) required transaction inclusion proofs, *before* and *after* applying TxChain, for blockchain size $h = 100000$ and $c = 1000$.

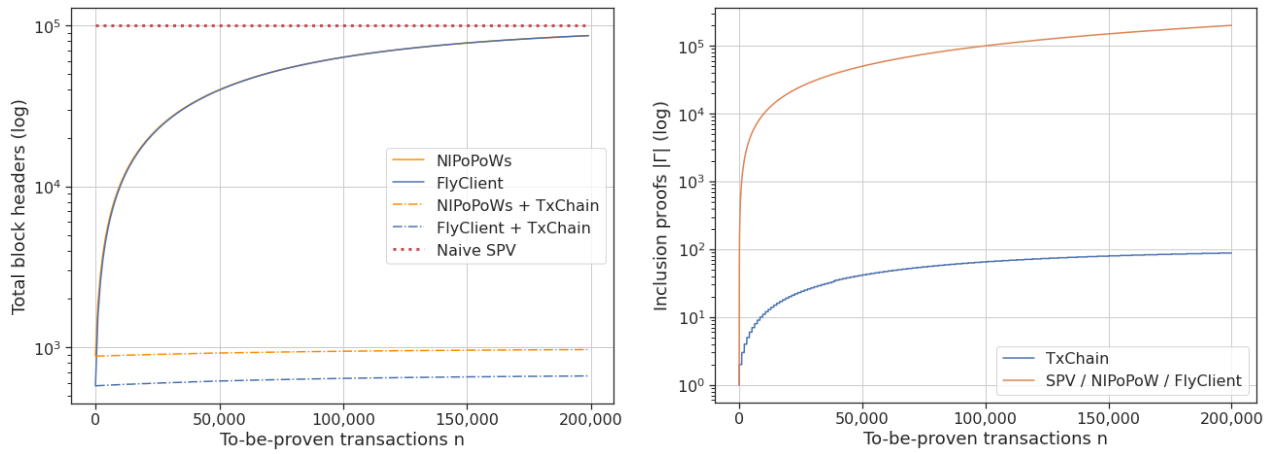
Table 5.1: Expected number of additionally required block inclusion proofs (and hence block headers) for different n in FlyClient and NIPoPoWs, before ($\mathbb{E}(|\mathcal{B}|)$) and after ($\mathbb{E}(|\mathcal{B}|)'$) applying TxChain. Results provided for a blockchain size $h = 100000$ and $c = 1000$.

n=$\|\Gamma\|$	FlyClient					Superblock NIPoPoWs				
	Vanilla		TxChain		Improvementfactor	Vanilla		TxChain		Improvementfactor
	$\mathbb{E}(\|\mathcal{B}\|)$	%	$\mathbb{E}(\|\mathcal{B}\|)'$	%		$\mathbb{E}(\|\mathcal{B}\|)$	%	$\mathbb{E}(\|\mathcal{B}\|)'$	%	
1	1	100.0	1	100.0	1.0	1	100.0	1	100.0	1.0
10	10	100.0	1	10.0	10.0	10	100.0	1	10.0	10.0
100	99	99.0	2	2.0	49.5	99	99.0	2	2.0	49.5
1000	989	98.9	2	0.2	494.5	986	98.6	2	0.2	493.0
10000	9461	94.61	11	0.11	860.09	9432	94.32	11	0.11	857.45
50000	39120	78.24	42	0.08	931.43	39000	78.0	42	0.08	928.57
100000	62848	62.85	65	0.07	966.89	62655	62.66	65	0.07	963.92
200000	85968	42.98	88	0.04	976.91	85704	42.85	88	0.04	973.91

A more detailed cost breakdown is provided in Table 5.1. We observe that as expected, TxChain becomes more effective as n increases, up until $n = |T| = h$. Statistically, given a blockchain size of 100000 and 50000 to-be-verified transactions, FlyClient on average requires the submission of 39120 block inclusion proofs and block headers, on top of the blocks sampled as part of the chain validity proof. NIPoPoWs, which sample 40% more blocks as part of the chain validity proof [BKLZ20], require 39000 additional block headers. If we apply TxChain's contingent transaction aggregation to FlyClient and NIPoPoWs, assuming a realistic $c = 1000$ (e.g. one that corresponds to a transaction with 1000 inputs in Bitcoin), we only need to download 42 additional block headers, achieving an improvement factor of 931x over FlyClient and 928x over NIPoPoWs.

TxChain achieves even higher improvement factors for higher values of $\Gamma = n$ in FlyClient and NIPoPoW, since $\mathbb{E}(|\mathcal{B}|) \leq n$. For 50000 to-be-verified transactions and a blockchain size of 100000, the use of TxChain improves over both "Vanilla" FlyClient and NIPoPoW by a factor of 1190x: instead of 50000, we require only 42 transaction inclusion proofs. It is worth mentioning that the same improvement identically applies to naïve SPV clients, as visualized in Figure 5.2(b).

We note the actual improvement in terms of storage and bandwidth costs depends on how TxChain, and specifically contingent transactions, are implemented in the underlying blockchain, as we discuss in Section 5.5.



(a) (b)

Figure 5.2: Effects of applying TxChain to FlyClient and NIPoPoWs. (a) Total number of block headers required for verification of n transactions ($\pi_{(C,C_h)} + \mathbb{E}(|\mathcal{B}|)$). (b) Number of transaction inclusion proofs Γ in light clients before and after applying TxChain (logarithmic y-axis). Numbers $h = 100000$ and $c = 1000$.

Limitations

While the design of TxChain is simple and avoids complex cryptographic schemes, making it compatible with the majority of existing blockchain systems, it also exhibits limitations. The requirement of including additional transactions in the blockchain results in additional transaction fees for the prover (c.f. Section 5.5.1). Further, TxChain may not be applicable in times of high network congestion, i.e., if a prover is unable to reliably include a contingent transaction in the blockchain. This in turn, in the worst case, may yield TxChain not applicable to instant or day-to-day payments. To summarize, TxChain is most effective in settings where the storage and especially bandwidth requirements of the verifier are the main bottleneck of a protocol, or even priced by byte – as is the case when verification is performed in on-chain smart contracts, as we show in Section 5.5.3.

5.5 Deploying TxChain in Practice

In this section, we discuss how TxChain can be added to light clients of the two major blockchains Bitcoin and Ethereum, and evaluate the achieved improvements. Note: we evaluate

both NIPoPoW and FlyClient under constant difficulty since NIPoPoW currently does not support variable difficulty [KMZ17, BKLZ20].

5.5.1 Fork Free Deployment

We first discuss how TxChain can be deployed in a fully backward compatible manner without requiring forks in Bitcoin (and similar systems). In blockchains like Ethereum, which rely on an account-based model, TxChain requires a soft or hard fork.

Bitcoin: Dust Output Spending

Bitcoin operates a so-called Unspent Transaction Output model (UTXO). Each new transaction consists of inputs and outputs, where inputs *spend* outputs of existing transactions. Outputs specify rules for how the coins *locked* in the unspent output (UTXO) can be spent, i.e. via smart contracts. In Bitcoin, these contracts are written in Script [Bit21b], a stack-based scripting language. UTXOs can only be spent as a whole. As of this writing, the only way to create conditional relations across transactions in Bitcoin is by creating *spending* relationships.

Table 5.2: Storage and bandwidth costs of naïve SPV, Flyclient and NIPoPoWs, *without* ("Vanilla") and with a *fork-free deployment* of TxChain, for different numbers of to-be-verified transactions n, for blockchain size $h = 630000$ (as of 5 May 2020) and $c = 1000$. FlyClient/NIPoPoW numbers provided for soft and hard fork deployment.

n	naïve SPV			FlyClient						Superblock NIPoPoWs					
				Soft Fork			Hard Fork			Soft Fork			Hard Fork		
	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor
1	50.4	50.4	1.0	0.51	0.51	1.0	0.1	0.1	1.0	0.77	0.77	1.0	0.15	0.15	1.0
10	50.41	50.4	1.0	0.52	0.51	1.02	0.1	0.1	1.04	0.78	0.77	1.01	0.15	0.15	1.03
100	50.49	50.4	1.0	0.62	0.51	1.21	0.15	0.1	1.5	0.88	0.77	1.14	0.2	0.15	1.33
1000	51.32	50.4	1.02	1.61	0.51	3.16	0.59	0.1	6.03	1.88	0.77	2.43	0.64	0.15	4.33
10000	59.58	50.42	1.18	11.51	0.53	21.58	5.05	0.11	44.04	11.77	0.8	14.81	5.1	0.16	30.97
50000	96.3	50.66	1.9	54.42	0.8	68.11	24.67	0.36	69.17	54.67	1.06	51.56	24.72	0.41	60.8
100000	142.2	51.39	2.77	105.69	1.5	70.68	48.84	1.03	47.61	105.92	1.76	60.31	48.89	1.08	45.46

To deploy TxChain in Bitcoin *without* consensus changes we can use *dust output* spending for the creation of contingent transactions. When creating transactions $\text{TX}_1, \ldots, \text{TX}_n$ the prover includes an additional output in each transaction, containing at least the minimum possible value transferable in Bitcoin ($54.60 \cdot 10^{-6}$ BTC which is approx. USD 0.4 as of 5 May 2020).

The contingent transaction TX_a then spends outputs of $\text{TX}_1, \ldots, \text{TX}_n$ and, due to Bitcoin's consensus rules, can hence only be included in the blockchain if all spent UTXOs already exist.

A practical limitation is that the prover must be able to spend from the outputs of transactions $\text{TX}_1, \ldots, \text{TX}_n$ when creating TX_a. In the simplest case, the prover is the author of $\text{TX}_1, \ldots, \text{TX}_n$ (i.e., controls the signing keys), and can hence spend from the corresponding outputs. For the case where $\text{TX}_1, \ldots, \text{TX}_n$ are authored by different users, we outline two simple coordination schemes. A straightforward approach is for the prover to publicly announce his public key, e.g. on a public bulletin board (can also be Bitcoin or Ethereum), such that users can create dust outputs spendable via the prover's signing key. The value accumulated from the dust outputs can thereby serve as means to cover the prover's costs for broadcasting TX_a. An approach without additional communication overhead is to use *anyone-can-spend* outputs and allow miners to aggregate $\text{TX}_1, \ldots, \text{TX}_n$ when claiming the dust outputs as fees. In both cases, the dust outputs of $\text{TX}_1, \ldots, \text{TX}_n$ can be supplemented with pre-defined timelocks, allowing transaction aggregation, e.g. only once every 6 hours. This contributes to provers/miners aggregating multiple transactions, rather than spending from each $\text{TX}_1, \ldots, \text{TX}_n$ as they are broadcast by users. Other, more complex and costly designs involving HTLCs [Bit21a] and zero-knowledge contingent payments [Bit20c] are left to future work.

