SoK: Communication Across Distributed Ledgers

Alexei Zamyatin
Imperial College London and SBA Research

Mustafa Al-Bassam
University College London

Dionysis Zindros
University of Athens and IOHK

Eleftherios Kokoris-Kogias
Ecole polytechnique fédérale de Lausanne

Pedro Moreno-Sanchez
TU Wien

Aggelos Kiayias
University of Edinburgh and IOHK

William J. Knottenbelt
Imperial College London

Abstract

Enabling secure communication across distributed systems is usually studied under the assumption of trust between the different systems and an external adversary trying to compromise the messages. 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 (CCC) thereby plays a fundamental role in cryptocurrency exchanges, sharding, bootstrapping of new and feature-extension of existing distributed ledgers. Unfortunately, existing proposals are designed ad-hoc for specific use-cases, making it hard to gain confidence on their correctness and composability.

We provide the first systematic exposition of protocols for CCC. First, we formalize the underlying research problem and show that CCC is *impossible without a trusted third party*, contrary to common beliefs in the blockchain community. We then develop a framework to evaluate existing and to design new cross-chain protocols. The framework is based on the use case, the trust model, and the 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. It aims to facilitate clearer communication between academia and industry in the field.

1 Introduction

Why Cross-Chain Communication is Worthy of Research. Since the introduction of Bitcoin [140] as the first decentralized ledger currency in 2008, the topic of blockchains (or distributed ledgers) has evolved into a well-studied field both in 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, 93–95], but is a critical component of scalabilty solutions (synchronization in sharded systems [22, 23, 26, 116, 175]), feature extensions (sidechains [37, 85, 112, 124] and cryptocurrency-backed assets [165, 177]), as well as bootstrapping of new and migration between existing systems [3, 51, 104, 158]. In addition, numerous competing projects aiming to create a single platform for cross-chain communication have recently emerged [20, 97, 121, 157, 166, 167, 170]. 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) [45], 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) [36]. 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. We formalize the underlying problem of Correct Cross-Chain Communication (CCC) (Section 2) and show it is *impossible without a trusted third party* (TTP) by relating CCC to the Fair Exchange problem (Section 3). With the impossibility result in mind, we propose a framework to design new and evaluate existing CCC protocols, focusing on the inherent trust assumptions thereof (Sections 4-5). Next, we overview CCC protocols proposed in literature and introduce a classification based on our framework (Section 6). Finally, we discuss implications for the blockchain, network, and threat models, and examine transcendence for privacy (Section 7).

Contact email: azamyatin@imperial.ac.uk

2 The CCC Problem

In this section, provide a formal definition of the Correct Cross-Chain Communication (CCC) problem and introduce the model for blockchains involved in CCC.

2.1 Distributed Ledger Model

We use the terms *blockchain* and *distributed ledger* as synonyms and introduce some notation, based on [85] with minor alterations.

Ledgers and State Evolution. When speaking of CCC, 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. 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). We denote a ledger as L (L_x and L_y respectively) and define its *state* as the dynamically evolving sequence of included transactions $\langle \mathrm{TX}_1, ..., \mathrm{TX}_n \rangle$. We assume that the evolution of the ledger state progresses in discrete *rounds* indexed by natural numbers $r \in \mathbb{N}$. At each round r, a new set of transactions (included in a newly generated block) is written to the ledger L. We use $\mathsf{L}^P[r]$ to denote the state of ledger L at round r, i.e., after applying all transactions *written* to the ledger since round $r-1$, according to the view of some party P (in a properly working blockchain system, all parties will eventually have agreeing ledgers). A transaction can be written to L only if it is consistent with the system's consensus rules, given the current ledger state $\mathsf{L}^P[r]$. This consistency is left for the particular system to define, and we describe it as a free predicate $\mathsf{valid}(\cdot)$ and we write $\mathsf{valid}(\mathrm{TX}, \mathsf{L}_x^P[r])$ to denote that TX is valid under the consensus rules of L_x at round r according to the view of party P. To denote that a transaction TX has been *included* in / successfully written to a ledger L as position r we write $\mathrm{TX} \in \mathsf{L}^P[r]$. While the ordering of transactions in a block is crucial for their validity, for simplicity, we omit the position of transactions in blocks and assume correct ordering implicitly.

