SoK: Communication Across Distributed Ledgers

Alexei Zamyatin*†, Mustafa Al-Bassam‡, Dionysis Zindros§††, Eleftherios Kokoris-Kogias¶, Pedro Moreno-Sanchez‖, Aggelos Kiayias**††, and William J. Knottenbelt*
*Imperial College London
†SBA Research
‡University College London
§University of Athens
¶Ecole polytechnique fédérale de Lausanne
‖TU Wien
**University of Edinburgh
††IOHK

***Abstract*—Communication across distributed systems, each running its own consensus, is a problem previously studied under the assumption of trust across systems. With the appearance of distributed ledgers or blockchains, numerous protocols have emerged, which attempt to achieve trustless communication between distrusting ledgers and participants. Cross-chain communication thereby plays a fundamental role in cryptocurrency exchanges, sharding, bootstrapping and extension of distributed ledgers. Unfortunately, existing proposals are designed ad-hoc for specific use-cases, making it hard to gain confidence on their correctness and to use them as building blocks for new systems.**

We provide the first systematic exposition of protocols for cross-chain communication. First, we formalize the underlying research problem and show cross-chain communication is *impossible without a trusted third party*, contrary to common beliefs in the blockchain community. We then develop a framework for evaluating existing and designing new cross-chain protocols, based on use case, trust model and security assumptions of interlinked blockchains. Finally, we identify security and privacy challenges faced by protocols in the cross-chain setting.

This Systematization of Knowledge (SoK) offers a comprehensive guide for designing protocols bridging the numerous distributed ledgers available today and aims to facilitate clearer communication between academia and industry in this field.

***Index Terms*—blockchains, distributed ledgers, cross-chain communication, interoperability**

1. Introduction

Why Cross-Chain Communication is Worthy of Research. Since the introduction of Bitcoin [135] as the first decentralized ledger currency in 2008, the topic of blockchains (or distributed ledgers) has evolved into a well studied field in both industry and academia. Nevertheless, developments are still largely driven by community effort, resulting in a plethora of blockchain-based digital currencies being created. Taking into account the heterogeneous nature of these systems in terms of design and purpose, it is unlikely that there shall emerge a "coin to rule them all", yielding interoperability an important research problem. Thereby, cross-chain communication is not only found in currency transfers and exchanges [18], [19], [89]–[91], but is a critical component of scalabilty solutions (synchronization in sharded systems [22], [23], [26], [112], [168]), feature extensions (sidechains [36], [81], [108], [120] and cryptocurrency-backed assets [159], [169]), as well as bootstrapping of new and migration between existing systems [3], [50], [100], [153]. In addition, numerous competing projects aiming to create a single platform for cross-chain communication have recently emerged [20], [93], [116], [152], [160], [161], [163]. However, in spite of the vast number of use cases and solution attempts, the underlying problem of cross-chain communication has neither been clearly defined, nor have the associated challenges been studied or related to existing research.

Historical Background and Difference to 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) [44], 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) [35]. 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 the 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*.

Contributions. In this work, we provide the first systematic analysis of communication across distributed ledgers. First, we overview the process of Cross-Chain Communication (CCC) and identify the main designs patters for

Contact email: azamyatin@imperial.ac.uk

protocols in CCC (Section 2). Next, we formalize the underlying problem of Correct Cross-Chain Communication and show it is *impossible without a trusted third party* by providing a reduction to the Fair Exchange problem (Section 3). We then introduce a framework to build new and evaluate existing CCC protocols. In particular, we systematize mechanisms used in committing, aborting and verifying in CCC, and analyze the inherent trust assumptions thereof (Sections 4- 5). Following this framework, we introduce a classification of CCC protocols proposed in literature (Section 6). Finally, we discuss implications for the blockchain, network, and threat models, and examine transcendence for privacy (Section 7). We overview related work in Section 8 and conclude in Section 9.

2. Overview of Cross-Chain Communication

In this section, we sketch the cross-chain communication (CCC) process and the design rationales of existing protocols. We defer the formal description of the CCC problem to Section 3.

2.1. The Cross-Chain Communication Process

The CCC process defines the interaction between two blockchains X and Y to synchronize transactions, each of which represents the transfer of goods, assets or objects. CCC can be compartmentalized into four phases.

Setup. The setup phase is required to establish the parameters for the rest of the CCC process. For example, in the case of an exchange of assets of digital goods, parties agree on the CCC protocol type, asset amount, and any additional conditions. Typically, this process occurs out-of-band and we hence shift our focus on the actual execution of CCC in the rest of this paper.

Commit on X. Upon a successful setup, a publicly verifiable commitment to execute the CCC protocol is published on X- typically a transaction being included in the blockchain. In case of an asset exchange, this could be a user locking assets, to be released upon fulfillment of some conditions (e.g. an event on chain Y).

Verify on Y. In this phase, the correctness of the commitment on X is verified on Y, as per the parameters agreed in the setup phase.

Commit (or Abort) on Y. If the verification succeeds, a commitment as agreed in the setup phase is published on Y. Following our exchange example, this could be a user triggering the prepared exchange of assets. Alternatively, the CCC can be aborted at this stage. In this case, the commitment on X can also be reverted, e.g. returning locked coins to their original owner on X, in our example.

2.2. Protocols for Cross-Chain Communication

We observe two main design patterns for protocols in CCC: (i) protocols which synchronize (atomic) exchanges of digital goods between chain X and chain Y; and (ii) protocols that solely transfer assets/objects from a chain X to chain Y. An explanatory visualization for both rationales is provided in Figure 1. In this section, we overview these two protocol families and defer a systematic classification of individual protocols to Section 6.

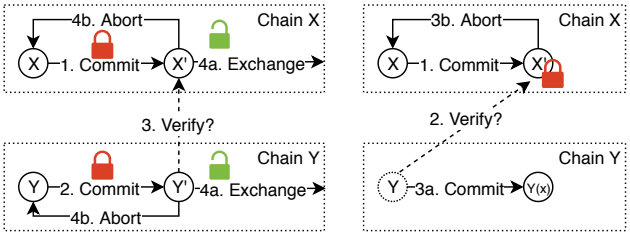

Figure 1: Difference between *bidirectional exchange* (left) and *one-way transfer* (right) CCC protocols. In exchange protocols, two assets/objects x and y are locked (and unlocked atomically if required). In transfers, a representation $y(x)$ of an asset/object x is created on y, after a "write" lock on x was obtained on X.

2.2.1. Bidirectional Exchanges. The family of bidirectional exchange protocols, as state in the name, encompasses those protocols to synchronize an atomic swap of two (or more) digital goods (assets or objects): x of chain X and y on Y. In practice, such protocols typically implement some form of a two-phase commit mechanism [18], [91], [108], where the parties can explicitly abort the exchange in case of a disagreement or failure during the commit step.

Interestingly, bidirectional exchange protocols are *blockchain agnostic* in that they do not require explicit involvement of the consensus participants of interlinked chains X and Y. That is, the only assumption is that participants of the CCC will be able to include transactions in the underlying ledgers. Notable examples are atomic cross-chain swaps which use HTLCs [18], [19], [90], [123] or similar symmetric cryptographic locks [46], [73], [124], [134], [140], [157] (cf. Section 4), and centralized, custodial exchange protocols [17], [41], [160].

2.2.2. One-way Transfers. The family of one-way transfers protocols consists of protocols that move assets/objects from one blockchain to another. Typically, this is achieved by obtaining a "write lock" on an asset/object x on chain X, i.e., preventing any further updates of x on X, and creating a *representation* $y(x)$ on Y. Now, the state of x can only be modified by modifying $y(x)$, comparable to the concept of *mutual exclusion* in concurrency control [67]. The state changes of $y(x)$ can also be reflected back to X by locking or destroying $y(x)$ and applying the updates to x when it is unlocked. This is, for example, the case for cryptocurrency-backed assets [17], [120], [159], [169], where changes in asset ownership executed on the issuing chain Y are propagated to the backing chain X. Although less common than in bidirectional exchanges, one-way transfer protocols may implement an explicit abort procedure, if the creation of $y(x)$ fails.

Contrary to bidirectional exchanges, one-way transfer protocols require explicit involvement of the consensus participants of interlinked chains X and Y, specifically in the verification step. In fact, we can classify one-way transfers into *homogeneous* and *heterogeneous*, two subfamilies defined upon the required blockchain security model.

Homogeneous homogeneous one-way transfer protocols *explicitly* make assumptions regarding the threat and network models of Y when constructing a communication

protocol from X. In practice, this is the case in sharding [23], [25], [112], [168], where every shard is considered to be secure by design [112], [168] leveraging, for example, an appropriate form of randomness [146], [155]. As a result, shards have "uniform" security, i.e., it is equally difficult for an adversary to compromise any shard. Following, if one shard is corrupted, the failure typically occurs system wide. This further applies to (child) blockchains which have hierarchical dependency to some parent chain, i.e., the child chain's security depends on that of the parent (e.g., merged mining [3], [100], [120], [156]). Since, if the parent fails, so does the child, we are forced to assume the parent's security holds when communicating with it from the child. Note that this relation does not necessarily hold inversely: by definition, a child can fail without affecting the parent (contrary to the critical impact of a failing shard) [36], [80].

Heterogeneous. In the design of heterogeneous one-way transfer protocols, interlinked chains cannot be assumed to have identical rule sets/data structures. In practice, it can be understood that *any user can create their own chain* and, in the absence of pre-defined specifications or rules ensuring e.g. sufficient honest participants exist, *no generic security assumptions can be relied upon*. Hence, a chain, even if capable of verifying the consensus rules of another chain, cannot be sure that the received information is indeed valid (except if it is fully validated, cf. Section 5.1) [36]. As a result, CCC protocols in the heterogeneous model typically make additional assumptions regarding interlinked chains, restricting applicability to a subset of existing systems [108] or introducing additional economic incentives [169].

3. The Cross-Chain Communication Problem

We proceed to formally define the problem of Correct Cross-Chain Communication (CCC) and provide a reduction to the Fair Exchange problem [29], [137], showing that CCC is *impossible without a trusted third party*.

3.1. Preliminaries

In literature, the terms blockchain and *distributed ledger* are often used as synonyms. We adopt this nomenclature as we proceed to introduce some notation, on the basis of [81] with minor alterations. We refer to [53], [78], [135] for reading on the basics of distributed ledgers.

When speaking of cross-chain communication, we consider the interaction between two distributed systems X and Y, which can have distinct consensus participants and may employ different agreement protocols. Thereby, it is assumed the majority of consensus participants in both X and Y are honest[1]. The data structures underlying X and Y are *blockchains* (or *chains*), i.e., append-only sequences of blocks, where each block contains a reference to its predecessor(s), e.g. via a hash.