Evaluation. In our evaluation, we use Bitcoin P2WPKH [Lib18] transactions. In Bitcoin, $C = 80$ bytes and $|H| = 32$ bytes. The average transaction size in 2019 was 534 bytes, while the average size of the coinbase transaction was 259 bytes, the average depth of the transaction Merkle tree was 12. The coinbase transaction is the first transaction of every block and is used by NIPoPoWs and FlyClient to include the interlink data / MMR root required for block inclusion proofs when deployed as a backward-compatible soft or velvet instead of a hard fork [ZSJ+18].

Summarizing, each block inclusion proof in NIPoPoW and FlyClient requires additionally $259 + 12 \cdot |H| = 643$ bytes, and each transaction inclusion proof 384 bytes - avoidable when applying TxChain. However, multi-input Bitcoin transactions that make use of TxChain incur some additional cost: 93 bytes per input and 45 bytes flat per contingent transaction (assuming one

P2WPKH output). Thereby, Bitcoin full nodes will relay transactions of up to 100kb[14], thus $c \approx 1000$.

We overview the storage and bandwidth costs of naïve SPV, FlyClient and NIPoPoWs with and without TxChain in Table 5.2, for a Bitcoin block height $h = 630000$ (as of 5 May 2020) and $c = 1000$. We observe that with TxChain the storage and verification costs remain *nearly constant*, offering a significant improvement over "Vanilla" light client implementations, e.g. achieving an improvement factor of 71x for FlyClient and 60x for NiPoPoWs for $n = 100000$. However, fork-free deployment comes at a cost: dust outputs increase the size of contingent transactions by 93 bytes per referenced input - hence, preventing TxChain from achieving the theoretical improvements outlined in Section 5.4.2. The costs for including a transaction with $c = 1000$ inputs in Bitcoin, at a fee of $3 \cdot 10^{-6}$ BTC per byte, amount to USD 21.2 (as of 5 May 2020).

5.5.2 Deployment via Soft or Hard Forks.

Considering both FlyClient and NIPoPoWs require a soft or hard fork to be deployed in Bitcoin and Ethereum, the minor modifications to transaction validity rules necessary for TxChain could arguably be added in parallel – if FlyClient or NIPoPoW are indeed deployed in practice. In both Bitcoin and Ethereum, TxChain can be deployed as a hard fork by introducing a new `TXEXISTS` instruction with the following semantics: (i) Pop one argument, representing the hash of a transaction, from the stack, (ii) push 1 to the stack if the transaction was found or 0 otherwise. Interestingly, Bitcoin allows re-purposing of unused instructions ("OpCodes"), enabling deployment via a soft fork.

Bitcoin: Soft Fork by Re-purposing `OP_NOP`

Bitcoin allows to introduce new instructions by re-purposing "reserved" OpCodes, e.g. `OP_NOP10`. These OpCodes are currently ignored during execution, allowing to add additional rules with-

[14] `github.com/bitcoin/bitcoin/blob/eb7daf4/src/policy/policy.h#L24`

out causing conflicts between upgraded and non-upgraded nodes. Specifically `TXEXISTS` can be implemented using the following sequence: `OP_PUSHDATA1 20 <TXID> OP_NOP10 OP_VERIFY`.

On non-upgraded nodes, `OP_NOP10` will be ignored and `OP_VERIFY` will evaluate to `true`, as the top element of the stack will be non-empty/non-zero (the transaction ID). On upgraded nodes, `OP_NOP10` will be interpreted as `TXEXISTS`, popping the transaction ID from the stack and pushing back 1 if the transaction exists or 0 otherwise. Therefore, `OP_VERIFY` will fail if and only if the node has been upgraded and the transaction ID does not exist, which enables a soft fork deployment.

A soft fork deployment in Bitcoin avoids the requirement to actually spend UTXOs in contingent transactions. This not only simplifies coordination for provers (no construction needed for the prover to spend from UTXOs), but also reduce the costs per referenced transaction / UTXO from 93 bytes (per input) to 32 bytes per transaction identifier (SHA256 hash) plus 4 bytes for the OpCode flags. This results in an expected 2.8x improvement over the fork-free deployment of TxChain. Detailed numbers are provided in Table 5.3. However, considering the simple deployment of TxChain in Bitcoin without consensus changes, we defer the implementation of `TXEXISTS` to future work.

Table 5.3: Estimates of storage and bandwidth costs of naïve SPV, Flyclient and NIPoPoWs, *without* ("Vanilla") and with a *fork-based deployment* of TxChain, for different numbers of to-be-verified transactions n. FlyClient and NIPoPoW numbers provided for soft fork and hard fork deployment. Numbers provided for a blockchain size $h = 630000$ (as of 5 May 2020) and $c = 1000$.

n	naïve SPV			FlyClient						Superblock NIPoPoWs					
				Soft Fork			Hard Fork			Soft Fork			Hard Fork		
	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor
1	50.4	50.4	1.0	0.51	0.51	1.0	0.1	0.1	1.0	0.77	0.77	1.0	0.15	0.15	1.0
10	50.41	50.4	1.0	0.52	0.51	1.02	0.1	0.1	1.04	0.78	0.77	1.01	0.15	0.15	1.03
100	50.49	50.4	1.0	0.62	0.51	1.21	0.15	0.1	1.5	0.88	0.77	1.14	0.2	0.15	1.33
1000	51.32	50.4	1.02	1.61	0.51	3.16	0.59	0.1	6.04	1.88	0.77	2.43	0.64	0.15	4.34
10000	59.58	50.41	1.18	11.51	0.53	21.87	5.05	0.11	47.03	11.77	0.79	14.94	5.1	0.16	32.4
50000	96.3	50.51	1.91	54.42	0.65	84.18	24.67	0.2	120.84	54.67	0.91	60.21	24.72	0.25	97.28
100000	142.2	50.77	2.8	105.69	0.92	114.92	48.84	0.45	108.49	105.92	1.18	89.7	48.89	0.5	97.78

Ethereum: New Instruction (Hard Fork)

Unlike Bitcoin which uses the UTXO model, Ethereum does not provide a native way of implementing transaction dependencies. Furthermore, the ID of a transaction cannot be accessed

from within smart contracts, making it impossible to implement `TXEXISTS` purely as a smart contract. To deploy TxChain on Ethereum, we hence propose a hard fork introducing `TXEXISTS` as a new instruction. We implement `TXEXISTS` in Geth (v1.9.15), the most commonly used Ethereum implementation, using Go 1.14.4 and construct a Solidity (v0.6.7) smart contract to perform TxChain transaction verification using the newly added instruction: the contract takes as input n transaction identifiers and returns `true` if all are in the valid chain, and reverts otherwise. This way, the (*contingent*) transaction calling the contract will only succeed if the n to-be-verified transactions are indeed in the main chain.

In Geth, Ethereum transactions are already indexed by hash: checking the existence of a transaction does not require any further indexing, but only a single random read in the underlying LevelDB database. This is equivalent to `EXCODESIZE` or `BALANCE` in terms of IO access [PL20]. However, given that the total number of transactions is vastly higher than the number of addresses, chances of cache miss are higher with `TXEXISTS` than with `BALANCE`. Therefore, we assign a conservative price of 2000 gas to the instruction, i.e., more than twice as expensive as the 900 gas of `EXCODESIZE` and `BALANCE`. Finding optimal pricing would require further benchmarking and is left to future work.

Evaluation. Using our modified version of the Geth client, we measure the cost of a transaction contingent on $c = 1147$ other transactions - deriving a conservative value for c by assuming a block gas limit of 5 million [PL20], i.e., 50% of the block gas limit. Given the 168.01 USD/ETH exchange rate as per 24 April 2020 and a 5 Gwei gas, this results in an upper limit of $\approx$ USD 12.6 per (full) contingent transaction. In more detail, the base costs for the contingent transaction amount to 26,633 gas (0.022 USD), and every additional referenced transaction adds 4,333 gas (0.0036 USD) to the cost. To measure the overall storage and bandwidth improvements when applied to NiPoPoWs and FlyClient, we assume the 2019 average Ethereum transaction size of 499 bytes. We note that storing a single hash in a smart contract on Ethereum, necessary to include the interlink data (NIPoPoW) or MMR root (FlyClient) in a block, requires a 167 byte transaction. Given Ethereum's block height $h = 10000000$ (as of 4 May 2020), $n = 100000$ transactions and $c = 1147$, TxChain achieves a 24x improvement over a soft fork deployment of FlyClient (28x for a hard fork) and a 17x improvement over a soft fork deployment of NIPoPoWs

(20x for a hard fork). We provide a detailed breakdown of the storage and bandwidth costs in Table 5.4.