Notion of Time. The state evolution of two distinct ledgers L_x and L_y may progress at different *time* intervals: In the time that L_x progresses *one* round, L_y may, for example, progress *forty* rounds (e.g., as in the case of Bitcoin [140] and Ethereum [56]). To correctly capture the ordering of transactions across L_x and L_y, we define a clock function τ which maps a given round on any ledger to the time on a global, synchronized clock $\tau : r \to t$. We assume that the two chains are nevertheless synchronized and that there is no clock drift between them. We use this conversion implicitly in the rest of this paper. For conciseness, we will use the notation $\mathsf{L}^P[t]$ to mean the ledger state in the view of party P at the round $r = \tau^{-1}(t)$ which corresponds to time t, namely $\mathsf{L}^P[\tau^{-1}(t)]$.

Persistence and Liveness. Each participant P adopts and maintains a local ledger state $\mathsf{L}^P[t]$ at time t, i.e., her current view of the ledger. The views of two distinct participants P and Q on the same ledger L may differ at time t (e.g., due to network delay): $\mathsf{L}^P[t] \neq \mathsf{L}^Q[t]$. 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 [81]:

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

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

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

Transaction Model. A transaction TX, when included, alters the state of a ledger L by defining operations to be executed and agreed upon by consensus participants $P_1, ..., P_n$. The expressiveness of operations is thereby left for the particular system to define, and can range from simple payments to execution of complex programs [171]. For generality, we do not differentiate between specifics transactions models (e.g. UTXO [140] or account-based models [171]).

2.2 CCC System Model

Consider two independent distributed systems X and Y with underlying ledgers L_x and L_y, as defined in Section 2.1. We assume a *closed* system model as in [123] with a process P running on X and a process Q running on Y. A process can influence the state evolution of the underlying system by (i) writing a transaction TX to the underlying ledger L (commit), or (ii) by stopping to interact with the system (abort). We assume that P possesses transaction TX_P, which can be written to L_x, and Q possesses TX_Q, which can be written to L_y. A function *desc* maps a transaction to some "description" which can be compared to an expected description value, e.g., specifying the transaction value and recipient (the description differs from the transaction itself in that it may not, for example, contain any signature). P possesses a description d_Q which characterizes the transaction TX_Q, while Q possesses d_P which characterizes TX_P. Informally, P wants TX_Q to be written to L_y and Q wants TX_P to be written to L_x. Thereby, $d_P = \mathit{desc}(\mathrm{TX}_P)$ implies TX_P is valid in X (at time of CCC execution), as it cannot be written to L_x otherwise (analogous for d_Q).

For the network, we assume no bounds on message delay or deviations between local clocks, unless the individual blockchain protocols require this. We treat failure to communicate as adversarial behavior. We note that, in the anonymous blockchain setting, more synchrony requirements are imposed than in the byzantine setting. Our construction does not impose any *additional* synchrony requirements than the individual ledger protocols. Hence, if P or Q become malicious, we indicate this using boolean "error variables" [83] m_P and m_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.

2.3 Correct Cross-Chain Communication

The goal of cross-chain communication can be described as the synchronization of processes P and Q such that *Q writes TX_Q to L_y if and only if P has written TX_P to L_x*. Thereby, it must hold that $desc(\mathrm{TX}_P) = d_Q \wedge desc(\mathrm{TX}_Q) = d_P$. The intuition is that TX_P and 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.

To this end, P must convince Q that it created a transaction 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 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 TX_P and TX_Q match the expected descriptions (and are valid), then TX_P will be included in L_x and TX_Q will be included in L_y. If either of the transactions are not as expected, then both parties abort.*

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

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

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

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

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

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

2.4 Generic CCC Protocol

We now describe a generic CCC protocol, which handles the synchronization of transactions, which can represent the transfer of good, assets or objects, between two blockchain-based distributed systems X and Y with ledgers L_x and L_y which have Persistence and Liveness. A visual representation is provided in Figure 1. A CCC protocol can be compartmentalized into four phases.