Ledgers and State Evolution. We denote a ledger as L (L_x and L_y respectively) and define its *state* as the dynamically evolving sequence of included transactions $\langle \mathtt{TX}_1, ..., \mathtt{TX}_n \rangle$. We assume that the evolution of the ledger state progresses in discrete *slots* s, indexed[2] by a natural number $i \in \mathbb{N}$. At each slot i, a new set of transactions (included in a newly generated block) is written to the ledger L. We use $\mathsf{L}[i]$ to denote the state of ledger L at slot i, i.e., after applying all transactions *written* to the ledger since slot $i-1$. A transaction can be written to L only if it is consistent with the system's consensus rules, given the current ledger state $\mathsf{L}[i]$. This consistency is left for the particular system to define, and we describe it as a free predicate $\mathsf{valid}(\cdot)$. In practice, the ordering at which transactions are included in the ledger is crucial for their validity, i.e., $\mathtt{TX}_j$ at position j can conflict with $\mathtt{TX}_l$ at position l (e.g. a double-spend). Depending on whether $j > l$, either $\mathtt{TX}_j$ or $\mathtt{TX}_l$ is valid and can be included in L, but not both. For simplicity, we assume correct ordering implicitly, and write $\mathsf{valid}(\mathtt{TX}, \mathsf{L}_x[i])$ (instead of explicitly specifying the transaction index as $\mathsf{valid}(\mathtt{TX}, \mathsf{L}_x[i][j])$).

Global Clock. 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* s_x slot, L_y may, for example, progress *forty* s_y slots (on average, as in the case of Bitcoin [135] and Ethereum [56]). To capture the order of transactions when synchronizing across L_x and L_y, we introduce a clock function τ which maps a given slot on any ledger to the time on a global, synchronized clock $\tau : s \to t$. For simplicity and readability, we use this conversion implicitly, i.e., given $\tau(i) = t$ we write $\mathsf{L}[t] = \mathsf{L}[i]$ instead of $\mathsf{L}[t] = \mathsf{L}[\tau(i)]$.

Persistence and Liveness. Each participant P adopts and maintains a local ledger state $\mathsf{L}^P[t]$ at time t ($\mathsf{L}_x^P[t]$ and $\mathsf{L}_y^P[t]$ respectively), i.e., her current view of the ledger. The views of two distinct participants P and Q on the same ledger L may differ at time t (e.g., due to network delay or message loss): $\mathsf{L}^P[t] = \mathsf{L}[i]$, $\mathsf{L}^Q[t] = \mathsf{L}[i-1]$, $\mathsf{L}^P[t] \neq \mathsf{L}^Q[t]$. 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 [78]:

Definition 1 (Liveness)**.** *Consider an honest party P of a ledger L and a* liveness delay parameter *u. If P attempts to write a transaction* $\mathtt{TX}$ *to its ledger at time $t \in \mathbb{N}$, then* $\mathtt{TX}$ *will appear in its ledger at time t', i.e., $\exists t' \in \mathbb{N} : t' \geq t \wedge \mathtt{TX} \in \mathsf{L}^P[t']$. Additionally, we require the interval $t' - t$ to be* upper bound *by u.*

Definition 2 (Persistence)**.** *Consider two honest parties P, Q of a ledger L and a* persistence delay parameter *k. If a transaction* $\mathtt{TX}$ *appears in the ledger of party P at time t, then it will eventually appear in the ledger of party Q at a time $t' > t$. Concretely, for all honest parties P and Q, we have that $\forall t \in \mathbb{N} : \exists t' \geq t : \mathsf{L}^Q[t'] = \mathsf{L}^P[t]$. Additionally, we require that the interval $t' - t$ is* upper bound *by k.*

(Smart) Contracts and State Manipulation. A transaction $\mathtt{TX}$, 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 mere hash functions [52] to (near) Turing complete programming languages [164].

1. Or, in the case of Proof-of-Work blockchains such as Bitcoin, the majority of computational power [135]

2. In practice often referred to as the blockchain *height*.

Assets/Objects. We denote assets/objects on ledgers L_x and L_y as x and y, respectively. Both assets and objects can have a state, which is encoded as part of the state of the underlying ledger L, i.e., in the evolving set of included transactions associated with the asset/object (for example, ownership).

Full nodes and Light Clients. We differentiate between two types of nodes in the networks: (i) those that store the entire transaction history and *validate* the full ledger state (*full nodes*), (ii) and those that only store transactions relevant to them and *verify* their inclusion in the underlying ledger, but do not check their validity (*light clients*).

3.2. Model and Specification

Model. Consider two independent distributed systems X and Y with underlying ledgers L_x and L_y, as defined in Section 3.1. We assume a *closed* system model as in [118] with a process P running on X and a process Q running on Y. The only way a process can influence the state evolution of the underlying system is by (i) writing a transaction $\mathtt{TX}$ to the underlying ledger L (commit), or (ii) by completely stopping to interact with the system (abort). We assume that P possesses transaction $\mathtt{TX}_P$, which can be written to L_x, and Q possesses $\mathtt{TX}_Q$, 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). Both processes posses descriptions d_P and d_Q of the transactions $\mathtt{TX}_P$ and $\mathtt{TX}_Q$ they expect from their counterparty. Note: $d_P = desc(\mathtt{TX}_P)$ implies $\mathtt{TX}_P$ is valid in X (at time of synchronization), as it cannot be written to L_x otherwise (analogous for d_Q).

We assume P and Q know each other's identity and no (trusted) third party is involved in the communication between the two processes. Further, we assume no bounds on message delay or deviations between local clocks and treat failure to communicate as adversarial behavior. Hence, if P or Q become malicious, we indicate this using boolean "error variables" [79] m_P and m_Q.

Specification. The goal of cross-chain communication can be described as the synchronization of processes P and Q such that Q *writes* $\mathtt{TX}_Q$ *to* L_y *if and only if* P *has written* $\mathtt{TX}_P$ *to* L_x. Thereby, it must hold that $desc(\mathtt{TX}_P) = d_Q \wedge desc(\mathtt{TX}_Q) = d_P$. The intuition is that $\mathtt{TX}_P$ and $\mathtt{TX}_Q$ are two transactions which must either both, or neither, be included in L_x and L_y, respectively. For example, they can constitute an exchange of assets which must be completed atomically. A visual representation is provided in Figure 2.

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

Definition 3 (Effectiveness)**.** *If both P and Q behave correctly and $\mathtt{TX}_P$ and $\mathtt{TX}_Q$ match the expected descriptions (and are hence valid), then $\mathtt{TX}_P$ will be included in L_x and $\mathtt{TX}_Q$ will be included in L_y. If either of the transactions are not as expected, then both parties abort.*

$$(desc(\mathtt{TX}_P) = d_Q \wedge desc(\mathtt{TX}_Q) = d_P \wedge m_P = m_Q = \bot \implies \mathtt{TX}_P \in \mathsf{L}_x \wedge \mathtt{TX}_Q \in \mathsf{L}_y) \wedge (desc(\mathtt{TX}_P) \neq d_Q \vee desc(\mathtt{TX}_Q) \neq d_P \implies \mathtt{TX}_P \notin \mathsf{L}_x \wedge \mathtt{TX}_Q \notin \mathsf{L}_y)$$

Definition 4 (Atomicity)**.** *There are no outcomes in which P writes $\mathtt{TX}_P$ to L_x but Q does not write $\mathtt{TX}_Q$, or Q writes $\mathtt{TX}_Q$ to L_y but P did not write $\mathtt{TX}_P$ to L_x.*

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

Definition 5 (Timeliness)**.** *Eventually, a process that behaves correctly will write a transaction $\mathtt{TX}$ to its ledger L.*

From Persistence and Liveness of L, it follows that eventually P writes $\mathtt{TX}_P$ to L_x and Q becomes aware of and verifies $\mathtt{TX}_P$.

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 $\mathtt{TX}_P$ and $\mathtt{TX}_Q$. Then a* correct *cross-chain communication protocol is a protocol which achieves $\mathtt{TX}_P \in \mathsf{L}_x \wedge \mathtt{TX}_Q \in \mathsf{L}_y$ 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 transaction match descriptions and both processes are behaving correctly. Atomicity globally restricts the outcome to exclude behaviors which place a disadvantage on either process. Timeliness guarantees eventual termination of the protocol, i.e., is a liveness property.

3.3. Correct Cross-Chain Communication is as Hard as Fair Exchange

We proceed to show that Correct Cross-Chain Communication is as hard as Fair Exchange, and hence impossible without a trusted third party.

Fair Exchange. The fair exchange of digital goods is a well-studied problem in computer science. 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 [30]. Fair exchange can be considered a subproblem of fair secure computation [42], and is known to be impossible without a trusted third party [137], [165].

Relating CCC to Fair Exchange. We proceed to show that Correct Cross-Chain Communication is impossible without a trusted third party (TTP) by reducing it to Fair Exchange [29], [30], [137].

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

Proof. Consider that the two processes P and Q are parties in a fair exchange. Specifically, P owns an item i_P and wishes to exchange it against an item i_Q owned by Q. Assume $\mathtt{TX}_P$ assigns ownership of i_P to Q and $\mathtt{TX}_Q$ ownership of i_Q to P ("descriptions" of $\mathtt{TX}_P$ and

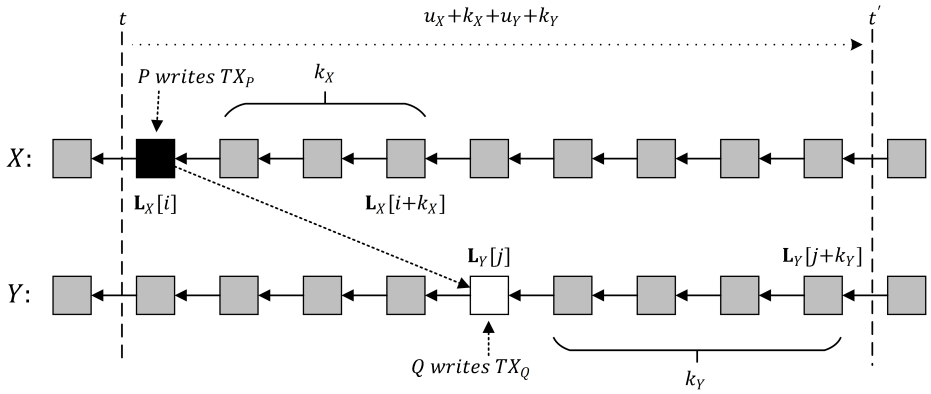

Figure 2: Simplified visualization of CCC between X and Y. Process Q writes $\mathtt{TX}_Q$ only if P has written $\mathtt{TX}_P$. We use k_X and k_Y to denote exemplary persistence delays of X and Y ($k_X = 3$, $k_Y = 4$), and set $u_x = u_y = 0$ for the liveness delays for the sake of simplicity.