Table 5.4: Estimates of storage and bandwidth costs of naïve SPV, Flyclient and NIPoPoWs, *without* ("Vanilla") and with a *fork-based deployment* of TxChain, for different numbers of to-be-verified transactions n. FlyClient and NIPoPoW numbers provided for soft fork and hard fork deployment. Numbers provided for a blockchain size $h = 10000000$ (as of 4 May 2020) and $c = 1047$.

n	naïve SPV			FlyClient						Superblock NIPoPoWs					
				Soft Fork			Hard Fork			Soft Fork			Hard Fork		
	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor	Vanilla in mB	TxChain in mB	Impr. factor
1	5,080.0	5,080.0	1.0	5.79	5.79	1.0	3.04	3.04	1.0	8.71	8.71	1.0	4.57	4.57	1.0
10	5,080.01	5,080.0	1.0	5.81	5.79	1.0	3.05	3.04	1.0	8.73	8.71	1.0	4.58	4.57	1.0
100	5,080.09	5,080.0	1.0	5.94	5.79	1.02	3.13	3.04	1.03	8.85	8.71	1.02	4.66	4.57	1.02
1000	5,080.88	5,080.0	1.0	7.23	5.79	1.25	3.96	3.04	1.3	10.15	8.71	1.16	5.49	4.57	1.2
10000	5,088.83	5,080.01	1.0	20.21	5.81	3.48	12.27	3.05	4.02	23.13	8.73	2.65	13.8	4.58	3.01
50000	5,124.15	5,080.08	1.01	77.78	5.92	13.13	49.15	3.14	15.63	80.69	8.84	9.12	50.68	4.68	10.84
100000	5,168.3	5,080.29	1.02	149.51	6.17	24.22	95.14	3.37	28.22	152.4	9.09	16.76	96.66	4.9	19.72

5.5.3 Case-Study: TxChain for Cross-Chain Transactions

Reducing the number of downloaded block headers and transaction inclusion proofs can be especially useful in the cross-chain setting, where storage and bandwidth costs are priced by the byte. To showcase the applicability of TxChain, we use contingent transactions to verify Bitcoin transactions on Ethereum, e.g. useful in protocols such as Xclaim [ZHL+19] where efficient cross-chain transaction inclusion proofs are imperative for secure operation. Specifically, we extend Interlay's BTC-Relay Solidity implementation [ZH19], a Bitcoin SPV client implemented as an Ethereum Solidity smart contract, with the TxChain functionality as described in Section 5.3.2 using the fork-free deployment in Bitcoin as presented in Section 5.5.1.

We compare the gas costs when verifying multiple transactions using BTC-Relay before and after applying TxChain: we are able to save up to 66.94% on the Ethereum gas costs. A detailed breakdown of the costs and improvements over the naïve SPV BTC Relay are given in the full paper version; the code is available as open source[12]. The measured cost improvements are thereby limited by Ethereum's memory pricing function: the costs are linear only up to 724 bytes (equiv. to TX_a with $n = 16$ contingent transactions), after which polynomial pricing is applied. Also, we are only able to parse TX_a with up to 90 contingent transactions in the smart contract due to Ethereum's block gas limit. As such, the savings achieved by TxChain

can be even higher on blockchain platforms with alternative memory pricing models and better support for Bitcoin primitives (e.g. Polkadot [Woo15] and RSK [Ler15]). Furthermore, applying TxChain to cross-chain SPV clients supporting FlyClient or NIPoPoWs - which, at the time of writing, do not yet exist - would further increase the cost savings, as discussed in Section 5.4.

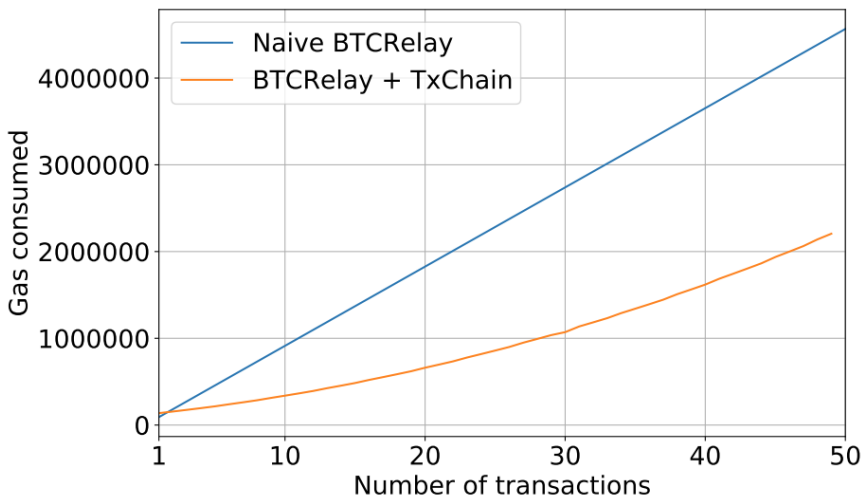


Figure 5.3: Comparison of gas costs for transaction inclusion verification and the necessary block header verification for BTC Relay without (*naïve*) and with TxChain. The block used has a total of 51 transactions.

Action	**Cost**	
	Gas	**USD**
Base function call cost	21,000	0.018
Merkle proof	38,038	0.032
Block inclusion	1,109	0.001
BTC Relay total	90,075	0.076
TxChain mean overhead, first 20 transactions	27,025	0.227
TxChain mean overhead	42,560	0.036

Figure 5.4: Breakdown of gas costs for BTC Relay verification, for a total of 51 verified Bitcoin transactions. USD costs computed with 5 Gwei gas price and 168.01 USD/ETH

5.6 Conclusion

We introduced the Probabilistic Sampling Dilemma, stating that light clients relying on probabilistic sampling suffer from inefficiency under high transaction volumes. We then presented

TxChain, a novel mechanism to reduce the number of transactions- and block inclusion proofs in blockchain light clients, leveraging contingent transaction aggregation. We showed TxChain is secure and offers significant efficiency improvements when applied as an extension to NIPoPoWs, FlyClient, and even naïve SPV clients. We implemented TxChain (i) on Bitcoin without requiring any consensus modifications, (ii) in Ethereum as a hard fork, and (iii) in a cross-chain Bitcoin light client in an Ethereum smart contract, showing the practicability of TxChain even in resource-constrained environments.

Chapter 6

Conclusion and Future Work

6.1 Summary of Thesis Achievements

In this thesis, we explored the problem of communication across distributed ledgers without requiring trusted intermediaries. We introduced the first formal definition of the Correct Cross-Chain Communication problem building upon the distributed ledger model of Garay et al [GKL16] and derived a generic protocol for cross-chain communication. Consequently, we proved the impossibility of Correct Cross-Chain Communication without a trusted third party by reduction from the Fair Exchange problem, refuting assumptions and claims commonly made within the blockchain community. Thereby, we identified the rationality and incentives of participants as the key to working around this theoretical result in practice.

Next, we introduced the Cross-Chain Design framework as a guide to designing new and evaluating existing blockchain interoperability protocols, focusing on security and trust assumptions, and applying it to classify existing cross-chain protocols. Based on our observations, we identified the main challenges of blockchain interoperability protocols in terms of security, privacy, and practicability and outlined potential avenues for future research.

Following the idea of incentivizing correct behavior of rational participants, we introduced Xclaim, the first *financially trustless* protocol to connect crypto-*currencies* like Bitcoin with

blockchain *platforms* like Ethereum and Polkadot. By imposing incentives via collateral insurance and ensuring public verifiability of misbehavior through cross-chain light clients, XCLAIM guarantees users can always redeem their cryptocurrency-backed assets for the backing asset on the underlying blockchain or will be reimbursed in a collateral currency at a beneficial rate. We showcased how XCLAIM enables users to create 1:1 Bitcoin-backed assets on Ethereum leveraging the concept of cryptocurrency-backed assets, first formalized in our work. XCLAIM requires no modifications to the underlying blockchain network and hence can be used for cross-chain transfers of the majority of existing cryptocurrencies. On a high level, cryptocurrency-backed assets created using XCLAIM resemble algorithmic stablecoins [KMHG+20], such as MakerDAO's DAI [Mak14], pegged to decentralized crypto- rather than fiat currencies and extended to support physical redemption for the underlying asset.

Finally, we presented TXCHAIN, a novel mechanism to improve the efficiency of blockchain light clients, particularly useful in the cross-chain settings where bandwidth and storage costs are priced by the byte. TXCHAIN batches a large number of transaction inclusion proofs into a single on-chain transaction, contingent on the existence and validity of the batched transactions. We showed how TXCHAIN can be used to increase the efficiency of blockchain light clients, improving in particular upon new techniques such as NiPoPoWs [KMZ17] and FlyClient [BBB+17]. TXCHAIN can be deployed on Bitcoin with and without protocol changes and as a hard fork to Ethereum. Concluding, we showed how TXCHAIN can be used to optimize chain relays between Bitcoin and Ethereum, particularly useful for systems like XCLAIM which require reliable, timely, and efficient cross-chain verification of transaction inclusion proofs.

6.2 Applications

The work presented in this thesis has found its way into industrial applications. Specifically, the XCLAIM protocol, presented in Chapter 4 has been adopted by several existing smart contract capable blockchains as a means to bridge Bitcoin and similar cryptocurrencies. Most notably, the Polkadot [Woo15] and Kusama [Web22a] blockchain networks have selected XCLAIM as

the officially recommended technique for connecting to Bitcoin and similar systems [Web22b]. Another blockchain in the process of adopting XCLAIM as the bridging mechanism is Harmony [Har21]. Similarly, a Bitcoin-backed asset system [Ane21], based on the XCLAIM protocol, is being developed for the Cardano [Car22] and Ergo [Erg22] blockchains.

Finally, Interlay, a startup founded by the author of this thesis, is further pursuing the development of XCLAIM as part of its decentralized network [Int20a, Int20b] and has deployed a production-ready version on the Kusama blockchain.

6.3 Future Work

Communication across decentralized blockchains is a relatively new field of research, only recently seeing first industrial applications deployed in practice. In this thesis, we have explored merely the tip of the iceberg of this emerging field of research and industry, attempting to lay the groundwork for what is yet to be discovered over the next years. Below, we outline what we believe to be fruitful avenues for future work, building upon this thesis.

6.3.1 Extending the CCC Framework to New Blockchain Paradigms

Interoperability Blockchains

Interoperability Blockchains are specialized sharded distributed ledgers that aim to serve as a communication layer between other blockchains [KB15, Woo15, Roc18, SN18, VTPM18, HHK+18, Wan17] and exhibit implementations of existing CCC protocols. Individual shards, which are coordinated via a parent chain running a BFT agreement protocol, connect to and import assets from existing blockchains via Migration CCC protocols, most commonly cryptocurrency-backed assets [Int20b]. A formal treatment of this design, also considering distributed computations, is presented in [LXS+19]. Cosmos [KB15] and Polkadot [Woo15] also implement new standards for (internal) cross-shard communication (IBC [Cos19] and XCMP [BCC+20] respectively). As of this writing, the aforementioned systems are under

active development, making it difficult to argue about their security, feasibility, and long-term adoption - leaving room for future analysis.