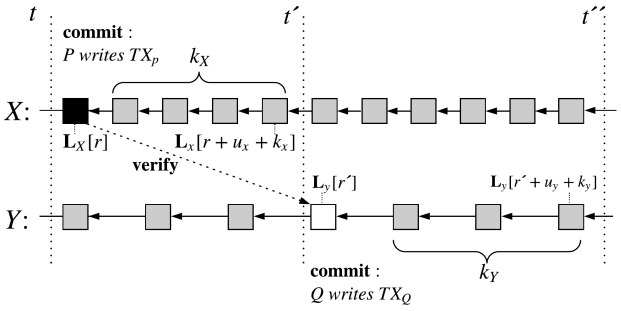

Figure 1: CCC **between X and Y. Process Q writes TX_Q only if P has written TX_P. We set exemplary persistence delays for X and Y as $k_X = 4$ and $k_Y = 3$, and liveness delays as $u_x = u_y = 0$. For readability, we do not visualize the abort phase.**

1) Setup. The setup phase is required to establish the parameters of CCC. Specifically, as CCC protocol is parameterized by the involved distributed systems X and Y and the corresponding ledgers L_x and L_y, the involved parties P and Q, the transactions TX_P and TX_Q as well as their descriptions d_P and d_Q. The latter descriptions ensure the validity TX_P and TX_Q and determine the application-level specification of a CCC protocol. For example, in the case of an exchange of assets or digital goods, d_P and d_Q define the asset types, transferred value, time constraints and any additional conditions agreed by parties P and Q. Typically, the setup process occurs out-of-band between the involved parties and we hence shift our focus on the actual execution of CCC in the rest of this paper.
2) (Pre-) Commit on X. Upon a successful setup, a publicly verifiable commitment to execute the CCC protocol is published on X: P writes[1] transaction TX_P to its local ledger L_X^P at time t in round r. Due to Persistence and Liveness of L_x, all honest parties of X will report TX_P as stable ($\mathrm{TX}_P \in \mathsf{L}_x$) in round $r + u_x + k_x$, where u_x is the liveness delay and k_x is the "depth" parameter of X.
3) Verify. Next, the correctness of the commitment on X by P is verified by Q. Specifically, Q checks (or receives proof from P) that (i) $d_P = desc(\mathrm{TX}_P)$ and (ii) $\mathrm{TX}_P \in \mathsf{L}_x$ hold. From Persistence and Liveness of X we know the latter check will succeed at time t' which corresponds to round $r + u_x + k_x$ on X, if P executed correctly.
4a) Commit on Y. Upon successful verification, a publicly verifiable commitment is published on Y: Q writes transaction TX_Q to its local ledger L_Y^Q at time t' in round r' on Y. Due to Persistence and Liveness of L_y, all honest parties of Y will report TX_Q as stable ($\mathrm{TX}_Q \in \mathsf{L}_y$) in round $r' + u_y + k_y$, where u_y is the liveness delay and k_y is the "depth" parameter of Y.
4b) Abort. If the verification fails and / or Q does not execute the commitment on Y, a CCC protocol can exhibit an abort step on X. This is achieved by "reverting" the state changes TX_P made to L_x. Since blockchains are append only data structures, reverting requires broadcasting an additional transaction $\mathrm{TX}_{P'}$ which resets X to the state before the commitment of TX_P.

[1] In off-chain protocols [88], the commitment can be done by exchanging pre-signed transactions or channel states, which will be written to the ledger at a later point.

It is worth noting that some CCC protocols, specifically such that facilitate *exchange* of assets, follow a two-phase commit design. In this case, steps 2 and 4a are executed in parallel, followed by the verification and (optional) abort steps on *both* X and Y.