$\mathtt{TX}_Q$). Then, $\mathtt{TX}_P$ must be included in L_x and $\mathtt{TX}_Q$ in L_y to finalize the exchange. In other words, if $\mathtt{TX}_Q \in \mathsf{L}_y$ and $\mathtt{TX}_P \in \mathsf{L}_x$, then P receives desired i_Q and Q receives desired i_P, i.e., P and Q fairly exchange i_P and i_Q.

A fair exchange protocol must fulfill three properties: *Effectiveness*, *(Strong) Fairness* and *Timeliness* [29], [137]. It is easy to see the Timeliness property of CCC is equivalent to Timeliness of fair exchange as defined in [137]. Effectiveness in fair exchange states that if P and Q behave correctly and do not want to abandon the exchange, and items i_P and i_Q are as expected by Q and P, then at the end of the protocol, P will have the desired i_Q and Q will have the desired i_P [137]. It is easy to see Effectiveness in CCC achieves exactly this property: if P and Q behave correctly and the transaction match the respective descriptions (i.e., assign i_P to Q and i_Q to P), then P will write $\mathtt{TX}_P$ to L_y and Q will write $\mathtt{TX}_Q$ to L_x, resulting in P receiving i_Q and Q receiving i_P. 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, and consequently appear in the local ledgers of all other participants of X and Y. Strong Fairness in fair exchange states that there is no outcome of the protocol, where P owns i_Q but Q does not own i_P, or Q owns i_P but P does not own i_Q [137]. In our setting, such an outcome is only possible if $\mathtt{TX}_P \in \mathsf{L}_x \wedge \mathtt{TX}_Q \notin \mathsf{L}_y$ or $\mathtt{TX}_P \notin \mathsf{L}_x \wedge \mathtt{TX}_Q \in \mathsf{L}_y$, which contradicts the Atomicity property of CCC. □

We assume the same asynchronous (explicitly) and deterministic (implicitly) system model (cf. Section 3.2) as [76], [137]. Since P and Q can simply stop participating in the CCC protocol, we also have the same crash failure model as [76], [137]. Hence, we conclude:

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

Proof. Assume there exists a asynchronous protocol C which solves CCC. Then, due to Lemma 1 there exists a protocol which 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 [76], [137]. In the open model, P and Q can be forced to make a decision by the system, i.e., transactions can be written on their behalf if they crash [112]. This can be achieved by leveraging smart contracts, similar to blockchain-based fair exchange protocols, e.g. [71]. As such, we can construct a smart contract on Y which will include $\mathtt{TX}_Q$ in L_y as soon as P includes $\mathtt{TX}_P$ in L_x, i.e., Q is allowed to crash. A contract, however, can only perform actions based on some prior input. Specifically, before writing $\mathtt{TX}_Q$ the contract must receive proof that $\mathtt{TX}_P$ was included in L_x. A protocol achieving CCC must hence assume (i) P (or Q) will always submit a proof showing $\mathtt{TX}_P \in \mathsf{L}_x$, i.e., introduce some form of *synchrony* assumptions, or (ii) resort to a TTP. We argue introducing a TTP is equivalent to introducing synchrony assumptions, as discussed in [137], and derive the following corollary:

Corollary 1. *There exists no* CCC *protocol tolerant against misbehaving nodes without a trusted third party.*

Proof. A trusted third party is equivalent to introducing some form of synchrony. As such, if there is no trusted third party, then the protocol is asynchronous. Theorem 1 now proves Corollary 1. □

The intuition behind this result is as follows.

If we assume that process P does not crash and hence submits the necessary proof the 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 represents its own trusted third party. However, if we cannot make assumptions on when the message will be delivered to the smart contract, a trusted third party is necessary to determine the outcome of the CCC: the TTP observes $\mathtt{TX}_P \in \mathsf{L}_x$ and informs the smart contract or directly enforces the inclusion of $\mathtt{TX}_Q$ in L_y. Thereby, we observe similarities to the concept of *failure detectors* [63], a construction used to introduce an implicit notion of time into distributed systems to solve consensus. In the context of fair exchange, and hence CCC, a failure detector, which can also be considered a (trusted) third party to the protocol, indicates whether a participate has failed or misbehaved.

3.4. 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 [27], [42], [71], [107], [111], [114]. 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. As long as the majority of consensus participants are honest, correct behavior of processes/participants of the fair exchange is enforced – typically, the correct release of i_Q to P if Q received i_P.

A CCC protocol aims to achieve synchronization between *two* such distributed ledgers, both of which are inherently trusted to operate correctly. Here, a (possibly additional) TTP can be used to (i) confirm to the consensus participants of Y that $\mathtt{TX}_P$ was included in L_x, who in turn enforce the inclusion of $\mathtt{TX}_Q$ in L_y; or (ii) directly enforce correct behavior of Q, such that $\mathtt{TX}_Q \in \mathsf{L}_y$.

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 can enforce correct behavior of the participants. In more detail, 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 the latter is, without question, more resilient against failures).

3.5. Incentives and Economically Trustless CCC

Several workarounds have been proposed in literature to counter the fair exchange problem. Most prominent alternatives include gradual release mechanisms, optimistic models, and partially fair secure computation [30], [42], [60], [115]. These workarounds suffer, among others, from a common drawback: they require a trusted party that does not collude with the adversary. Further, when an adversary aborts, the honest parties have to spend extra efforts to restore fairness, e.g., the trusted server in the optimistic model needs to be contacted each time fairness is breached. 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. This has paved the way to design economically trustless CCC protocols that follow a game theoretic model under the assumption that actors behave rationally [169]. We remark nevertheless that such assumption only hold if incentives are aligned and players are economically rational. Malicious/altruistic actors can still breach CCC properties (e.g., even if there is no economic damage, the correct execution of the communication itself still fails).

4. Commit and Abort in CCC

We now overview techniques for implementing the different phases of CCC protocols, discussing ways of implementing the necessary TTP.

The commit and abort steps, as described in Section 2, go hand by hand in the CCC process. Intuitively, the commit step is crucial to freeze an asset x in chain X and provide a time window to be verified by participants at Y according to the agreement reached at the setup. This time window must be, however, limited to ensure that x can be further used by the original owner if the CCC ends in unsuccessful exchange or transfer of assets. The abort step is thus crucial to limit the aforementioned time window.

4.1. Commit Phase

First, we discuss the different techniques to handle the commit phase of CCC protocols. We vertebrate this discussion on the assumption upon which the commit phase is designed: (i) relying on a TTP; (ii) relying on CCC participants and assuming synchronous communication among them; and (iii) combinations thereof.

4.1.1. Trusted Third Party (Coordinators). A *coordinator* (also referred to as *vault* [169]) is a TTP that assists other protocol participants in achieving agreement to either commit or abort the cross-chain transfer.

We can describe the coordinator types attending to two criteria: *custody of assets* and *involvement in blockchain consensus*. We first introduce both classification criteria and then detail our classification of coordinators.

Custody of Assets. Custody refers to where the control over assets of (honest) participants resides and we can 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. It is possible to mitigate this trust assumption by introducing collateral (i.e., a deposit of coins from the custodian) and penalties if the custodian misbehaves [86], [87], [158], [159], [169].
- **Escrows** receive control over the participant's funds *conditional* to certain prearranged constraints being fulfilled. The release of the assets depend thus on that the underlying chain correctly verifies the fulfillment for the condition whereas the Escrow can only fail to take action (e.g., crash). Moreover, from the game theoretic perspective, Escrows are expected to lose utility from misbehaving and are hence often referred to as "untrusted" third parties in the blockchain community.

Involvement in Blockchain Consensus. Coordinators can optionally also take part in the blockchain consensus protocol. We hence differentiate between *consensus-level* and *external* coordinators.

- **Consensus-level** coordinators refer to, as the name states, TTPs that are additionally consensus participants of 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.

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

- **External Custodians (Committees).** In practice, instead of relying on the availability and honest behavior of a single external coordinator, trust assumptions can also 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 [61]. An important distinction to make here is between *static*, i.e., unchanged over time (usually permissioned), and *dynamic* committees, where a pre-defined mechanism is responsible for member election. The latter is a well studied problem, e.g. in Proof-of-Work [66], [109], [110], [139] and Proof-of-Stake [43], [65], [106], [131] blockchains. Practical examples for such CCC protocols relying on committees include [13], [68], [112].
- **Consensus-level Custodian (Consensus Committee)** Consensus-level custodians are identical to external custodians, except that they are also responsible for agreeing on the state of the underlying ledger. Often, this technique is used if the blockchain on which the commit step is executed runs a BFT consensus protocol and there hence already exists a static committee of consensus participants. The later can be reused as the TTP in CCC: the trust in the honest behavior of the committee implicitly stems from the assumption that the underlying ledger operates correctly. Examples include sharded blockchains, such as [23], [112], [116], [163].
- **External Escrows (Multisignature Contracts).** External Escrows can be seen as a special case of committees (i.e., External Custodians) where the coordinator is transformed from Custodian to Escrow by means of a *multisignature contract*. Multisignature contracts require the signature of the participant P (i.e., the asset owner) in addition to those of the (subset of) committee members, i.e., $P+M, M \leq N$. The committee can thus only execute actions pre-authorized by the participant and can at most freeze assets, but not commit theft.
- **Consensus-level Escrow (Smart Contracts)** are programs stored in a ledger which are executed and their result agreed upon by consensus participants [56], [59]. 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. Depending on the system properties, smart contracts can be (near) Turing complete [56], or limited to a subset of operations [135], [148]. In addition, smart contracts can be used to collateralize CCC participants, penalize misbehavior [86], [87], [158] or pay premium for correct participation [85] – even if the participants are located on alternate chains, potentially without support for smart contracts themselves [169].

4.1.2. Synchrony Assumptions (Lock Contracts). An alternative to coordinators consists in relying on synchronous communication between participants and leveraging locking mechaniusms which stemm security from cryptographic hardness assumptions. Observations in practice show these techniques are typically used in bidirectional exchange CCC protocols implementing two-phase commit, where the same (*symmetric*) locks are created on both chains and released atomically. We provide an overview, differentiating between the cryptographic primitives relied upon.

- **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 [18], [19], [90], [123].
- **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 script 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 is most blockchains today [46], [48], [73], [124], [134], [140], [157].
- **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 [142] build upon inherently sequential functions where the result is only revealed after a predefined number of operations are performed. Verifiable delay functions [46] improve upon timelock puzzles on 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 [41].

4.1.3. Hybrid (Watchtowers). Instead of fully relying on coordinators being available or synchrony assumptions among participants holding, it is possible to it is possible to employ so called *watchtowers*, i.e., trusted third parties / service providers which act as a fallback if CCC participants experience crash failures. 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 [32], [34], [103], [125]. This construction was first introduced and applied to off-chain payment channels [84].