Off-Chain Protocols

One of the most actively developed fields in blockchain research are off-chain ("L2") communication networks [GMSR+19], which aim to improve scalability (and privacy) of distributed ledgers: most transactions are executed off-chain and only channel opening and closure are written to the ledger. The influx of L2 solutions is thereby creating a new field for CCC research: (i) communication across off-chain channels [MMK+17, MMSS+18, TMSM19, HAB+16], and (ii) communication between off-chain and on-chain networks [Ale21, Lig22]. While similar to conventional CCC protocols, the "off-chain" nature of L2 solutions requires more complex techniques for the verification phase of CCC: intermediate states in off-chain protocols cannot be verified by existing chain relays, which only support verification of on-chain commitments, and must hence resort to cryptographic techniques such as adaptor signatures [AEE+20] or succinct proofs of knowledge [BCCT12, BSBHR18, BBB+17].

6.3.2 Extensions and Improvements to XCLAIM

Capital Efficiency

XCLAIM is the first financially trustless cross-chain communication protocol, leveraging over-collateralization and cross-chain transaction inclusion proofs to incentivize correct behavior of network participants (in particular, Vaults) and reimburse users in case of failure. What collateralization adds in terms of security improvements, it introduces challenges in the form of capital efficiency. Vaults, who receive the backing asset (e.g. BTC) into custody, must lock up capital as collateral insurance against their own misbehavior for indefinite periods, considering it is unknown how long the created cryptocurrency-backed assets will remain in use. In practice, this means the annual fees earned by Vaults must exceed the opportunity costs of the locked-up capital, i.e., Vaults must earn more from XCLAIM than from investing their capital into other

financial products. We identify multiple paths for future work exploring different approaches to improve capital efficiency:

- **Non-custodial XCLAIM**. The reason Vaults must lock collateral is that they receive custody over the backing asset and could, in theory, commit theft. A solution is hence to explore mechanisms to avoid Vaults receiving full custody over users' assets. This can be achieved, for example, by employing multisignatures to share custody between Vault and the user. The challenge that remains to be solved is fungibility: multisignatures ultimately create a link between the Vault and the created cryptocurrency-backed asset, i.e., each CbA is uniquely tied to the Vault it was created with yielding it non-compatible with the majority of existing decentralized financial protocols. We have made a first step in this direction in follow-up work [BZ22] by combining Xclaim with concepts from our work on off-chain blockchain scaling [KZF+18] and techniques used in Bitcoin payment channels [PD16].

- **Re-using Vault Collateral**. An alternative approach is to find ways to re-use the collateral locked by Vaults, without interfering with the incentives of Xclaim. Ideas include using interest-bearing assets, i.e., assets representing investment positions in other financial products, as collateral and adding native exchange and over-collateralized lending functions to the Xclaim smart contract, enabling Vaults to trade and loan their collateral while the underlying economic value remains locked in Xclaim.

Multi-Collateral Support

Another useful improvement to Xclaim is to extend the collateral system to support more than one collateral asset. On one hand, this allows Vaults to diversify their collateral position, providing them with more flexibility to protect Xclaim against exchange rate fluctuations. On the other hand, this is a prerequisite to accepting interest-bearing assets as collateral means to improve capital efficiency.

6.3.3 Efficient Cross-Chain Light Clients

Cross-chain state verification via chain relays is a fundamental part of robust CCC protocols. While current light/SPV clients suffice for e.g., mobile devices, they often remain infeasible for deployment on top of blockchains for CCC protocols, where storage and bandwidth are priced by the byte. Recent works on sub-linear light clients, including TxChain presented in this work, have achieved first significant theoretical [KMZ17, BKLZ20, KPZ20] and practical performance improvements [DKKZ20, ZAPK20, WE20b]. A potential direction for future research is the applicability of zero-knowledge cryptography [BCCT12, BSBHR18, BBB$^+$17] as means to pave the way towards (near)constant verification times and costs for chain relays. First schemes for blockchains with built-in support for such proof systems have been proposed in [MS18, BBF18, BSCS16] but are yet to be tested in practice.

Bibliography

[AB19] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.

[ABC16] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts. Cryptology ePrint Archive, Report 2016/1007, Oct 2016. Accessed: 2016-11-08.

[ABSB+18] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. In *2018 Network and Distributed System Security Symposium (NDSS)*, 2018.

[ABV+18] John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A decentralized blockchain oracle. *arXiv preprint arXiv:1808.00528*, 2018.

[ACDCKK18] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*, pages 111–131. Springer, 2018.

[AEE+20] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized bitcoin-compatible channels. *IACR Cryptol. ePrint Arch.*, 2020:476, 2020.

[AFJ06] Dana Angluin, Michael J Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems*, pages 37–50, 2006.

[AGM18] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. arXiv:1803.05069, 2018.

[Air22] Airswap. Airswap. `https://www.airswap.io/`, 2022. Accessed 2021-07-30.

[AKKW19] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *arXiv preprint arXiv:1910.10434*, 2019.

[AKW19] Georgia Avarikioti, Eleftherios Kokoris Kogias, and Roger Wattenhofer. Brick: Asynchronous state channels. *arXiv preprint arXiv:1905.11360*, 2019.

[AKWW19] Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. Bitcoin security under temporary dishonest majority. In *23rd Financial Cryptography and Data Security (FC)*, 2019.

[Ale21] Alexei Bosworth et al. Submarine swaps service. Online. `https://github.com/submarineswaps/swaps-service`, 2021. Accessed: 2022-01-23.

[ALS+18] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. Towards secure and efficient payment channels. *arXiv preprint arXiv:1811.12740*, 2018.

[And15] Marcin Andrychowicz. Multiparty computation protocols based on cryptocurrencies, 2015. Accessed: 2017-02-15.

[Ane21] AnetaBTC. AnetaBTC Litepaper. `https://medium.com/@anetaBTC/anetabtc-litepaper-v1-0-171f29b3276a`, 2021. Accessed: 2022-01-08.

[ASB18] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *CoRR*, abs/1809.09044, 2018.

[Aso98] Nadarajah Asokan. Fairness in electronic commerce. 1998.

[ASW98a] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, pages 86–99. IEEE, 1998.

[ASW98b] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 591–606. Springer, 1998.

[Aur19] Aurora Labs. Idex whitepaper. `https://blog.idex.io/resources/idex-whitepaper`, 2019. Accessed 2022-01-13.

[Bal] Clare Baldwin. Bitcoin worth $72 million stolen from bitfinex exchange in hong kong. *Reuters*. Accessed 2021-05-23.

[BBB+17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Efficient range proofs for confidential transactions, 2017. Accessed:2017-11-10.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Bulletproofs: Short Proofs for Confidential Transactions and More*. IEEE, 2018.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, 2018.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188, 2018. `https://eprint.iacr.org/2018/1188`.

[BCC+20] Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinç Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stew-

art, et al. Overview of polkadot and its design considerations. *arXiv preprint arXiv:2005.13456*, 2020.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[BCD+14] 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, 2014.

[BCG15a] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, 2015:1015, 2015.

[BCG15b] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.

[BDM93] Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.

[Beh20] Alexander Behrens. A massive honeypot: Ren holds $100m in bitcoin in centralized wallet. Decrypt. `https://decrypt.co/40110/massive-honeypot-ren-holds-100m-bitcoin-centralized-wallet`, 2020. Accessed: 2022-01-30.

[BGB17] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. Proofs-of-delay and randomness beacons in ethereum. *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.

[BGDE18] Peter Bennink, Lennart van Gijtenbeek, Oskar van Deventer, and Maarten Everts. An analysis of atomic swaps on and between ethereum blockchains using smart contracts. Tech. report, 2018. `https://work.delaat.net/rp/2017-2018/p42/report.pdf`.

[BHG87] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency control and recovery in database systems*, volume 370. Addison-wesley New York, 1987.

[Bin19] Binance. Bitcoin-Pegged Token on Binance Chain. https://www.binance.com/en/blog/347360878904684544/introducing-bitcoinpegged-token-on-binance-chain, 2019. Accessed 2021-04-05.

[Bin22] Binance. Binance exchange. Online, 2022. https://www.binance.com/en, Accessed 2022-01-13.

[Bit13] Bitcoin Community. Atomic swap. Bitcoin Wiki, 2013.

[Bit15] Bitcoin Community. Bitcoin wiki: Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification, 2015. Accessed: 2022-01-13.

[bit18] Confirmations. https://en.bitcoin.it/wiki/Confirmation, 2018. Accessed 2021-11-28.

[Bit19] Bitcoin Community. Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://en.bitcoinwiki.org/wiki/Simplified_Payment_Verification, 2019. Accessed 2021-05-16.

[Bit20a] Bitcoin Community. Bitcoin Wiki: Atomic cross-chain trading. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading, 2020. Accessed 2021-05-16.

[Bit20b] Bitcoin Community. Pay-to-Pubkey Hash. https://en.bitcoinwiki.org/wiki/Pay-to-Pubkey_Hash, 2020. Accessed 2022-01-09.

[Bit20c] Bitcoin Community. Zero Knowledge Contingent Payments. en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, 2020. Accessed 2022-01-30.

[Bit20d] Bitcoin Wiki. Invoice address. https://en.bitcoin.it/wiki/Invoice_address, 2020. Accessed: 2022-07-25.

[Bit21a] Bitcoin Community. Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts, 2021. Accessed 2021-05-16.

[Bit21b] Bitcoin Community. Script. https://en.bitcoin.it/wiki/Script, 2021. Accessed 2021-11-28.

[BJZ+17] Iddo Bentov, Yan Ji, Fan Zhang, Yunqi Li, Xueyuan Zhao, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. Cryptology ePrint Archive, Report 2017/1153, 2017. Accessed:2017-12-04.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology–CRYPTO 2014*, pages 421–439, 2014.

[BKLZ20] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Superlight clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020.

[Blo22] Blockchain.com. Bitcoin transaction fees. https://www.blockchain.com/en/charts/transaction-fees?timespan=all, 2022. Accessed 2022-01-13.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Annual International Cryptology Conference*, pages 236–254. Springer, 2000.

[Bon16] Joseph Bonneau. Why buy when you can rent? bribery attacks on bitcoin consensus. In *BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research*, February 2016.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. Accessed: 2016-11-08.