A further observation is that a CCC protocol necessarily requires a conditional state transition to occur on Y, given a state transition on X. As such, we do not consider (oracle) protocols which merely relay data across distributed ledgers [4, 51, 55, 57, 163], as CCC protocols by themselves.

3 Impossibility of CCC

In this section we show that CCC is impossible without a trusted third party by reducing it to the Fair Exchange problem [29, 142].

Fair Exchange. 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 [31]. Fair exchange can be considered a sub-problem of fair secure computation [43], and is known to be impossible without a trusted third party [74, 75, 142, 172]. We provide a definition of Fair Exchange in accordance to existing literature in Appendix A.

3.1 Relating CCC to Fair Exchange.

We proceed to show that Correct Cross-Chain Communication is impossible without a trusted third party (TTP) under our deterministic system model by reducing it to Fair Exchange [29, 31, 142]. We recall, a fair exchange protocol must fulfill three properties: *Effectiveness*, *(Strong) Fairness* and *Timeliness* [29, 142].

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

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

We observe the definition of Timeliness in CCC is equivalent to the definition of Timeliness in fair exchange protocols, as defined in [142]. Effectiveness in fair exchange states that if P and Q behave correctly and do not want to abandon the exchange (i.e., $m_P = m_Q = \bot$), and items a_P and a_Q are as expected by Q and P, then at the end of the protocol, P will own the desired a_Q and Q will own the desired a_P [142]. It is easy to see Effectiveness in CCC achieves exactly this property: if P and Q behave correctly and $desc(\mathtt{tx}_P) = d_P$ and $desc(\mathtt{tx}_Q) = d_Q$, i.e., $\mathtt{tx}_P$ transfers a_P to Q and $\mathtt{tx}_Q$ transfers a_Q to P, then P will write $\mathtt{tx}_P$ to L_y at time t and Q will write $\mathtt{tx}_Q$ to L_x before time t'. From Persistence and Liveness of L_x and L_y we know both transactions will eventually be written to the local ledgers of P and Q, consequently all other honest participants of X will report $\mathtt{tx}_P \in \mathsf{L}_X$ and honest participants of Y will report $\mathtt{tx}_Q \in \mathsf{L}_Y$. From our model we know that honest participants constitute majorities in both X and Y. Hence, P will receive a_Q and Q will receive a_P.

Strong Fairness in fair exchange states that there is no outcome of the protocol, where P receives a_Q but Q does not receive a_P, or, vice-versa, Q receives a_P but P does not receive a_Q [142]. 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 construct a protocol for Fair Exchange using CCC in Appendix B. It is left to show that CCC is defined under the same model as Fair Exchange. The CCC model assumes the same asynchronous (explicitly) and deterministic (implicitly) system model (cf. Section 2.2) as [78, 142]. As we allow P and Q to stop participating in the CCC protocol at any time, we also have the same crash failure model as [78, 142]. Hence, we conclude:

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

Proof. Assume there exists an asynchronous protocol C which solves CCC. Then, due to Lemma 1 there exists a protocol 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 [78, 142]. In the open model, P and Q can be forced to make a decision by the system (or environment), i.e., transactions can be written on their behalf if they crash [116]. In the case of CCC, this means that distributed system Y, or more precisely, the consensus of Y, can write $\mathtt{tx}_Q$ to L_y on behalf of Q if P wrote $\mathtt{tx}_P$ to L_x. We observe that Y becomes the TTP in this scenario: both P and Q must agree that the consensus of Y will write $\mathtt{tx}_Q$ to L_y if $\mathtt{tx}_P \in \mathsf{L}_x$. In practice, 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, the execution of which is enforced by consensus of Y, that will write $\mathtt{tx}_Q$ to L_y if P includes $\mathtt{tx}_P$ in L_x, i.e., Q is allowed to crash.