4.2. Abort Phase

We now discuss different techniques to handle the abort phase of CCC protocols. We organize our discussion based on the same assumptions upon which the commit phase is designed: (i) relying on a TTP; (ii) relying on CCC participants and assuming synchronous communication among them; and (iii) combinations thereof.

4.2.1. Trusted Third Party (Coordinators). Similarly to the commit phase, an abort can be handled by a trusted

third party. Thereby, the TTP must be the same TTP which executed the commit step of the CCC protocol - as such, the possible techniques are the same as described in Section 4.1.1.

4.2.2. Synchrony Assumptions (Timelocks). Alternatively, it is possible to enforce synchrony by introducing timelocks, after the expiry of which the protocol is aborted. 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 complimented with *timelocks*: after expiry of the timelock, assets are returned to their original owner. Thereby, we differentiate between two types of timelocks:

- **Absolute timelocks**, where a transaction becomes valid only after a certain point in time, defined in by a timestamp or a block (ledger at index i, $\mathsf{L}[i]$) located in the future.
- **Relative timelocks**, where a transaction $\mathtt{TX}_2$ becomes valid only after a given time value or number of *confirmations* [5] have elapsed since the inclusion of another transaction $\mathtt{TX}_1$ in the underlying ledger. For example, assuming transaction $\mathtt{TX}_1$ was included in $\mathsf{L}[i]$, then relatively timelocked $\mathtt{TX}_2$ becomes valid when the chain reaches ledger state $\mathsf{L}[j]$ where $j = i+c$, with c denoting the number of required confirmations ($i, j, c \in \mathbb{N}$). Typically, $\mathtt{TX}_1$ and $\mathtt{TX}_2$ are related as $\mathtt{TX}_2$ spends assets transferred in $\mathtt{TX}_1$ [141]. While more practical than absolute timelocks (no need for external clock), as of this writing, we are not aware of schemes allowing the creation of relative timelocks across ledgers.

4.2.3. Hybrid (Watchtowers). After expiry of a timelock, the CCC protocol is aborted. However, participants typically must be online to regain control over the assets locked as part of the CCC commit phase. In most cases, one-way transfer CCC protocols do not introduce an upper bound on the delay until funds must be recovered from the commit (lock) is introduced. However, in bidirectional exchange protocols, where timelocks are often used to prevent both parties' funds from being frozen indefinitely, a timely recover may be necessary. Thereby, participants must come online within some pre-defined period or entrust a trusted third party, e.g. a watchtower [32], [34], [103], [125], with the recovery of the locked assets. This can be the case in HTLC atomic swaps [2], [19], [90], [141], when either party crashes after the secret used in the hash lock has been revealed.

5. Cross-Chain State Verification

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. We present a classification based on the scope of the verification process, i.e., differentiating between what is being verified (Section 5.1) and discuss the relation between these verification classes (Section 5.2). We then overview implementation techniques for cross-chain state verification, differentiating, as in previous sections, between (i) using a TTP, (ii) relying on verification by participants, (iii) and combinations thereof (Section 5.3).

5.1. 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 $\mathtt{TX}_1 \in \mathsf{L}_x$ although $\mathtt{TX}_1$ 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 mining [3], [100]: 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 [51], [55], [64], [70].

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 [1], [11], [135]: 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 [122], [135]. 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 [62] (or some other form of cryptographic accumulator [39]), e.g. Merkle trees [129]. 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* [104], [105], also referred to as SPV (simplified payment verification) [135] 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 [1], [169]. The first known implementation of cross-chain state agreement verification (for PoW blockchains) is BTCRelay [4]: a smart contract which allows to verify the state of Bitcoin on Ethereum[3].
- **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 [81].
- **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 $2f + 1$

3. Similar contracts have consequently been proposed for other chains [6], [7], [10], [12], [14], [15].

members of the committee, where f is the number of faulty nodes that can be tolerated [61]. If the committee membership is dynamically changing, the verification process needs to capture the rotating configuration of the committee, which can incur significant cost for parties that rarely synchronize.

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 [36], [104], [105], [132] and FlyClient [122]. Both techniques rely on probabilistic sampling but differ in the selection heuristic. Superblock (Ni)PoPoWs sample blocks which exceed the required PoW difficulty[4], i.e., randomness is sourced from the unpredictability of the mining process, whereas FlyClient suggests to sample blocks using an optimized heuristic after the chain has been generated (using randomness from the PoW hashes [51]). Such probabilistic sampling techniques may be potentially applicable to Proof-of-Stake blockchains [122], however, to the best of our knowledge, no concrete schemes have been put forth at the time of writing. For blockchains maintained by a static BFT committee, the verified signatures can be combined into aggregate signatures [109], [110] 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 [77], [113], [136], 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).

Verification of State Evolution. Once verified by some chain Y that chain X has reached agreement on a ledger state $\mathsf{L}_x[i]$, it is then possible for (users on) Y to verify that certain transactions have been included in L_x (and hence taken place on X). As mentioned, block headers typically reference included transactions via vector commitments. As such, to verify that $\mathtt{TX} \in \mathsf{L}_x[i]$ the vector commitment on $\mathsf{L}_x[i]$ needs to be opened at the index of that transaction, e.g. by providing a Merkle tree path to the leaf containing $\mathtt{TX}$ (e.g. as in Bitcoin).

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 [45], STARKs [37] or Bulletproofs [54]. First schemes for blockchains offering such proofs for each state transition are put forth in [38], [47], [128]. 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" [24] is created. Fraud proofs typically are proofs of state evolution, i.e., 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 tree paths for conflicting transactions in case of 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 complimented by verification of data availability. A scheme for such proofs was put forth in [24] and extended in [167], which allows to verify with high probability that all of the data in a block is available, by sampling chunks in an erasure-coded version of a block.

5.2. 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 $\mathsf{L}[i]$ (State Evolution), it is first necessary to verify that the ledger state at L [i] is indeed agreed upon (State Agreement). Finally, to verify that a the 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 complimentary security measure, and can be added to any of the classes to protect against data withholding attacks. We illustrate this relationship in Figure 3.

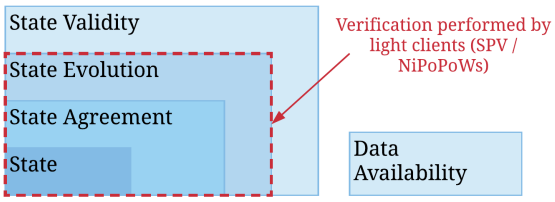

Figure 3: Venn diagram visualizing the relation of cross-chain state verification classes. The red dotted line highlights the minimum requirement for correctly operating light clients, i.e., verifying SPV / NiPoPoWs in the case of PoW blockchains.

5.3. Implementation Techniques

We discuss implementations of cross-chain verification, differentiated by whether they rely on synchrony assumptions, a TTP, or a combination thereof (hybrid).

4. It is a property of the PoW mining process that a certain percentage of blocks exceeds or fall short of the required difficulty.

- **Synchrony Assumption (Direct Observation).** The simplest approach to cross-chain verification is to require all participants of a CCC protocol 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 [19], [90], but also in parent-child settings where one chain by design verifies or validates the other [36], [81], [120]. This relies on a synchrony assumption that requires the CCC participants to observe and act upon the state evolution of chains within a certain time, in order to complete the CCC.
- **Trusted Third Party (Coordinators).** Similar to the commit and abort phases of CCC, the verification of the state of interlinked chains can be handled by a trusted third party (also referred to as *validator* [163]). Thereby, we differentiate between the following techniques:
  - **External Validators.** A simple approach is to outsource the verification step of CCC to a (trusted) third party, external to the verifying ledger (in our case Y), as in [17], [160]. The TTP can thereby 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 [68], [112], [119] – leveraging the assumption that misbehavior of consensus participants indicates a failure of the chain itself. The verification process can be further encoded in smart contracts, as in the case of BTCRelay [4], which verifies Bitcoin block headers. Thereby, smart contracts have recently been used to verify succinct proofs of knowledge [72], [171], which in turn can (theoretically) enable proactive verification of State Validity in CCC protocols.
  - **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 [24] or accuse coordinators of misbehavior requiring them to prove correct operation [86], [101], [158].
- **Hybrid (Watchtowers).** Synchrony and TTP assumptions can be combined, such that a CCC protocol works if at least either a synchrony or TTP assumption holds. For example, in case one of the parties in the CCC goes offline, TTPs in the form of *watchtowers*, that take over the role of the offline party can be relied upon. See Section 4.1.3.

6. Evaluation of CCC Protocols

In this section, we evaluate existing CCC protocols based on the introduced classifications. We present our results in Table 1. We note that we performed this evaluation to the best of our knowledge, but are not able to cover all existing projects released in the blockchain community. As such, our focus lies on published academic papers and on community-driven projects deployed to practice.

Methodology. We group protocols by their design rationale, i.e., bidirectional exchanges and one-way transfers. For the latter we split protocols according to the underlying blockchain security assumptions: homogeneous and heterogeneous. The main focus of the evaluation lies on how each protocol handles the impossibility result from Section 3 during each phase of the CCC process: Commit on X, verify and commit on Y, and, if implemented, abort on X in case of failure. We further evaluate the restrictions of becoming a trusted third party / intermediary in each protocol, i.e., whether the TTP selection is dynamic or static (pre-defined). We also consider collateralization of TTPs in protocols where participants are reimbursed in case of failure, to minimize financial damages faced.

6.1. Bidirectional Exchange Protocols

Exchange protocols are generally agnostic to the security assumptions made for the interlinked blockchains, as the default application is the exchange of assets between distinct ledgers.

We observe multiple techniques, with different underlying trust models, the simplest and currently most used being custodial (centralized) exchanges. Recent works, such as Tesseract [41], attempt to reduce the risk of theft by the custodian by leveraging trusted hardware, e.g. Intel SGX [94]. The long standing alternative to entrusting a custodian with to-be-exchanged funds are atomic swaps, first introduced in 2012 using hashed-timelock contracts for commit and abort phases [18], [19], and recently formalized in [90]. Hashlocks can thereby be replaced with signature locks [124], improving privacy and cross-platform support. As of today, the adoption of HTLC atomic swaps is scarce, which can be explained by the strict online requirements for participants: the initiator of the swap can steal funds of the receiving party, if the later does not claim the initiator's committed assets before the abort timelock expires (after the secret to the hashlock was released). Notarized atomic swaps [160] remove the online requirement for users, by entrusting a set of coordinators (*notaries*) with the verification (and timely reaction) to the release of the HTLC secret - however, introducing the risk of coordinators colluding with exchange counterparties to commit theft.