[BSAB+17] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. arXiv:1711.03936, 2017. Accessed:2017-12-11.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptol. ePrint Arch., Tech. Rep*, 46:2018, 2018.

[BSCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2016.

[BT93] Ozalp Babaoglu and Sam Toueg. Understanding non-blocking atomic commitment. *Distributed systems*, pages 147–168, 1993.

[But14] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2014. Accessed: 2016-08-22.

[But16] Vitalik Buterin. Chain interoperability. Tech. report, `https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf`, 2016. Accessed: 2017-03-25.

[But18] Vitalik Buterin. Cross-shard contract yanking. Online, `https://ethresear.ch/t/cross-shard-contract-yanking/1450`, 2018.

[But20] Vitalik Buterin. Why does ethereum use secp256k1? Ethereum Community Forum. `https://forum.ethereum.org/discussion/comment/53/#Comment_53`, 2020. Accessed: 2021-04-02.

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

[BWQ99] S Blake-Wilson and M Qu. Standards for efficient cryptography (sec) 2: Recommended elliptic curve domain parameters. *Certicom Research, Oct*, 1999.

[BZ22] Theodore Bugnet and Alexei Zamyatin. XCC: Theft-Resilient and Collateral-Optimized Cryptocurrency-Backed Assets. Online. `https://docs.interlay.io/_assets/papers/XCC_paper.pdf`, 2022.

[C+16] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, pages 1–4. Chicago, IL, 2016.

[Car22] Cardano. Cardano blockchain. `https://cardano.org/`, 2022. Accessed: 2022-01-08.

[CC00] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Annual International Cryptology Conference*, pages 93–111. Springer, 2000.

[CCLM09] Thomas Chesney, Iain Coyne, Brian Logan, and Neil Madden. Griefing in virtual worlds: causes, casualties and coping strategies. *Information Systems Journal*, 19(6):525–548, 2009.

[CDE+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gün. On scaling decentralized blockchains. In *3rd Workshop on Bitcoin and Blockchain Research, Financial Cryptography 16*, 2016.

[CDFZ17] Alexander Chepurnoy, Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake, 2017. Accessed: 2017-03-22.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*, pages 55–72. Springer, 2013.

[Chr15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[CKWN16] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167. ACM, 2016.

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

[Coi19] Coinmarketcap. Top Cryptocurrency Decentralized Exchanges. `https://coinmarketcap.com/rankings/exchanges/dex/`, 2019. Accessed 2021-04-05.

[Coi22] CoinMarketCap. CoinMarketCap. `https://coinmarketcap.com/`, 2022. Accessed: 2022-01-09.

[Con17a] Consensys. BTC Relay Serpent Implementation. `https://github.com/ethereum/btcrelay`, 2017. Accessed 2022-01-09.

[Con17b] Consensys. Project alchemy. `https://github.com/ConsenSys/Project-Alchemy`, 2017. Accessed 2022-01-09.

[Cos19] Cosmos Community. Inter-blockchain communication protocol (ibc) specification. Online. `https://github.com/cosmos/ibc`, 2019. Accessed: 2021-04-02.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. volume 43, pages 225–267, 1996.

[CZDK17] Zhi-dong CHEN, YU Zhuo, Zhang-bo DUAN, and HU Kai. Inter-blockchain communication. *DEStech Transactions on Computer Science and Engineering*, (cst), 2017.

[Dec17] Decred. Decred cross-chain atomic swapping. `https://github.com/decred/atomicswap`, 2017. Accessed 2021-05-16.

[DEF18] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984. ACM, 2018.

[DEFM17] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. Cryptology ePrint Archive, Report 2017/635, 2017. Accessed:2017-11-20.

[Dex17] Dexaran. Erc223: Token standard. `https://github.com/ethereum/EIPs/issues/223`, 2017. Accessed 2021-06-27.

[DFZ16] Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716, 2016. Accessed: 2017-02-06.

[DGKR18] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[DH20] Apoorvaa Deshpande and Maurice Herlihy. Privacy-preserving cross-chain atomic swaps. In *International Conference on Financial Cryptography and Data Security*, pages 540–549. Springer, 2020.

[Dij01] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.

[DKKZ20] Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. A gas-efficient superlight bitcoin client in solidity. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 132–144, 2020.

[Dog21] Dogethereum. Dogerelay. `https://github.com/dogethereum/dogerelay`, 2021. Accessed 2022-01-09.

[Dou02] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[DPW+16] Johnny Dilley, Andrew Poelstra, Jonathan Wilkins, Marta Piekarska, Ben Gorlick, and Mark Friedenbach. Strong federations: An interoperable blockchain solution to centralized third party risks. *arXiv preprint arXiv:1612.05491*, 2016.

[DW14] Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and mtgox. In *Computer Security-ESORICS 2014*, pages 313–326, 2014.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[ear22] earn.com. Predicting bitcoin fees for transactions. `https://bitcoinfees.earn.com/`, 2022. Accessed 2021-11-28.

[EJS17] Steve Ellis, Ari Juels, and Nazarov Sergey. Chainlink: A decentralized oracle network. Online, 2017. `https://research.chain.link/whitepaper-v1.pdf`, Accessed 2021-09-19.

[EMSM19] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *CCS*, 2019.

[Erg22] Ergo platform. Ergo. `https://ergoplatform.org`, 2022. Accessed: 2022-01-08.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454, 2014.

[ET18] Jacob Eberhardt and Stefan Tai. Zokrates-scalable privacy-preserving off-chain computations. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1084–1091. IEEE, 2018.

[eth15] ethers. Ethereum contract allowing ether to be obtained with bitcoin. https://github.com/ethers/EthereumBitcoinSwap, 2015. Accessed: 2018-10-30.

[Eth17] Ethereum. Serpent programming language. https://github.com/ethereum/serpent, 2017. Accessed 2022-01-09.

[Eth21] Etherscan.io. Ethereum BlockSize History. https://etherscan.io/chart/blocksize, 2021. Accessed 2021-11-28.

[Eth22a] EtherDelta. Etherdelta. https://etherdelta.com/, 2022. Accessed 2021-07-30.

[Eth22b] Ethereum. Solidity progamming language. https://github.com/ethereum/solidity, 2022. Accessed 2022-01-09.

[Eth22c] Etherscan. Ropsten testnet explorer, 2022.

[Eve82] Shimon Even. A protocol for signing contracts. Technical report, Computer Science Department, Technion. Presented at CRYPTO'81, 1982.

[Eve18] EveripediaNetwork. Eth-eos-relay. https://github.com/EveripediaNetwork/eth-eos-relay, 2018. Accessed 2019-08-15.

[EY80] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical report, Computer Science Department, Technion, 1980.

[FGKJ18] Bryan Ford, Linus Gasser, Eleftherios Kokoris Kogias, and Philipp Jovanovic. Cryptographically verifiable data structure having multi-hop forward and backwards links and associated systems and methods. Google Patents, December 13 2018. US Patent App. 15/618,653.

[Fis83] Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International Conference on Fundamentals of Computation Theory*, pages 127–140, 1983.

[FKO+18] Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of wealth in proof-of-stake cryptocurrencies. *arXiv preprint arXiv:1809.07468*, 2018.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. volume 32, pages 374–382, 1985.

[Fuz08] Rachele Fuzzati. *A formal approach to fault tolerant distributed consensus*. PhD thesis, 2008.

[Gär98] Felix C Gärtner. Specifications for fault tolerance: A comedy of failures. 1998.

[GESM17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, page 2. ACM, 2017.

[GGJ+20] Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, Michael Straka, and Eran Tromer. Plumo: Towards scalable interoperable blockchains using ultra light validation systems. 2020.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016.

[GK18] Juan Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. Cryptology ePrint Archive, Report 2018/754, 2018.

[GKCC14] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? volume 12, pages 54–60, 2014.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310, 2015.

[GKL16] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty, 2016. Accessed: 2017-02-06.

[GKO20] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. *arXiv preprint arXiv:2002.01847*, 2020.

[GKR18] Peter Gaži, Aggelos Kiayias, and Alexander Russell. Stake-bleeding attacks on proof-of-stake blockchains. Cryptology ePrint Archive, Report 2018/248, 2018. Accessed:2018-03-12.

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdo rf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC*, pages 3–16. ACM, 2016.

[GKZ19a] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *To appear in the Proceedings of the IEEE Symposium on Security & Privacy*. IEEE Computer Society Press, 2019.

[GKZ19b] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. *IEEE Security and Privacy. IEEE*, 2019.

[GM16] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701, 2016. Accessed: 2017-08-07.

[GMSR+19] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. Cryptology ePrint Archive, Report 2019/360, 2019. `https://eprint.iacr.org/2019/360`.

[Gol98] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[GRKC15] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[HAB+16] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub, 2016. Accessed: 2017-09-29.

[Har21] Harmony. Kusama Guide. `https://github.com/harmony-one/onebtc`, 2021. Accessed: 2022-01-08.

[HB18] Dominik Harz and Magnus Boman. The scalability of trustless trust. In *International Conference on Financial Cryptography and Data Security*, pages 279–293. Springer, 2018.

[Her18] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.

[HH18] Abraham Hinteregger and Bernhard Haslhofer. An empirical analysis of monero cross-chain traceability. *arXiv preprint arXiv:1812.02808*, 2018.

[HHK+18] Dr Hosp, Toby Hoenisch, Paul Kittiwongsunthorn, et al. Comit-cryptographically-secure off-chain multi-asset instant transaction network. *arXiv preprint arXiv:1810.02174*, 2018.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[HLG19] Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The arwen trading protocols. Whitepaper, 2019. `https://www.arwen.io/whitepaper.pdf`.

[HLS19] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *arXiv preprint arXiv:1905.09743*, 2019.

[HLY19] Runchao Han, Haoyu Lin, and Jiangshan Yu. On the optionality and fairness of atomic swaps. Cryptology ePrint Archive, Report 2019/896, 2019. https://eprint.iacr.org/2019/896.

[IN83] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983.

[Int20a] Interlay. interBTC Open Source Implementation. https://github.com/interlay/interbtc, 2020. Accessed: 2022-01-08.

[Int20b] Interlay. InterBTC Specification. https://spec.interlay.io/, 2020. Accessed: 2022-01-08.