However, it remains the question how system Y (the consensus participants of Y) becomes aware that $\mathtt{tx}_P \in \mathsf{L}_x$. In practice, a smart contract, can only perform actions based on some input. As such, before writing $\mathtt{tx}_Q$ the contract consensus of Y must observe and verify that $\mathtt{tx}_P$ was included in L_x. A protocol achieving CCC must hence make one of the following assumptions. Either, there exists a TTP that will ensure correct execution of CCC; or the protocol assumes P, or Q, or some other honest, online party (can in turn be Y) will always deliver a proof for $\mathtt{tx}_P \in \mathsf{L}_x$ to Y within a known upper bound, i.e., the protocol introduces some form of *synchrony* assumption. As argued in [142], we observe that introducing a TTP and relying on a synchrony assumption have are equivalent, 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 to the smart contract on Y, and that this message is delivered to the smart contract within a know upper bound, then we can be sure that CCC will occur correctly. Thereby, P intuitively represents its own trusted third party. However, if we cannot make assumptions on when the message will be delivered to the smart contract, as is the case in the asynchronous model, a trusted third party is necessary to determine the outcome of the CCC: the TTP observes $\mathrm{TX}_P \in \mathsf{L}_x$ and informs the smart contract or directly enforces the inclusion of 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 is a (trusted) third party to the protocol, indicates whether a participate has failed or misbehaved.

3.2 What is a Trusted Third Party?

Numerous recent works use *a single* distributed ledger such as Bitcoin and Ethereum to construct (optimistic) fair exchange protocols [27, 43, 71, 111, 115, 119]. 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 a_Q to P if Q received a_P.

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

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 enforces 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).

3.3 Incentives and Rational 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 [31, 43, 60, 120]. These workarounds suffer, among others, from a common drawback: they require a (some form of) 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. First suggested in the context of rational exchange protocols [161], the economic dimension of blockchains enabled a shift in this paradigm: Rather than forcing an honest user to invest time and money to achieve fairness, the malicious user is economically punished when breaching fairness and the victim is reimbursed. This has paved the way to design *economically trustless* CCC protocols that follow a game theoretic model under the assumption that actors behave rationally [177]. We remark that malicious/altruistic actors can nevertheless breach CCC propertie: even if there is no economic damage to parties P or Q, the correct execution of the communication itself still fails.

4 Classification of CCC Designs

In our model of CCC protocols in Section 2.1 we described the three phases any CCC protocol implements (after the initial setup): commit, verify, and an optional abort. We now overview techniques for implementing the different phases of CCC protocols and the possible TTP models. As shown in our impossibility result in Section 3, CCC at minimum requires either a synchrony assumption amongst communicating parties or must resort to a TTP. We hence consider assumptions of three different categories: (*i*) relying outright on a TTP, such a coordinator to facilitate the CCC process, (*ii*) relying on a synchrony assumption, and (*iii*) a hybrid approach where either a synchrony assumption needs to hold or a TTP takes over as a fallback, if the assumption fails.

4.1 (Pre-)Commit Phase

First, we discuss the different techniques to handle the commit phase of CCC protocols. In this phase, assets are typically locked on X, until the CCC is successfully executed or aborted.

4.1.1 Trusted Third Party (Coordinators) A *coordinator* (also referred to as *vault*[177]) 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 [90, 91, 164, 165, 177].
- **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 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).** Instead of relying on the availability and honest behavior of a single external coordinator, trust assumptions can be distributed among a set of N committee members. Decisions require the acknowledgment (e.g. digital signature) of at least $M \leq N$ members, whereby consensus can be achieved via Byzantine Fault Tolerant (BFT) agreement protocols such as PBFT [61, 113]. 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, 113, 114, 144] and Proof-of-Stake [44, 65, 110, 136] blockchains. Practical examples for such CCC protocols relying on committees include [13, 68, 116].
- **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, 116, 121, 170].
- **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 [140, 153]. In addition, smart contracts can be used to collateralize CCC participants, penalize misbehavior [90, 91, 164] or pay premium for correct participation [89] – even if the participants are located on alternate chains, potentially without support for smart contracts themselves [177].