An alternative to interactive swaps via HTLCs or signature locks are SPV atomic swaps [18], [19], [90], [160]. Hereby, the initiator of the swap locks assets y in a smart contract on Y, which will release the later to anyone who can prove correct payment of x on X to the initiator (Verification of State Evolution and/or Validity). However, in accordance with the impossibility result, the counterparty can fail to provide the proof of payment on X to the smart contract on time - again enabling theft through a malicious abort by the initiator.

Recently, hybrid versions of HTLC and signature lock atomic swaps have been introduced, most notably Arwen [89] and A2L [157], where users enter assets into a multisignature escrow with an exchange coordinator. This relieves the user of strict online requirements while reducing the risk of theft by the TTP to a (long) lockup of assets in the worst case. However, this requires a more complex setup process (similar to payment channels [141]) and introduces higher costs in the form of additional fees to prevent malicious lockup of coordinator funds (griefing).

TABLE 1: Evaluation of Cross-Chain Communication protocols. Notation for non-binary TTP values: ○ relies on participants being available and synchrony, ● relies on TTP, ◑ hybrid.

		Protocol	CCC Protocol Execution: Commit on X			Verify / Commit on Y			Abort on X		
			TTP	Dynamic TTP?	Type	TTP	Collateral?	Type	TTP	Dynamic TTP?	Type
Exchange (Agnostic)		**Custodial Exchange** [41], [160]	●	✗	External Custodian (single, pre-defined)	●	✗	External Validator	●	✗	TTP (same)
		HTLC Atomic Swaps [18], [19], [90], [160]	○	-	Hash Lock	○	✗	Direct Observation	○	-	Timelock
		Notarized HTLC Atomic Swaps [160]	○	-	Hash Lock	●	✗	External Validator	◑	-	Timelock + TTP
		SPV Atomic Swaps [8], [57], [91], [108], [169]	●	✓	Smart Contract	●	✗	Smart Contract	-	-	-
		Collateralized SPV Atomic Swaps [108], [169]	●	✓	Smart Contract	●	✓	Smart Contract	-	-	-
		ECDSA Atomic Swaps [124]	○	-	Signature Lock	○	✗	Direct Observation	○	-	Timelock
		A2L [157]	◑	✗	Multisig Escrow + Signature Lock	○	✗	Direct Observation	◑	✗	Timelock + TTP
		Arwen [89]	◑	✗	Multisig Escrow + Hash Lock	○	✗	Direct Observation	◑	✗	Timelock + TTP
Transfer	Heterogeneous	**XCLAIM** [169]	●	✓	External Custodian (single, unrestricted)	●	✓	Smart Contract	-	-	-
		Multisig XCLAIM [170]	●	✓	Multisig Escrow (single, unrestricted)	●	✓	Smart Contract	-	-	-
		Dogethereum [159]	●	✓	External Custodian (single, unrestricted)	●	✓	Smart Contract	-	-	-
		PoS Sidechains [81]	●	✗	External Custodian (Consensus of Y)	●	✗	Smart Contract	-	-	-
		tBTC [16]	●	✗	External Custodian	●	✓	Smart Contract	-	-	-
		wBTC [17]	●	✗	External Custodian (single, pre-defined)	●	✗	Direct Observation	-	-	-
	Homogeneous	**ATOMIX** [112]	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	●	✗	TTP (same)
		SBAC [23], [151]	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	●	✗	TTP (same)
		Rapidchain [168]	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	-	-	-
		Fabric Channels [26]	●	✗	Consensus Custodian	●	✗	Consensus Committee	●	✗	TTP (same)
		Federated Sidechains/Pegs [36], [68]	●	✗	External Custodian (Consensus of Y)	●	✗	Consensus Committee	-	-	-
		PoS Sidechains [81]	●	✗	External Custodian (Consensus of Y)	●	✗	Consensus Committee	-	-	-
		Rootstock [119], [120]	●	✗	External Custodian (Consensus of Y)†	●	✗	Consensus Committee	-	-	-
		Proof-of-Burn [102], [153]	●	✓	Smart Contract / Burn address	●	✗	Smart Contract / Consensus Committee	-	-	-
		Merged Mining/Staking [81], [100]	●	✗	Consensus Custodian	●	✗	Consensus Committee	-	-	-
		Randomness Beacon [51]	●	✓	Smart Contract	●	✗	Smart Contract / Consensus Committee	-	-	-

† The Rootstock sidechain plans to rely on Bitcoin's (chain X) consensus participants (only those merge-mining Rootstock) to act as (consensus) custodians. As of this writing, however, the consensus committee of Rootstock (chain Y) is used as external custodian.

6.2. One-Way Transfers

Moving on to one-way asset transfers, a first observation is that this protocol family generally relies on the existence of a TTP.

6.2.1. Heterogeneous Transfers. Heterogeneous one-way transfer protocols, in most cases, focus on transferring assets to other ledgers via *cryptocurrency-backed assets* [26], [81], [120], [169]. An asset x is committed/locked on L_x, while a representation of this asset $y(x)$ is unlocked on L_y. This asset can then be used just like any other native asset y on L_y: transferred, exchanged, or e.g. used with smart contracts. The latter can be particularly useful if L_x itself only has limited scripting capabilities (such as Bitcoin). After use, $y(x)$ can be redeemed for the corresponding amount of x. Compared to exchange protocols, where each swap requires transaction broadcast on all interlinked chains[5], cryptocurrency-backed assets require synchronization only twice: once to issue and a second time to redeem the transferred assets.

The trust assumptions in this protocol family range from a single centralized custodian [17], to a dynamic network of collateralized intermediaries as in the case of XCLAIM [169]. In the latter approach, any participant can lock collateral in a smart contract on the issuing chain Y and act as custodial coordinator (or *vault*) for locking of x on X (commit phase). If the coordinator attempts to defraud users (fail to prove correct behavior), the smart contract on Y will automatically slash collateral and reimburse victims. Should (escrow) smart contracts be available on both interlinked chains, custodial coordinators serve only as fallback or performance improvement. A similar collateralization approach is followed by tBTC [16], yet with the restriction that the set of custodial coordinators is pre-defined (static). The risk of theft by coordinators can thereby be reduced by using multisignature escrows rather than custodial coordinators - at the cost of introducing complexity in terms of broadcast transactions and data published to the underlying blockchains [170].

A drawback of cryptocurrency-backed assets between heterogeneous chains is the necessity to maintain a stable exchange rate between the backing assets x and the assets y of the issuing chain, which are used as collateral. As such, a sudden significant fluctuation of the exchange rate can result in (i) financial damages to users holding $y(x)$, and in turn (ii) network congestion on both X and Y as users race to recover assets x. A further disadvantage of existing heterogeneous one-way transfer protocols is the lack of a dedicated abort phase on chain X: if Y fails to issue backed assets $y(x)$, locked assets x can remain frozen indefinitely at the TTP's discretion.

6.2.2. Homogeneous Transfers. Similar to heterogeneous transfers, this protocol group focuses on moving assets between different chains. The main difference hereby is that interlinked chains either maintain identical security assumptions or are dependent on one-another, e.g. exhibit a parent-child relation. We differentiate between communication (i) among shards in sharded blockchains and (ii) from a parent to child blockchains.

With the goal of *sharding* being to improve transaction throughput in blockchains, efficient communication among individual shards is a necessity. As such, we

5. Note: while payment channels allow to keep transactions off-chain, correct synchronization of e.g. time across ledgers introduces risk to race conditions (see Section 7.2).

observe cross-shard communication protocols [23], [26], [112], [151], [168] to rely on consensus participants of (individual) shards to execute both commit, verify and abort phases of CCC. Thereby, communication in sharded blockchains follows the assumption that consensus committees of all shards can be trusted, as otherwise the system is considered to fail ("correct by construction" assumption [112]). The main difference observed in this protocol group is the execution of an explicit abort phase in e.g. ATOMIX [112], as opposed to the assumption that proofs of correct commit/lock execution will be timely delivered between / accepted by consensus committees of different shards, as e.g. in SBAC [23], [151].

Sidechains, as first introduced by Back et al. in 2012 [36] aim to extend the functionality of existing blockchains (parent) by allowing to move assets to so-called child blockchains, which run their own consensus but to some extent are dependent on the parent chain. While the design of CCC protocol for sidechains are very similar to those of sharded systems, a core difference is that the homogeneous security assumptions only apply when transferring from parent to child, but not vice-versa. As such, e.g. in the case of Federated Sidechains/Pegs [36], [68], participants of the parent chain X cannot assume correct operation of the sidechain Y and hence consider the consensus committee of Y as *external*. Rootstock seeks to reduce this risk by using trusted hardware for the coordinators on parent chain X (in this case, Bitcoin) [119], [120].

Finally, mechanisms for bootstrapping new blockchains, such as merged mining [3], [100] and Proof-of-Burn [102], [153], as opposed to CCC protocols discussed so far, aim to transfer information other than digital assets. In the case of merged mining, this is a proof that some proof-of-work was performed on the parent chain, in Proof-of-Burn it is a proof that some assets were destroyed. A further difference hereby is the absence of a mechanism to return the transferred information back to the parent chain: these protocols are deployed as *velvet forks* [172] and hence the parent chain generally remains oblivious to the existence of the child chain(s).

6.3. General Observations

We proceed to briefly overview general observations with respect to existing CCC protocols.

Tendency to TTPs. The majority of CCC protocols resort to TTPs, rather than relying on participants being online and networks communication being synchronous (although synchronous models are often assumed nevertheless). Further, with the exceptions of XCLAIM [169] and Dogethereum [159], such CCC protocols utilize a predefined, static committee of coordinators.

Static Committees. If one of the interlinked chains Y implements a BFT agreement protocol and hence has a static consensus committee, this committee is typically used as TTP in all phases of CCC protocols. Arguably, this makes sense, as participants of the other blockchain X must anyway trust in the secure operation of Y (honest majority of the consensus committee).

Collateralization Trend. In recent works [16], [108], [159], [169], there is a trend towards collateralizing coordinators, with the aim of preventing financial damages to users and incentivizing correct behavior of TTPs - this way potentially achieving *economically trustless* CCC protocols (cf. Section 3.5). Here, the exchange rate is crucial to ensure that collateral has sufficient value to punish misbehavior, and stabilization measures are necessary to mitigate both short and long term fluctuations.

Absence of Hybrid Verification. Surprisingly, despite numerous recent works on constructing and optimizing watchtowers for payment channel networks [32], [34], [103], [125], these techniques have not yet been applied to CCC protocols – despite the similarity between payment channels and some CCC approaches (e.g. HTLC atomic swaps).