[Int22] Intel. Intel software guard extensions (intel® sgx) sdk. https://github.com/intel/linux-sgx, 2022. Accessed 2022-01-13.

[Ja19] John Jones and abitmore. Optional htlc preimage length and hash160 addition. BSIP 64, blog post, 2019. https://github.com/bitshares/bsips/issues/163.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[JRB19] Sandra Johnson, Peter Robinson, and John Brainard. Sidechains and interoperability. *arXiv preprint arXiv:1903.04077*, 2019.

[JSZ+19] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar Weippl. Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. Cryptology ePrint Archive, Report 2019/775, 2019. https://eprint.iacr.org/2019/775.

[JZS+17] Aljosha Judmayer, Alexei Zamyatin, Nicholas Stifter, Artemios G. Voyiatzis, and Edgar Weippl. Merged mining: Curse or cure? In *CBT'17: Proceedings*

of the International Workshop on Cryptocurrencies and Blockchain Technology, Sep 2017.

[KB15] Jae Kwon and Ethan Buchman. Cosmos: A network of distributed ledgers. `https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md`, 2015.

[KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 418–429. ACM, 2016.

[Kee19] Keep Network. tbtc: A decentralized redeemable btc-backed erc-20 token. `http://docs.keep.network/tbtc/index.pdf`, 2019. Accessed: 2022-01-13.

[KGC+18] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*, pages 1353–1370. USENIX Association, 2018.

[KJG+16] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, August 2016.

[KK19] Eleftherios Kokoris-Kogias. Robust and scalable consensus for sharded distributed ledgers. Technical report, Cryptology ePrint Archive, Report 2019/676, 2019.

[KKAS+18] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gailly, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. Calypso: Auditable sharing of private data over blockchains. Technical report, Cryptology ePrint Archive, Report 2018/209, 2018.

[KKGK+16] Lefteris Kokoris-Kogias, Linus Gasser, Ismail Khoffi, Philipp Jovanovic, Nicolas Gailly, and Bryan Ford. Managing identities using blockchains and CoSi. In *9th*

Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016), 2016.

[KKJG+18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[KKSK19] Yujin Kwon, Hyoungshick Kim, Jinwoo Shin, and Yongdae Kim. Bitcoin vs. bitcoin cash: Coexistence or downfall of bitcoin cash? arXiv:1902.11064, 2019.

[KKZ20] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer, 2020.

[KL12] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50–63, 2012.

[KLS16] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, Springer, 2016.

[KM20] Hugh Karp and Reinis Melbrandis. Nexus mutual: A peer-to-peer discretionary mutual on the ethereum blockchain. Online, 2020. `https://nexusmutual.io/assets/docs/nmx_white_paperv2_3.pdf`, Accessed 2021-09-19.

[KMHG+20] Ariah Klages-Mundt, Dominik Harz, Lewis Gudgeon, Jun-You Liu, and Andreea Minca. Stablecoins 2.0: Economic foundations and risk-based models. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 59–79, 2020.

[KMZ17] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963, 2017. Accessed:2017-10-03.

[KNW19] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. 2019.

[Kob91] Neal Koblitz. Cm-curves with good cryptographic properties. In *Annual international cryptology conference*, pages 279–287. Springer, 1991.

[Kom19] Komodo. Barterdex. `https://docs.komodoplatform.com/whitepaper/chapter6.html`, 2019. Accessed 2022-01-13.

[KPZ20] Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. *IACR Cryptology ePrint Archive*, 2020:1122, 2020.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[Kyb17] Kyber Network. Peace relay. `https://github.com/KyberNetwork/peace-relay`, 2017. Accessed 2022-01-09.

[Kyb19a] Kyber Network. Eos-eth relay. `https://github.com/KyberNetwork/bridge_eth_smart_contracts`, 2019. Accessed 2022-01-17.

[Kyb19b] Republic Protocol Kyber Network, BitGo Inc. Wrapped bitcoin. `https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf`, 2019. Accessed: 2018-05-03.

[Kyb22] Kyber Network. Kyber swap. `https://docs.kyberswap.com/`, 2022. Accessed 2022-01-13.

[KZ18] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.

[KZ19] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*, pages 21–34, 2019.

[KZF+18] Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. *Cryptology ePrint Archive, Report 2018/642*, 2018.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

[Lam89] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, 1989.

[LEK+17] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter Pietzuch, and Emin Gun Sirer. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454*, 2017.

[Ler15] S Demian Lerner. Rootstock: Bitcoin powered smart contracts. `https://docs.rsk.co/RSK_White_Paper-Overview.pdf`, 2015.

[Ler18] Sergio Demian Lerner. Drivechains, sidechains and hybrid 2-way peg designs. Technical report, Tech. Rep. [Online], 2018.

[Lib18] Libbitcoin developers. P2WSH Transactions. `https://github.com/libbitcoin/libbitcoin-system/wiki/P2WPKH-Transactions`, 2018. Accessed 2022-01-30.

[Lig22] Lightning Labs. Lightning loop. Online. `https://github.com/lightninglabs/loop`, 2022. Accessed: 2022-01-23.

[Lit21] Litecoin community. Litecoin reference implementation. `github.com/litecoin-project/litecoin`, 2021. Accessed 2021-06-30.

[Lla22] DeFi Llama. DeFi Dashboard. `https://defillama.com/`, 2022. Accessed: 2022-01-09.

[LXS+19] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, and Yih-Chun Hu. Hyperservice: Interoperability and programmability across heterogeneous blockchains. *arXiv preprint arXiv:1908.09343*, 2019.

[Mak14] MakderDAO. Maker Protocol Documentation. `https://makerdao.com/en/whitepaper/`, 2014. Accessed: 2022-01-09.

[MBB+18] Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.

[MBKM17] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning, 2017. Accessed: 2017-03-22.

[MCJ17] Dmitry Meshkov, Alexander Chepurnoy, and Marc Jansen. Revisiting difficulty control for blockchain systems. Cryptology ePrint Archive, Report 2017/731, 2017. Accessed: 2017-08-03.

[MD19] Mahdi Miraz and David C Donald. Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. *Annals of Emerging Technologies in Computing (AETiC) Vol*, 3, 2019.

[Mer87] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

[MHM17] Patrick McCorry, Ethan Heilman, and Andrew Miller. Atomically trading with roger: Gambling on the success of a hardfork. In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*, Sep 2017.

[MHM18] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.

[Mic16] Silvio Micali. Algorand: The efficient and democratic ledger, 2016. Accessed: 2017-02-09.

[Mil12] Andrew Miller. The high-value-hash highway, bitcoin forum post, 2012.

[MMK+17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *CCS*, pages 455–471, 2017.

[MMSS+18] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Multi-hop locks for secure, privacy-preserving and interoperable payment-channel networks. *Cryptology ePrint Archive, Report 2018/472*, 2018.

[Moo13] Moore, Tyler and Christin, Nicolas. Beware the middleman: Empirical analysis of bitcoin-exchange risk. In *International Conference on Financial Cryptography and Data Security*, pages 25–33. Springer, 2013.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[MS18] Izaak Meckler and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. `https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf`, 2018.

[MSRL+19] Pedro Moreno-Sanchez, Randomrun, Duc V. Le, Sarang Noether, Brandon Goodell, and Aniket Kate. Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. Cryptology ePrint Archive, Report 2019/595, 2019. `https://eprint.iacr.org/2019/595`.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.

[Nam22] Namecoin community. Namecoin reference implementation. `https://github.com/namecoin/namecoin-core`, 2022. Accessed 2022-01-09.

[Nas51] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[NKKJ+] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds.

[Noe20] Sarang Noether. Discrete logarithm equality across groups. Online. `https://www.getmonero.org/resources/research-lab/pubs/MRL-0010.pdf`, 2020. Accessed: 2022-01-23.

[NRTV07] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

[Pag14] Jose Pagliery. Another bitcoin exchange goes down. CNN Tech, `http://money.cnn.com/2014/02/11/technology/bitcoin-bitstamp/`, 2014. Accessed 2021-05-23.

[Par20] Parity Technologies. Parity-Bridge. `https://github.com/paritytech/parity-bridge`, 2020. Accessed 2022-01-09.

[PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. `https://lightning.network/lightning-network-paper.pdf`, 2016. Accessed: 2016-07-07.

[PG99] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology ..., 1999.

[PL20] Daniel Perez and Benjamin Livshits. Broken metre: Attacking resource metering in evm. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[PoA18] PoA Network. Poa bridge. `https://github.com/poanetwork/poa-bridge`, 2018. Accessed 2021-05-23.

[Poe17] Andrew Poelstra. Scriptless scripts. Presentation slides, 2017. https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf.

[PS16] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. *Cryptology ePrint Archive*, 2016.

[PSs16] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks, 2016. Accessed: 2016-08-01.

[Pto20] Ptokens.io. Provable ptokens. Online, 2020. https://ptokens.io/ptokens-rev5b.pdf ,Accessed 2021-09-19.

[Pul22a] DeFi Pulse. Bitcoin at Work. https://defipulse.com/btc, 2022. Accessed: 2022-01-09.

[Pul22b] DeFi Pulse. The Decentralized Finance Leaderboard. https://defipulse.com/, 2022. Accessed: 2022-01-09.

[Ren20] Ren. Renvm. Online, 2020. https://renproject.io/litepaper.pdf, Accessed 2021-09-19.

[RNS14] Jeremy Rubin, Manali Naik, and Nitya Subramanian. Merkelized abstract syntax trees. http://www.mit.edu/~jlrubin/public/pdfs/858report.pdf, 2014.

[Rob20] Peter Robinson. The merits of using ethereum mainnet as a coordination blockchain for ethereum private sidechains. *The Knowledge Engineering Review*, 35, 2020.

[Roc18] Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies. 2018. Accessed: 2022-01-23.

[Ros12] Meni Rosenfeld. Overview of colored coins. https://bitcoil.co.il/BitcoinX.pdf, 2012. Accessed: 2016-03-09.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[RWG+18] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, et al. Tls-n: Non-repudiation over tls enabling ubiquitous content signing. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[SBABD19] Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, and George Danezis. Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. *arXiv preprint arXiv:1901.11218*, 2019.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[SDF+19] Vasilios A. Siris, Dimitrios Dimopoulos, Nikos Fotiou, Spyros Voulgaris, and George C. Polyzos. Interledger smart contracts for decentralized authorization to constrained things, 2019.