4.1.2 Synchronous Assumptions (Lock Contracts) An alternative to coordinators consists in relying on synchronous communication between participants and leveraging locking mechanisms which harvest security from cryptographic hardness assumptions. Observations in practice show these techniques are used in CCC protocols that facilitate asset exchanges and implement two-phase commit, where the same (*symmetric*) locks are created on both chains and released atomically. 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, 94, 128].
- **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 [47, 49, 73, 129, 139, 145, 162].
- **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 [147] build upon inherently sequential functions where the result is only revealed after a predefined number of operations are performed. Verifiable delay functions [47] 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 [42].

4.1.3 Hybrid (Watchtowers) Instead of fully relying on coordinators being available or synchrony assumptions among participants holding, it is possible to employ so called *watchtowers*, i.e., service providers which act as a fallback if CCC participants experience crash failures. We observe strong similarities with optimistic fair exchange protocols [30, 31, 60]. 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 [33, 35, 107, 130]. This construction was first introduced and applied to off-chain payment channels [88].

4.2 Verification Phase

The verification phase, during which the commitment on X is verified on Y (or vice-versa), can similarly be executed under different trust models, as detailed in the following.

4.2.1 Trusted Third Party (Coordinators). The simplest approach to cross-chain verification is to rely on a trusted third party (also referred to as *validators* [170]) to handle the verification of the state changes on interlinked chains during CCC execution. 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, 166]. 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, 116, 125] – 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, 169], 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 [90, 105, 164].

4.2.2 Synchrony Assumption (Direct Observation). Similar to the commit phase of CCC, one can require all participants of a CCC protocol to execute the verification phase individually: i.e., to run (fully validating) nodes in all involved chains. This is often the case in exchange protocols, such as atomic swaps using symmetric locks such as HTLCs [19, 94], but also in parent-child settings where one chain by design verifies or validates the other [37, 85, 124]. 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.

4.2.3 Hybrid (Watchtowers). Just like in the commit phase, synchrony and TTP assumptions can be combined in the verification phase, such that a CCC protocol initially relies on a synchrony assumption, but can fall back to a TTP (*watchtowers*, c.f Section 4.1.3) to ensure correct termination if messages are not delivered within a per-defined period.

4.3 Abort Phase

We now discuss different techniques to handle the abort phase of CCC protocols.

4.3.1 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.

4.3.2 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 TX_2 becomes valid only after a given time value or number of *confirmations* [5] have elapsed since the inclusion of another transaction TX_1 in the underlying ledger. For example, assuming transaction TX_1 was included in $\mathsf{L}[r]$, then relatively timelocked TX_2 becomes valid when the chain reaches ledger state $\mathsf{L}[r']$ where $r' = r + c$, with c denoting the number of required confirmations ($r, r', c \in \mathbb{N}$). Typically, TX_1 and TX_2 are related as TX_2 spends assets transferred in TX_1 [146]. ALthough more practical than absolute timelocks (no need for external clock), we are not aware of schemes allowing the creation of relative timelocks across ledgers.

4.3.3 Hybrid (Watchtowers) After expiry of a timelock, the CCC protocol is aborted. However, participants 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 CCC protocols, where timelocks are used to prevent a party's funds from being frozen indefinitely in case of failure, a timely recover may be necessary. Thereby, participants must come online within some predefined period or entrust a trusted third party, e.g. a watchtower [33, 35, 107, 130], with the recovery of the locked assets. This can be the case in HTLC atomic swaps [2, 19, 94, 146], when either party crashes after the hashlock's secret has been revealed.

5 Cross-Chain State Verification

As described in Section 2.4, a critical component of cross-chain communication is the verification of the state evolution of a chain X from within another chain Y, i.e., that X is in a certain state after the commit step. In this section we discuss the different elements of the chain that can be verified during the process, to complement the process of verifying state evolution. We show that there is a classification for what is verified (Section 5.1), overview existing techniques for each class, and discuss the relation between the verification classes (Section 5.2).