Interoperability Blockchains. Recently, there has been an influx of so called *interoperability blockchains* – specialized sharded distributed ledgers which aim to serve as communication platform between other blockchains [20], [93], [116], [152], [161], [163]. Thereby, individual shards, which are coordinated via a parent chain running a Byzantine fault tolerant agreement protocol, connect to and import assets from existing blockchains. Thereby, these projects have in common that they rely on existing techniques such as cryptocurrency-backed assets [81], [159], [169] to bridge the gap to existing systems (and are hence not included in the evaluation above). A formal treatment of this design, also considering distributed computations, is presented in [121].

7. Implications for Blockchain, Threat, Network, and Privacy Models

In this section, we overview implications of cross-chain communication on distributed ledgers and necessary considerations when designing CCC protocols.

7.1. Threat Model and Attacks

Security and Adversary Model. Both X and Y can have a well defined security model on their own. However, these security models are not necessarily the same and even further, it might not be trivial to compare the guarantees they provide. For instance, X may rely on PoW and thus assume that adversarial hash computation is bound by $\alpha \leq 33\%$ [74], [82], [145]. On the other hand, Y may use PoS 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 [75], [80] may be lower than that of accumulating computational power, or vice-versa [49]. Since permissionless distributed ledgers (such as PoW or PoS) are not Sybil resistant [69], i.e., provide weak identities at best, quantifying adversary strength is nearly impossible, even within a single ledger [31]. However, this task becomes even more difficult in the cross-chain setting: not only can consensus participants (i) "hop" between different chains [117], [130], destabilizing involved systems, but also (ii) be susceptible to bribing attacks, which can be executed cross-chain, making detection unlikely [99], [127].

Consensus Finality Guarantees. Interlinked chains X and Y may assume different finality guarantees in their ledgers. Consider the following: X accepts a transaction

as valid when confirmed by k subsequent blocks, e.g. as in PoW blockchains [78]; instead, Y deems transactions valid as soon as they are written to the ledger ($k = 1$, e.g. [21]). A CCC protocol triggers a state transition on Y conditioned on a transaction included in X, however, later an (accidental) fork occurs on X (perhaps deeper than k). While the state of X will be reverted, this may not be possible in Y according to consensus rules - although the Atomicity property of CCC would require such measures.

Replay Attacks. 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. Protections involve the use of sequence numbers, or chains keeping track of previously processed proofs [58], [126], [151]. Special consideration may be needed in case of permanent blockchain forks, as this may require updating the way verification is performed [126], [169].

Increasing Verification Cost (Griefing). An adversary can increase the cost of the verification of a transaction across chains. For instance, a spam attack makes the ledger grow in size, increasing both verification time and cost. This in turn may impair cross-chain state verification, especially in heterogeneous settings, where verification of external consensus is typically an expensive operation [169]. In sharded systems, an adversary can try to create transactions, the validation of which requires information from (almost) all shards, significantly increasing cost and defeating the purpose of sharding itself.

Composability Attacks. We recall, in blockchains with *stablizing* [28], [162] consensus, a security parameter k is used to denote the number of blocks or *confirmation* [5] a transaction should have, before being accepted as secure [78], [135], [138], i.e., with the probability of a reversion being negligible. The same applies to state agreement and state evolution proofs. In addition, the value linked a verified transaction must be considered: the higher the potential gain by an adversary, the higher the risk of an attack, and the more confirmations should be required [150]. However, following recent works [173], we argue *at least* the entire *composition* of a block must be considered, as this is the total value an adversary can gain from executing a successful attack. Recall that for a transaction to be reversed or modified, the entire block must be altered.

7.2. Network model

Synchrony Across Chains. The absence of a global clock across chains requires to either agree and trust a third party as external clock, or rely on chain-dependent time definition, such as the block generation rate [78], hindering a seamless synchronization across chains [78], [169]. Several factors, such as the instance of the consensus algorithm, computation and communication capabilities of consensus participants or peer-to-peer network delay must be considered for a correct operation of cross-chain communication protocols, especially if timelocks are used.

Data Availability. Protocols leveraging verification of state agreement or validity across chains typically rely on timely arrival of proofs and accompanying data (block headers, transactions, ...). Furthermore, existing sub-linear state verification techniques relying on probabilistic sampling require additional data to be included in the verified blockchain [105], [122]. If an adversary can exclude this data from the chain, these protocols not only become less efficient but may potentially exhibit vulnerabilities [122]. This is a particular problem in heterogeneous settings, if data availability is not enforced by consensus, e.g. if protocols are deployed via velvet forks [172]. One possible solution is to include data availability proofs [23], at the cost of increasing complexity of the process.

7.3. Blockchain Model

Cryptographic Primitives. Interconnected chains X and Y may leverage different cryptographic schemes, or different instances of the same scheme. Thereby, cross-chain communication often requires compatible cryptographic primitives: a CCC protocol between a system X using ECDSA [95] as its digital signature scheme and a system Y using Schnorr [147] is only possible if both schemes are instantiated over the same elliptic curve [124]. Similarly, HTLC-based protocols 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* [97].

Language Expressiveness. The functionality of CCC protocols is typically restricted by the computational operations supported by the involved chains. These can reach from near Turing complete environments [56], over restricted operation sets [135], [148], to the (near) absence of scripting functionality altogether [9], [52]. CCC protocols must hence (i) consider the operations supported by both X and Y and leverage the intersecting functionality [124]; or (ii) move assets from chain with limited functionality to those with high(er) language expressiveness [68], [159], [169].

7.4. Privacy and Linkability

Privacy is crucial in any financial interaction and thus in cross-chain communication as well. Ideally it should not be possible for an observer to determine what two events have been synchronized across chains (e.g., what two assets have been exchanged and by whom). Among other advantages, this improves the *fungibility* of payments. However, there exist several privacy attack vectors in cross-chain communication: (i) recent works [83], [123] show attacks leveraging the locking mechanism and some countermeasures have been proposed [123], [124], [144]; (ii) heuristics to explore blocks from different cryptocurrencies [100] as well as forks [154] to cluster miner and user accounts [92]; (iii) CCC protocols leveraging coordinators, similar to and payment hubs [84], [98], also lead to privacy leakages that enable further account clustering [166]. Recent works [83], [88], [157] propose measures that allow to preserve the anonymity of participants, if added to CCC protocols.

8. Related Work

We believe that our study represents the most comprehensive systematic investigation of cross-chain communication to date. Yet, we list here other works and efforts by the blockchain community, illuminating different subsets of this space and supporting our study.

The first work discussing cross-chain communication, excluding forum discussions, is a technical report by Back et al. [36]. 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 more recent 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 [57]. Siris et al. provide an iterative overview of protocols for atomic cross-chain swaps and "sidechains" [149], focusing mostly on community driven efforts, rather than academic publications. Similarly, Johnson et al. discuss open source interoperability projects related to Ethereum [96], while Robinson evaluates Ethereum as a coordination platform for communication among other blockchains [143]. Bennik et al. [40] and similarly Miraz et al. [133] 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 [33].

9. Concluding Remarks

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 formalize 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. The classifications and comparative evaluations introduced in this work, taking into account both academic research and the vast number of online resources, allow to better understand the similarities and differences between existing cross-chain communication approaches - and possibly contribute to clearer communication between academia and industry in this field. Finally, by discussing implications and open challenges related to blockchain, network, and threat models, as well as privacy and linkability, we offer a comprehensive guide for designing protocols, bridging multiple distributed ledgers.

10. Acknowledgements

We would like express our gratitude to Georgia Avarikioti, Daniel Perez and Dominik Harz for helpful comments and feedback on earlier versions of this manuscript. We also thank Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, and Edgar Weippl for insightful discussions during the course of this research.

This research was funded by Bridge 1 858561 SESC, Bridge 1 864738 PR4DLT (all FFG), the Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), the competence center SBA-K1 funded by COMET, Chaincode Labs and the Austrian Science Fund (FWF) through the Meitner program. Mustafa Al-Bassam is funded by a scholarship from the Alan Turing Institute. Alexei Zamyatin is supported by the Binance Research Fellowship.

References

[1] Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv, accessed: 2018-05-16

[2] Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts, accessed: 2018-05-16

[3] Bitcoin wiki: Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification, accessed: 2018-05-03

[4] Btcrelay. https://github.com/ethereum/btcrelay, accessed 2019-08-15

[5] Confirmations. https://en.bitcoin.it/wiki/Confirmation, accessed: 2018-11-28

[6] Dogerelay. https://github.com/dogethereum/dogerelay, accessed 2019-08-15

[7] Eth-eos-relay. https://github.com/EveripediaNetwork/eth-eos-relay, accessed 2019-08-15

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

[9] Monero reference implementation. https://github.com/monero-project/monero, accessed: 2018-07-30

[10] Parity-Bridge. https://github.com/paritytech/parity-bridge, accessed 2019-08-15

[11] The parity light protocol - wiki. https://wiki.parity.io/The-Parity-Light-Protocol-(PIP), accessed: 2018-10-30

[12] Peace relay. https://github.com/loiluu/peacerelay, accessed 2019-08-15

[13] Poa bridge. https://github.com/poanetwork/poa-bridge, accessed: 2018-05-23

[14] Project alchemy. https://github.com/ConsenSys/Project-Alchemy, accessed 2019-08-15

[15] Project waterloo. https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum-1c230ac65524, accessed 2019-08-15

[16] tbtc: A decentralized redeemable btc-backed erc-20 token. http://docs.keep.network/tbtc/index.pdf, accessed: 2019-11-15

[17] Wrapped bitcoin. https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf, accessed: 2018-05-03

[18] Alt chains and atomic transfers. bitcointalk.org (2013), https://bitcointalk.org/index.php?topic=193281.msg2003765#msg2003765

[19] Atomic swap. Bitcoin Wiki (2013), https://en.bitcoin.it/wiki/Atomic_swap

[20] Wanchain whitepaper. https://www.wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf (2017)

[21] Abraham, I., Gueta, G., Malkhi, D.: Hot-stuff the linear, optimal-resilience, one-message bft devil. arXiv:1803.05069 (2018), https://arxiv.org/pdf/1803.05069.pdf

[22] Al-Bassam, M.: Lazyledger: A distributed data availability ledger with client-side smart contracts (2019), https://arxiv.org/pdf/1905.09274.pdf

[23] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., Danezis, G.: Chainspace: A sharded smart contracts platform. In: 2018 Network and Distributed System Security Symposium (NDSS) (2018)

[24] Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. CoRR **abs/1809.09044** (2018), http://arxiv.org/abs/1809.09044

[25] Alp, E.C., Kokoris-Kogias, E., Fragkouli, G., Ford, B.: Rethinking general-purpose decentralized computing. In: Proceedings of the Workshop on Hot Topics in Operating Systems. pp. 105–112. ACM (2019)