[SJS+18] Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Alexei Zamyatin, and Edgar Weippl. Agreement with satoshi - on the formalization of nakamoto consensus. Cryptology ePrint Archive, Report 2018/400, 2018.

[SKK20] Alistair Stewart and Eleftherios Kokoris-Kogia. Grandpa: a byzantine finality gadget. *arXiv preprint arXiv:2007.01560*, 2020.

[SN18] Matthew Spoke and Nuco Engineering Team. Aion: The third-generation blockchain network. `https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf`, 2018. Accessed 2021-04-17.

[SSJ+19] Nicholas Stifter, Philipp Schindler, Aljosha Judmayer, Alexei Zamyatin, Andreas Kern, and Edgar Weippl. Echoes of the past: Recovering blockchain metrics from merged mining. In *Proceedings of the 23nd International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2019.

[SSZ15] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin, 2015. Accessed: 2016-08-22.

[Sta22] ETH Gas Station. Ethereum fee estimates, 2022.

[Ste12] Iain Stewart. Proof of burn. `https://en.bitcoin.it/wiki/Proof_of_burn`, 2012. Accessed 2021-05-10.

[Syn20] Synthetix. Synthetix System Documentation. `https://docs.synthetix.io/litepaper`, 2020. Accessed 2021-04-05.

[Syv98] Paul Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Proceedings. 11th IEEE Computer Security Foundations Workshop (Cat. No. 98TB100238)*, pages 2–13. IEEE, 1998.

[SZ16] Yonatan Sompolinsky and Aviv Zohar. Bitcoin's security model revisited. *arXiv preprint arXiv:1605.09193*, 2016.

[SZ18] Yonatan Sompolinsky and Aviv Zohar. Phantom: A scalable blockdag protocol. Cryptology ePrint Archive, Report 2018/104, 2018. Accessed:2018-01-31.

[TE17] Jason Teutsch and TrueBit Estsblishment. On decentralized oracles for data availability. 2017.

[Tea20] Huobi Blockchain Team. H-Tokens White Paper. `https://www.htokens.finance/static/pdf/whitepaper-en.pdf`, 2020. Accessed 2021-04-05.

[Tec21] Parity Technologies. The parity light protocol - wiki. `https://github.com/ethereum/devp2p/blob/master/caps/pip.md`, 2021. Accessed 2021-10-30.

[Tet16] Tether. Tether: Fiat currencies on the bitcoin blockchain. `https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf`, 2016.

[Tie13] TierNolan (pseudonym). Alt chains and atomic transfers. `https://bitcointalk.org/index.php?topic=193281.msg2003765#msg2003765`, 2013.

[Tie16] TierNolan. Atomic swaps using cur and choose. `https://bitcointalk.org/index.php?topic=1364951`, 2016. Accessed 2021-05-16.

[TMSM19] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^2l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. Cryptology ePrint Archive, Report 2019/589, 2019. `https://eprint.iacr.org/2019/589`.

[TR19] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756*, 2019.

[TS15] Stefan Thomas and Evan Schwartz. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf*, 2015.

[TSB19] Jason Teutsch, Michael Straka, and Dan Boneh. Retrofitting a two-way peg between blockchains. *arXiv preprint arXiv:1908.03999*, 2019.

[TT17] Paolo Tasca and Claudio J Tessone. Taxonomy of blockchain technologies. principles of identification and classification. *arXiv preprint arXiv:1708.04872*, 2017.

[VB15] Fabian Vogelsteller and Vitalik Buterin. Erc20: Token standard. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`, 2015. Accessed 2021-06-27.

[VBMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[VTPM18] Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. Quant overledger whitepaper. https://www.quant.network/, 2018.

[Vuk15] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[Wan17] Wanchain. Wanchain whitepaper. `https://www.wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf`, 2017.

[WB17] Will Warren and Amir Bandeali. 0xproject whitepaper. `https://0xproject.com/pdfs/0x_white_paper.pdf`, 2017. Accessed 2021-05-23.

[WE20a] Martin Westerkamp and Jacob Eberhardt. zkrelay: Facilitating sidechains using zksnark-based chain-relays. In *Workshop on the Security & Privacy on the Blockchain*. IEEE, 2020.

[WE20b] Martin Westerkamp and Jacob Eberhardt. zkrelay: Facilitating sidechains using zksnark-based chain-relays. *Contract*, 1(2):3, 2020.

[Web22a] Web3 Foundation. Kusama Guide. `https://guide.kusama.network/`, 2022. Accessed: 2022-01-08.

[Web22b] Web3 Foundation. Polkadot Wiki - Bridging Methods. `https://wiki.polkadot.network/docs/learn-bridges#bridging-methods`, 2022. Accessed: 2022-01-08.

[Wik20] Ethereum Wiki. Light client protocol. `https://eth.wiki/en/concepts/light-client-protocol`, 2020. Accessed 2021-11-20.

[WKPK16] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.

[Woo15] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper, https://polkadot.network/PolkaDotPaper.pdf*, 2015.

[Woo17] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07). `https://ethereum.github.io/yellowpaper/paper.pdf`, 2017. Accessed: 2018-01-03.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 640–656. IEEE, 2015.

[XWS+17] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 243–252. IEEE, 2017.

[Yak18] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper, https://solana.com/solana-whitepaper.pdf*, 2018.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[YKM19] Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 837–850, 2019.

[YSL+19] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. *arXiv preprint arXiv:1910.01247*, 2019.

[ZABZ+19] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. Technical report, IACR Cryptology ePrint Archive, 2019: 1128, 2019.

[ZAPK20] Alexei Zamyatin, Zeta Avarikioti, Daniel Perez, and William J Knottenbelt. Txchain: Efficient cryptocurrency light clients via contingent transaction aggregation. Sep 2020.

[ZH19] Alexei Zamyatin and Dominik Harz. BTC Relay Solidity Implementation. `https://github.com/crossclaim/btcrelay-sol`, 2019. Accessed 2022-01-09.

[ZHL+19] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. *IEEE Security and Privacy. IEEE*, 2019.

[ZJ18] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains (technical report). Technical report, 2018.

[ZMM+19] Fan Zhang, Sai Krishna Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for tls. *arXiv preprint arXiv:1909.00938*, 2019.

[ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[ZSJ+18] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, and William J. Knottebelt. (Short Paper) A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.

Appendix A

Systematization of Cross-Chain State Verification

As described in Section 3.1.5, a critical component of cross-chain communication is the verification of the state "evolution" of a chain X from within another chain Y, i.e., that X is in a certain state after the commit step. In this section we discuss the different elements of the chain that can be verified during the process, to complement the process of verifying state evolution. We show that there is a classification for what is verified (Section A.1), overview existing techniques for each class, and discuss the relation between the verification classes (Section A.2).

A.1 Verification Classes

If a party P on X is misbehaving, it may withhold information from a party Q on Y (i.e., not submit a proof), but it should not be able to trick Q into accepting an incorrect state of L_x (e.g., convince Q that $\textsc{tx}_1 \in \mathsf{L}_x$ although $\textsc{tx}_1$ was never written).

Verification of State. The simplest form of cross-chain verification is to check whether a specific state *exists*, i.e., is reachable but has not necessarily been agreed upon by the consensus participants. A representative example is the verification of Proof-of-Work in merged min-

ing [Bit15, JZS+17]: the child chain Y only parses a given X block and verifies that the hash of the Y candidate block was included, and checks that the PoW hash exceeds the difficulty target of Y. Note that Y does not care whether the block is actually part of L_x. Another example is the use of blockchains as a public source of randomness [BCG15b, BGB17, DFZ16, CDFZ17].

Verification of State Agreement. In addition to the existence of a state, a proof can provide sufficient data to verify that consensus has been achieved on that state. Typically, the functionality of this verification is identical to that of blockchain light clients [Nak08, Bit19, Tec21]: instead of verifying the entire blockchain of X, all block headers and only transactions relevant to the CCC protocol are verified (and stored) on Y. The assumption thereby is that an invalid block will not be included in the verified blockchain under correct operation [Nak08, BKLZ20]. Block headers can be understood as the meta-data for the block, including a commitment to all the transactions in the block, which are typically referenced using a vector commitment [CF13] (or some other form of cryptographic accumulator [BDM93]), e.g. Merkle trees[Mer87]. We discuss how proofs of state agreement differ depending on the underlying consensus mechanism below (non-exhaustive):

- **Proof-of-Work.** To verify agreement in PoW blockchains, a primitive called *(Non-interactive) Proofs of Proof-of-Work* [KLS16, KMZ17], also referred to as SPV (simplified payment verification) [Nak08] is used. Thereby, the verifier of a proof must at least check for each block that (i) the PoW meets the specified difficulty target, (ii) the specified target is in accordance with the difficulty adjustment, and (iii) the block contains a reference to the previous block in the chain [Bit19, ZHL+19]. The first known implementation of cross-chain state agreement verification (for PoW blockchains) is BTCRelay [Con17a]: a smart contract that allows verifying the state of Bitcoin on Ethereum[15].

- **Proof-of-Stake.** If the verified chain uses Proof-of-Stake in its consensus, the proofs represent a dynamic collection of signatures, capturing the underlying stake present in the chain. These are referred to as *Proofs of Proof-of-Stake* (PoPoS) and a scheme in this direction was put forth in [GKZ19b].

[15] Similar contracts have been proposed for other chains [Con17b, Kyb17, Dog21, Kyb19a, Par20, Eve18].

- **BFT.** In case the blockchain is maintained by a BFT committee, the cross-chain proofs are simplified and take the form of a sequence of signatures by $2f + 1$ members of the committee, where f is the number of faulty nodes that can be tolerated [CL+99]. If the committee membership is dynamically changing, the verification process needs to capture the rotating configuration of the committee [NKKJ+].

Sub-linear State Agreement Proofs. Verifying all block headers results in proof complexity linear in the size of the blockchain. However, there exist techniques for achieving *sub-linear* (logarithmic in the size of the chain) complexity, which rely on probabilistic verification. For PoW blockchains, we are aware of two approaches: Superblock (Ni)PoPoWs [Mil12, BCD+14, KLS16, KMZ17] and FlyClient [BKLZ20]. Both techniques rely on probabilistic sampling but differ in the selection heuristic. Superblock (Ni)PoPoWs sample blocks which exceed the required PoW difficulty[16], i.e., randomness is sourced from the unpredictability of the mining process, whereas FlyClient suggests sampling blocks using an optimized heuristic after the chain has been generated (using randomness from the PoW hashes [BCG15b]). For blockchains maintained by a static BFT committee, the verified signatures can be combined into aggregate signatures [KJG+16, KK19] for optimization purposes. These signature techniques are well known and have been invented prior to blockchains, and we hence do not elaborate further on these schemes. In the dynamic setting, skipchains [FGKJ18, NKKJ+, KKGK+16], i.e., double-linked skiplists which enable sub-linear crawling of identity chains, can reduce costs from linear to logarithmic (to the number of configurations). Recently, a number of light client protocols that leverage the compression properties of zero-knowledge proof systems have been proposed [GKO20, GGJ+20, WE20b].

Verification of State Evolution. Once verified by some chain Y that chain X has reached agreement on a ledger state $\mathsf{L}_x[r]$, it is then possible for (users on) Y to verify that certain transactions have been included in L_x. As mentioned, block headers typically reference included transactions via vector commitments. As such, to verify that $\textsc{tx} \in \mathsf{L}_x[r]$ the vector commitment on $\mathsf{L}_x[r]$ needs to be opened at the index of that transaction, e.g. by providing a Merkle tree

[16] It is a property of the PoW mining process that a certain percentage of blocks exceeds or fall short of the required difficulty.

path to the leaf containing TX (e.g. as in Bitcoin). Thereby, multiple transactions can be aggregated in a single proof [ZAPK20].

Verification of State Validity. Even though a block is believed to have consensus, it may not be a valid block if it contains invalid transactions or state transitions (e.g. a PoW block meeting the difficulty requirements, but containing conflicting transactions). Fully validating nodes will reject these blocks as they check all included transactions. However, in the case of cross-chain communication, where chains typically only verify state agreement but not validity, detection is not directly possible. We classify two categories of techniques to enable such chains, and non-full nodes (i.e., light clients), to reject invalid blocks:

- In **proactive** state validation, nodes ensure that blocks are valid before accepting them. Apart from requiring participants to run fully validating nodes, this can be achieved by leveraging "validity proofs" through succinct proofs of knowledge, using systems such as SNARKs [BCCT12], STARKs [BSBHR18] or Bulletproofs [BBB+17]. First schemes for blockchains offering such proofs for each state transition are put forth in [MS18, BBF18, BSCS16]. Informally speaking, this is a "guilty until proven innocent model": nodes assume blocks that have consensus are invalid until proven otherwise.

- In **reactive** state validation, nodes follow an "innocent until proven guilty" model. It is assumed that blocks that have consensus only contain valid state transition unless a state transition "fraud-proof" [ASB18] is created. Fraud proofs typically are proofs of state evolution, i.e., the opening of the transaction vector commitment in the invalid block at the index of the invalid transaction, e.g. via Merkle tree paths. Depending on the observed failure, more data may be necessary to determine inconsistencies (e.g. Merkle paths for conflicting transactions in a double spend).

Verification of Data Availability. Consensus participants may produce a block header, but not release the included transactions, preventing other participants from validating the correctness of the state transition. To this end, verification of state validity can be complemented by verification of data availability. A scheme for such proofs was put forth in [ASB18, YSL+19],

which allows verifying with high probability that all the data in a block is available, by sampling chunks in an erasure-coded version of a block.

A.2 Relation between Verification Classes

Verification of State Agreement requires to first verify a specific state exists or has been proposed (Verification of State). To verify a transaction was included at $\mathsf{L}[r]$ (State Evolution), it is first necessary to verify that the ledger state at $\mathsf{L}[r]$ is indeed agreed upon (State Agreement). Finally, to verify that a state (transition) is indeed valid (State Validity), one must first verify that all associated transactions were indeed included in the ledger (State Evolution). Verification of Data Availability serves as a complementary security measure, and can be added to any of the classes to protect against data withholding attacks. We illustrate this relationship in Figure A.1.

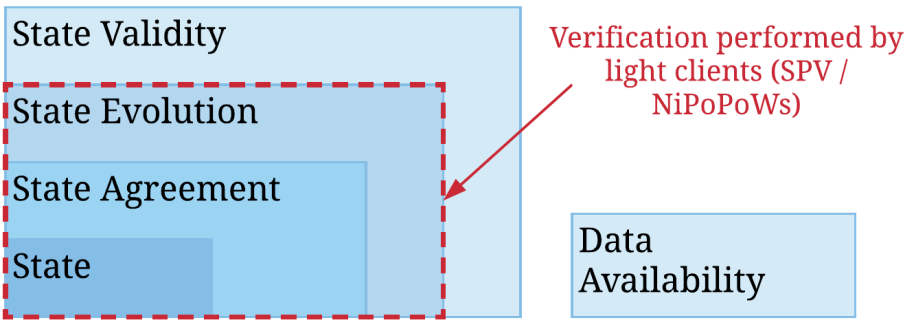


Figure A.1: Venn diagram of cross-chain state verification classes. The red, dotted line highlights the minimum requirement for correctly operating light clients, i.e., SPV/NIPoPoWs/FlyClient in the case of PoW blockchains.

Appendix B

Proof-of-Work Light Client Model

A blockchain *light client*, sometimes referred to as an SPV-Client [Bit19], is a program capable of reading and verifying the state of a blockchain. That is, a light client stores and maintains block headers and allows to verify transaction inclusion proofs.

In the following, we provide a formal model for the requirements of a program π to represent a functioning light client for a proof-of-work blockchain $\mathcal{C}_{pow}$[17].

Notation. We denote $H(x)$ as the output of a cryptographic hash function over some input x. Further, C_i shall denote the block header of the block at position i in the blockchain, represented by the tuple $\langle S_{i-1}, \tau_i, M_i, N_i, t_i \rangle$, where

- S_{i-1} is the reference to the PoW hash (i.e., solution) of the predecessor of block i,
- τ_i is the (expected) PoW difficulty target at block i as defined by consensus rules,
- M_i is the root hash of the Merkle tree of the hashes of all transactions $(\text{TX}_0, \text{TX}_1, ..., \text{TX}_n)$ included in i,
- N_i is the random nonce used to generate the PoW solution hash $S_i = H(C_i)$,
- and t_i is the timestamp specifying when block i was generated.

[17] Verification for alternate structures, such as direct acyclic graphs (DAGs) [SZ18], is analogous. The reduction of DAG to a chain is trivial: a chain is a DAG where each vertex only has one predecessor.

We refer to the header of the first ($i = 0$), so called, *genesis* block as $\mathcal{G}$. We assume protocol rules of C require that the PoW difficulty target τ is adjusted every r blocks based on the relation of the time between each two adjustments and some pre-defined desired block generation rate. Note, while potentially useful for more extensive block validity checks, for simplification we ignore other information usually included in the block headers. Furthermore, as it is not of greater relevance to our model, we assume the same cryptographically secure hash function $H()$ is used to calculate both the hashes of block headers and transactions.

We require a chain relay program π to support the following functionalities with regards to the state of a Proof-of-Work blockchain $\mathcal{C}_{pow}$:

Functionality 1 (Difficulty Adjustment)**.** *Program π has knowledge of the difficulty adjustment rate r and ideal block generation rate and, given C_i and C_{i+r}, where $i \pmod{i} = 0$, outputs the new difficulty target τ_{i+r} according to consensus rules of $\mathcal{C}_{pow}$.*

Functionality 2 (Block Validation)**.** *Program π has a function* `checkBlock` *which takes as input a block header C_i and returns* `True` *if and only if C_i is the pre-image of S_i, τ_i is the difficulty target required at block i and it holds that $S_i \leq \tau_i$.*

Functionality 3 (Chain Validation)**.** *Program π has a function* `checkChain` *which takes as input the genesis block $\mathcal{G}$, a list of consecutive block headers $(C_1, C_2, ..., C_n)$ and returns* `True` *if and only if it holds that $\forall i \leq n$:* `checkBlock`$(C_i) =$ `True` *and for each two consecutive block headers C_i and C_{i+1} it holds that $S_i \in C_{i+1}$, i.e., C_i is the predecessor of C_{i+1}.*

Definition 11 (Valid Chain)**.** *We define the tuple $\langle \mathcal{G}, (C_1, ..., C_n) \rangle$ as a* valid chain *if* `checkChain` *outputs* `True` *given this tuple as input.*

Functionality 4 (Main Chain Detection)**.** *Program π provides a function denoted* `mainChain` *which takes as input two valid chains*
$\langle \mathcal{G}, (C_1, ..., C_i, C_{i+1}, ..., C_n) \rangle$ *and*
$\langle \mathcal{G}, (C_1, ..., C_i, C'_{i+1}, ..., C'_m) \rangle$ *where $n \neq m$ and for every $j \geq i$ it holds that $C_j \neq C'_j$, and outputs the main chain according to the consensus rules of $\mathcal{C}_{pow}$, e.g., the longest chain in the case of Nakamoto consenus.*

Functionality 5 (Transaction Inclusion Verification)**.** *Program π has a function* `checkTransaction` *which, if given a valid chain $\langle \mathcal{G}, (C_1, ..., C_n) \rangle$, a block header C_i, a transaction* TX *and a Merkle tree path p, outputs* `True` *if and only if $H(\mathrm{TX})$ is contained in the Merkle tree with root $M_i \in C_i$ at the position defined by p, $C_i \in (C_1, ..., C_n)$ s.t. $i \leq n$ and the provided chain is the main chain of $\mathcal{C}_{pow}$.*

Definition 12 (Light Client)**.** *A program π is a* light client *of a Proof-of-Work blockchain $\mathcal{C}_{pow}$, if it satisfies Functionalities 1-5 with regards to $\mathcal{C}_{pow}$.*

The presented model focuses on Proof-of-Work blockchains but can be easily extended to other consensus mechanisms, such as Proof-of-Stake-based models, and is left to future work. The modifications required mainly revolve around replacing the PoW difficulty adjustment verification with knowledge of the qualifying signers at each signing epoch, and the verification of signatures rather than the PoW target τ.