[26] Androulaki, E., Cachin, C., De Caro, A., Kokoris-Kogias, E.: Channels: Horizontal scaling and confidentiality on permissioned blockchains. In: European Symposium on Research in Computer Security. pp. 111–131. Springer (2018)

[27] Andrychowicz, M.: Multiparty computation protocols based on cryptocurrencies (2015), https://depotuw.ceon.pl/bitstream/handle/item/1327/dis.pdf, accessed: 2017-02-15

[28] Angluin, D., Fischer, M.J., Jiang, H.: Stabilizing consensus in mobile networks. In: Distributed Computing in Sensor Systems. pp. 37–50. Springer (2006), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.1040&rep=rep1&type=pdf

[29] Asokan, N.: Fairness in electronic commerce (1998)

[30] Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 591–606. Springer (1998)

[31] Avarikioti, G., Käppeli, L., Wang, Y., Wattenhofer, R.: Bitcoin security under temporary dishonest majority. In: 23rd Financial Cryptography and Data Security (FC) (2019), https://www.tik.ee.ethz.ch/file/ab83461dc5ca3b739c079a27f3757e94/bitcoin%20security%20under%20temporary%20dishonest%20majority.pdf

[32] Avarikioti, G., Kogias, E.K., Wattenhofer, R.: Brick: Asynchronous state channels. arXiv preprint arXiv:1905.11360 (2019)

[33] Avarikioti, G., Kokoris-Kogias, E., Wattenhofer, R.: Divide and scale: Formalization of distributed ledger sharding protocols. arXiv preprint arXiv:1910.10434 (2019)

[34] Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards secure and efficient payment channels. arXiv preprint arXiv:1811.12740 (2018)

[35] Babaoglu, O., Toueg, S.: Understanding non-blocking atomic commitment. Distributed systems pp. 147–168 (1993)

[36] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014), https://blockstream.com/sidechains.pdf

[37] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptology ePrint Archive **2018**, 46 (2018)

[38] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography Conference. pp. 31–60. Springer (2016)

[39] Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 274–285. Springer (1993)

[40] Bennink, P., Gijtenbeek, L.v., Deventer, O.v., Everts, M.: 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

[41] Bentov, I., Ji, Y., Zhang, F., Li, Y., Zhao, X., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. Cryptology ePrint Archive, Report 2017/1153 (2017), https://eprint.iacr.org/2017/1153.pdf, accessed:2017-12-04

[42] Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Advances in Cryptology–CRYPTO 2014. pp. 421–439. Springer (2014), http://eprint.iacr.org/2014/129.pdf

[43] Bentov, I., Pass, R., Shi, E.: Snow white: Provably secure proofs of stake (2016), https://eprint.iacr.org/2016/919.pdf, accessed: 2016-11-08

[44] Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems, vol. 370. Addison-wesley New York (1987)

[45] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: 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. pp. 326–349. ACM (2012)

[46] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (2018)

[47] Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188 (2018), https://eprint.iacr.org/2018/1188.pdf, https://eprint.iacr.org/2018/1188

[48] Boneh, D., Naor, M.: Timed commitments. In: Annual International Cryptology Conference. pp. 236–254. Springer (2000)

[49] Bonneau, J.: 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), http://fc16.ifca.ai/bitcoin/papers/Bon16b.pdf

[50] Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source (2015), https://eprint.iacr.org/2015/1015.pdf, accessed: 2015-10-25

[51] Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source. IACR Cryptology ePrint Archive **2015**, 1015 (2015)

[52] Bonneau, J., Miller, A.: Fawkescoin: Bitcoin without public-key crypto. In: Security Protocols XXII. pp. 350–358. Springer (2014), http://www.jbonneau.com/doc/BM14-SPW-fawkescoin.pdf

[53] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: IEEE Symposium on Security and Privacy (2015), http://www.ieee-security.org/TC/SP2015/papers-archived/6949a104.pdf

[54] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Efficient range proofs for confidential transactions (2017), http://web.stanford.edu/~buenz/pubs/bulletproofs.pdf, accessed:2017-11-10

[55] Bünz, B., Goldfeder, S., Bonneau, J.: Proofs-of-delay and randomness beacons in ethereum (2017)

[56] Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform (2014), https://github.com/ethereum/wiki/wiki/White-Paper, accessed: 2016-08-22

[57] Buterin, V.: Chain interoperability. Tech. report (2016), https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf, accessed: 2017-03-25

[58] Buterin, V.: Cross-shard contract yanking. https://ethresear.ch/t/cross-shard-contract-yanking/1450 (2018)

[59] Cachin, C.: Architecture of the hyperledger blockchain fabric (2016), https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf, accessed: 2016-08-10

[60] Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Annual International Cryptology Conference. pp. 93–111. Springer (2000)

[61] Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186 (1999), http://pmg.csail.mit.edu/papers/osdi99.pdf

[62] Catalano, D., Fiore, D.: Vector commitments and their applications. In: International Workshop on Public Key Cryptography. pp. 55–72. Springer (2013)

[63] Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. vol. 43, pp. 225–267. ACM (1996), https://ecommons.cornell.edu/bitstream/handle/1813/7192/95-1535.pdf?sequence=1

[64] Chepurnoy, A., Duong, T., Fan, L., Zhou, H.S.: Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake (2017), http://eprint.iacr.org/2017/232.pdf, accessed: 2017-03-22

[65] David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573 (2017), http://eprint.iacr.org/2017/573.pdf, accessed: 2017-06-29

[66] Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and mtgox. In: Computer Security-ESORICS 2014. pp. 313–326. Springer (2014), http://www.tik.ee.ethz.ch/file/7e4a7f3f29917847860372 85f4876f5c/malleability.pdf

[67] Dijkstra, E.W.: Solution of a problem in concurrent programming control. In: Pioneers and Their Contributions to Software Engineering, pp. 289–294. Springer (2001)

[68] Dilley, J., Poelstra, A., Wilkins, J., Piekarska, M., Gorlick, B., Friedenbach, M.: Strong federations: An interoperable blockchain solution to centralized third party risks. arXiv preprint arXiv:1612.05491 (2016)

[69] Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems. pp. 251–260. Springer (2002), http://www.cs.cornell.edu/people/egs/cs6460-spring10/sybil.pdf

[70] Duong, T., Fan, L., Zhou, H.S.: 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716 (2016), https://eprint.iacr.org/2016/716.pdf, accessed: 2017-02-06

[71] Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984. ACM (2018)

[72] Eberhardt, J., Tai, S.: Zokrates-scalable privacy-preserving off-chain computations

[73] Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: CCS (2019)

[74] Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security. pp. 436–454. Springer (2014), http://arxiv.org/pdf/1311.0243

[75] Fanti, G., Kogan, L., Oh, S., Ruan, K., Viswanath, P., Wang, G.: Compounding of wealth in proof-of-stake cryptocurrencies. arXiv preprint arXiv:1809.07468 (2018)

[76] Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. vol. 32, pp. 374–382. ACM (1985), http://macs.citadel.edu/rudolphg/csci604/ImpossibilityofConsensus.pdf

[77] Ford, B., Gasser, L., Kogias, E.K., Jovanovic, P.: Cryptographically verifiable data structure having multi-hop forward and backwards links and associated systems and methods (Dec 13 2018), uS Patent App. 15/618,653

[78] Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty (2016), http://eprint.iacr.org/2016/1048.pdf, accessed: 2017-02-06

[79] Gärtner, F.C.: Specifications for fault tolerance: A comedy of failures (1998)

[80] Gaži, P., Kiayias, A., Russell, A.: Stake-bleeding attacks on proof-of-stake blockchains. Cryptology ePrint Archive, Report 2018/248 (2018), https://eprint.iacr.org/2018/248.pdf, accessed:2018-03-12

[81] Gazi, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. IEEE Security and Privacy. IEEE (2019)

[82] Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdo rf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC. pp. 3–16. ACM (2016)

[83] Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701 (2016), http://eprint.iacr.org/2016/701, accessed: 2017-08-07

[84] Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Off the chain transactions. Cryptology ePrint Archive, Report 2019/360 (2019), https://eprint.iacr.org/2019/360.pdf, https://eprint.iacr.org/2019/360

[85] Han, R., Lin, H., Yu, J.: On the optionality and fairness of atomic swaps. Cryptology ePrint Archive, Report 2019/896 (2019), https://eprint.iacr.org/2019/896

[86] Harz, D., Boman, M.: The scalability of trustless trust. arXiv:1801.09535 (2018), https://arxiv.org/pdf/1801.09535.pdf, accessed:2018-01-31

[87] Harz, D., Gudgeon, L., Gervais, A., Knottenbelt, W.J.: Balance: Dynamic adjustment of cryptocurrency deposits. Cryptology ePrint Archive, Report 2019/675 (2019), https://eprint.iacr.org/2019/675.pdf, https://eprint.iacr.org/2019/675

[88] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub (2016), https://eprint.iacr.org/2016/575.pdf, accessed: 2017-09-29

[89] Heilman, E., Lipmann, S., Goldberg, S.: The arwen trading protocols. Whitepaper, https://www.arwen.io/whitepaper.pdf

[90] Herlihy, M.: Atomic cross-chain swaps. arXiv:1801.09515 (2018), https://arxiv.org/pdf/1801.09515.pdf, accessed:2018-01-31

[91] Herlihy, M., Liskov, B., Shrira, L.: Cross-chain deals and adversarial commerce. arXiv preprint arXiv:1905.09743 (2019)

[92] Hinteregger, A., Haslhofer, B.: An empirical analysis of monero cross-chain traceability. arXiv preprint arXiv:1812.02808 (2018)

[93] Hosp, D., Hoenisch, T., Kittiwongsunthorn, P., et al.: Comit-cryptographically-secure off-chain multi-asset instant transaction network. arXiv preprint arXiv:1810.02174 (2018)

[94] Intel Corp.: Software Guard Extensions Programming Reference, Ref. 329298-002US. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf (2014), https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[95] Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**(1), 36–63 (2001)

[96] Johnson, S., Robinson, P., Brainard, J.: Sidechains and interoperability. arXiv preprint arXiv:1903.04077 (2019)

[97] Jones, J., abitmore: Optional htlc preimage length and hash160 addition. BSIP 64, blog post, https://github.com/bitshares/bsips/issues/163

[98] Jourenko, M., Kurazumi, K., Larangeira, M., Tanaka, K.: Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Report 2019/352 (2019), https://eprint.iacr.org/2019/352.pdf, https://eprint.iacr.org/2019/352

[99] Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gaži, P., Meiklejohn, S., Weippl, E.: Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. Cryptology ePrint Archive, Report 2019/775 (2019), https://eprint.iacr.org/2019/775.pdf, https://eprint.iacr.org/2019/775

[100] Judmayer, A., Zamyatin, A., Stifter, N., Voyiatzis, A.G., Weippl, E.: Merged mining: Curse or cure? In: CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology (Sep 2017), https://eprint.iacr.org/2017/791.pdf

[101] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: Proceedings of the 27th USENIX Conference on Security Symposium. pp. 1353–1370. USENIX Association (2018)

[102] Karantias, K., Kiayias, A., Zindros, D.: Proof-of-burn. International Conference on Financial Cryptography and Data Security (2019)

[103] Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: A responsive lightweight watchtower (2019)

[104] Kiayias, A., Lamprou, N., Stouka, A.P.: Proofs of proofs of work with sublinear complexity. In: International Conference on Financial Cryptography and Data Security. pp. 61–78. Springer, Springer (2016)

[105] Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963 (2017), https://eprint.iacr.org/2017/963.pdf, accessed:2017-10-03

[106] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol (2016), https://pdfs.semanticscholar.org/a583/3270b14e251f0b16d86438d04652b1b8d7f3.pdf, accessed: 2018-08-19

[107] Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 705–734. Springer (2016), https://eprint.iacr.org/2015/574.pdf

[108] Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: International Conference on Financial Cryptography and Data Security. Springer (2018)

[109] Kogias, E.K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, Austin, TX (Aug 2016), http://arxiv.org/pdf/1602.06997.pdf

[110] Kokoris-Kogias, E.: Robust and scalable consensus for sharded distributed ledgers. Tech. rep., Cryptology ePrint Archive, Report 2019/676 (2019)

[111] Kokoris-Kogias, E., Alp, E.C., Siby, S.D., Gailly, N., Gasser, L., Jovanovic, P., Syta, E., Ford, B.: Calypso: Auditable sharing of private data over blockchains. Tech. rep., Cryptology ePrint Archive, Report 2018/209 (2018)

[112] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 583–598. IEEE (2018)

[113] Kokoris-Kogias, L., Gasser, L., Khoffi, I., Jovanovic, P., Gailly, N., Ford, B.: Managing identities using blockchains and CoSi. In: 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016) (2016)

[114] Kumaresan, R., Bentov, I.: Amortizing secure computation with penalties. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 418–429. ACM (2016)

[115] Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. Computer Networks **56**(1), 50–63 (2012)

[116] Kwon, J., Buchman, E.: Cosmos: A network of distributed ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md (2015)

[117] Kwon, Y., Kim, H., Shin, J., Kim, Y.: Bitcoin vs. bitcoin cash: Coexistence or downfall of bitcoin cash? arXiv:1902.11064 (2019), https://arxiv.org/pdf/1902.11064.pdf

[118] Lamport, L.: A simple approach to specifying concurrent systems. Communications of the ACM **32**(1), 32–45 (1989)

[119] Lerner, S.D.: Rootstock: Bitcoin powered smart contracts. https://docs.rsk.co/RSK_White_Paper-Overview.pdf (2015)

[120] Lerner, S.: Drivechains, sidechains and hybrid 2-way peg designs. Tech. rep., Tech. Rep. [Online] (2018), https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf

[121] Liu, Z., Xiang, Y., Shi, J., Gao, P., Wang, H., Xiao, X., Hu, Y.C.: Hyperservice: Interoperability and programmability across heterogeneous blockchains. arXiv preprint arXiv:1908.09343 (2019)

[122] Luu, L., Buenz, B., Zamani, M.: Flyclient super light client for cryptocurrencies https://stanford2017.scalingbitcoin.org/files/Day1/flyclientscalingbitcoin.pptx.pdf, accessed 2018-04-17

[123] Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: CCS. pp. 455–471 (2017)

[124] Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS (2019)

[125] McCorry, P., Bakshi, S., Bentov, I., Miller, A., Meiklejohn, S.: Pisa: Arbitration outsourcing for state channels. IACR Cryptology ePrint Archive **2018**, 582 (2018)

[126] McCorry, P., Heilman, E., Miller, A.: 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), http://homepages.cs.ncl.ac.uk/patrick.mc-corry/atomically-trading-roger.pdf

[127] McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), http://fc18.ifca.ai/bitcoin/papers/bitcoin18-final14.pdf

[128] Meckler, I., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf (2018)

[129] Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 369–378. Springer (1987)

[130] Meshkov, D., Chepurnoy, A., Jansen, M.: Revisiting difficulty control for blockchain systems. Cryptology ePrint Archive, Report 2017/731 (2017), http://eprint.iacr.org/2017/731.pdf, accessed: 2017-08-03

[131] Micali, S.: Algorand: The efficient and democratic ledger (2016), https://arxiv.org/pdf/1607.01341.pdf, accessed: 2017-02-09

[132] Miller, A.: The high-value-hash highway, bitcoin forum post (2012), https://bitcointalk.org/index.php?topic=98986.0

[133] Miraz, M., Donald, D.C.: 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)

[134] Moreno-Sanchez, P., Randomrun, Le, D.V., Noether, S., Goodell, B., Kate, A.: Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. Cryptology ePrint Archive, Report 2019/595 (2019), https://eprint.iacr.org/2019/595

[135] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (Dec 2008), https://bitcoin.org/bitcoin.pdf, accessed: 2015-07-01

[136] Nikitin, K., Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Gasser, L., Khoffi, I., Cappos, J., Ford, B.: CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds

[137] Pagnia, H., Gärtner, F.C.: On the impossibility of fair exchange without a trusted third party. Tech. rep., Technical Report TUD-BS-1999-02, Darmstadt University of Technology ... (1999)

[138] Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks (2016), http://eprint.iacr.org/2016/454.pdf, accessed: 2016-08-01

[139] Pass, R., Shi, E.: Hybrid consensus: Scalable permissionless consensus (Sep 2016), https://eprint.iacr.org/2016/917.pdf, accessed: 2016-10-17

[140] Poelstra, A.: Scriptless scripts. Presentation slides, https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf

[141] Poon, J., Dryja, T.: The bitcoin lightning network (2016), https://lightning.network/lightning-network-paper.pdf, accessed: 2016-07-07

[142] Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)

[143] Robinson, P.: The merits of using ethereum mainnet as a coordination blockchain for ethereum private sidechains (2019), https://arxiv.org/pdf/1906.04421.pdf

[144] Rubin, J., Naik, M., Subramanian, N.: Merkelized abstract syntax trees. http://www.mit.edu/~jlrubin/public/pdfs/858report.pdf (2014)

[145] Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in bitcoin (2015), http://arxiv.org/pdf/1507.06183.pdf, accessed: 2016-08-22

[146] Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Practical continuous distributed randomness. Cryptology ePrint Archive, Report 2018/319 (2018), https://eprint.iacr.org/2018/319.pdf

[147] Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)

[148] Sergey, I., Kumar, A., Hobor, A.: Scilla: a smart contract intermediate-level language. arXiv:1801.00687 (2018), https://arxiv.org/pdf/1801.00687.pdf, accessed:2018-01-08

[149] Siris, V.A., Dimopoulos, D., Fotiou, N., Voulgaris, S., Polyzos, G.C.: Interledger smart contracts for decentralized authorization to constrained things (2019), https://arxiv.org/pdf/1905.01671.pdf

[150] Sompolinsky, Y., Zohar, A.: Bitcoin's security model revisited (2016), http://arxiv.org/pdf/1605.09193.pdf, accessed: 2016-07-04

[151] Sonnino, A., Bano, S., Al-Bassam, M., Danezis, G.: Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. arXiv preprint arXiv:1901.11218 (2019)

[152] Spoke, M., Nuco Engineering Team: Aion: The third-generation blockchain network. https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf, accessed 2018-04-17

[153] Stewart, I.: Proof of burn (2012), https://en.bitcoin.it/wiki/Proof_of_burn, accessed: 2017-05-10

[154] Stifter, N., Schindler, P., Judmayer, A., Zamyatin, A., Kern, A., Weippl, E.: 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), https://fc19.ifca.ai/preproceedings/41-preproceedings.pdf

[155] Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460. Ieee (2017)

[156] Sztorc, P.: Blind merged mining. http://www.truthcoin.info/blog/blind-merged-mining/, accessed 2018-04-17

[157] Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^{2}l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. Cryptology ePrint Archive, Report 2019/589 (2019), https://eprint.iacr.org/2019/589

[158] Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains (March 2017), https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf, accessed:2017-10-06

[159] Teutsch, J., Straka, M., Boneh, D.: Retrofitting a two-way peg between blockchains. Tech. rep. (2018), https://people.cs.uchicago.edu/~teutsch/papers/dogethereum.pdf

[160] Thomas, S., Schwartz, E.: A protocol for interledger payments. URL https://interledger. org/interledger. pdf (2015)

[161] Verdian, G., Tasca, P., Paterson, C., Mondelli, G.: Quant overledger whitepaper. https://www.quant.network/wp-content/uploads/2018/09/Quant_Overledger_Whitepaper-Sep.pdf (2018)

[162] Vukolic, M.: Eventually returning to strong consistency (2016), https://pdfs.semanticscholar.org/a6a1/b70305b27c556aac779fb65429db9c2e1ef2.pdf, accessed: 2016-08-10

[163] Wood, G.: Polkadot: Vision for a heterogeneous multi-chain framework. White Paper (2015)

[164] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07) (2017), https://ethereum.github.io/yellowpaper/paper.pdf, accessed: 2018-01-03

[165] Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167. IEEE (1986)

[166] Yousaf, H., Kappos, G., Meiklejohn, S.: Tracing transactions across cryptocurrency ledgers. In: 28th {USENIX} Security Symposium ({USENIX} Security 19). pp. 837–850 (2019)

[167] Yu, M., Sahraei, S., Li, S., Avestimehr, S., Kannan, S., Viswanath, P.: Coded merkle tree: Solving data availability attacks in blockchains. arXiv preprint arXiv:1910.01247 (2019)

[168] Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: A fast blockchain protocol via full sharding. Cryptology ePrint Archive, Report 2018/460 (2018), https://eprint.iacr.org/2018/460.pdf

[169] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.: Xclaim: Trustless, interoperable, cryptocurrency-backed assets. IEEE Security and Privacy. IEEE (2019)

[170] Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.J.: Multisignatures for cryptocurrency-backed tokens (poster). CBT'18: International Workshop on Cryptocurrencies and Blockchain Technology (2018)

[171] Zamyatin, A., Perez-Hernandez, D., Harz, D.: Snark-relay: Towards bitcoin transaction inclusion proofs via zksnark (2019), https://github.com/dec3ntral/snark-relay

[172] Zamyatin, A., Stifter, N., Judmayer, A., Schindler, P., Weippl, E., Knottebelt, W.J.: (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), https://eprint.iacr.org/2018/087.pdf

[173] Zindros, D.: Summa proofs are not composable (2019), https://medium.com/@dionyziz/summa-proofs-are-not-composable-57b87825f428