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 $\mathrm{TX}_1 \in \mathsf{L}_x$ although 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, 104]: 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 [52, 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, 140]: 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 [127, 140]. 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 [40]), e.g. Merkle trees[134]. 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* [108, 109], also referred to as SPV (simplified payment verification) [140] 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, 177]. 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[2].
- **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 [85].
- **BFT.** In case the blockchain is maintained by a BFT committee, the cross-chain proofs are simplified and take the form of a sequence of signatures by $2f + 1$ members of the committee, where f is the number of faulty nodes that can be tolerated [61]. If the committee membership is dynamically changing, the verification process needs to capture the rotating configuration of the committee [141].

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 [37, 108, 109, 137] and FlyClient [127]. Both techniques rely on probabilistic sampling but differ in the selection heuristic. Superblock (Ni)PoPoWs sample blocks which exceed the required PoW difficulty[3], 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 [52]). For blockchains maintained by a static BFT committee, the verified signatures can be combined into aggregate signatures [113, 114] 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 [79, 118, 141], i.e., double-linked skiplists which enable sub-linear crawling of identity chains, can reduce costs from linear to logarithmic (to the number of configurations). Recently, a number for light client protocols that leverage the compression properties of zero-knowledge proof systems have been proposed [80, 82, 169].

Verification of State Evolution. Once verified by some chain Y that chain X has reached agreement on a ledger state $\mathsf{L}_x[r]$, it is then possible for (users on) Y to verify that certain transactions have been included in L_x. As mentioned, block headers typically reference included transactions via vector commitments. As such, to verify that $\mathsf{TX} \in \mathsf{L}_x[r]$ the vector commitment on $\mathsf{L}_x[r]$ needs to be opened at the index of that transaction, e.g. by providing a Merkle tree path to the leaf containing TX (e.g. as in Bitcoin). Thereby, multiple transactions can be aggregated in a single proof [176].

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 [46], STARKs [38] or Bulletproofs [54]. First schemes for blockchains offering such proofs for each state transition are put forth in [39, 48, 133]. 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 paths for conflicting transactions in a double spend).

Verification of Data Availability. Consensus participants may produce a block header, but not release the included transactions, preventing other participants from validating the correctness of

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

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

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, 174], which allows to verify with high probability that all 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 $\mathsf{L}[r]$ (State Evolution), it is first necessary to verify that the ledger state at L [r] is indeed agreed upon (State Agreement). Finally, to verify that a state (transition) is indeed valid (State Validity), one must first verify that all associated transactions were indeed included in the ledger (State Evolution). Verification of Data Availability serves as complimentary security measure, and can be added to any of the classes to protect against data withholding attacks. We illustrate this relationship in Figure 2.

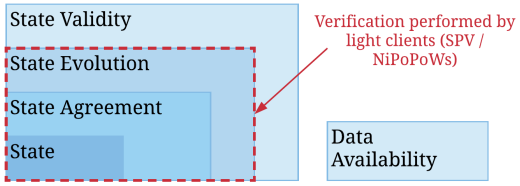

Figure 2: Venn diagram of cross-chain state verification classes. The red, dotted line highlights the minimum requirement for correctly operating light clients, i.e., verifying SPV / NIPoPoWs in the case of PoW blockchains.

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.

Methodology. We group protocols by their design rationale. Specifically, we differentiate between two CCC protocol families: (i) *exchange*, which synchronize the exchange of assets on two ledgers, and (ii) asset migration, which allow to move an asset or object to a different ledger. The main focus of the evaluation lies on how each protocol handles the impossibility result from Section 2 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 (anyone can become a TTP) or static (pre-defined set of TTPs). We also consider collateralization of TTPs in rational CCC protocols, where participants are reimbursed in case of failure, to minimize financial damages faced.

6.1 Exchange Protocols

Exchange protocols 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 implement some form of a two-phase commit mechanism [26, 95, 112, 116], where parties can explicitly abort the exchange in case of disagreement or failure during the commit step. Interestingly, 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.

We observe multiple techniques, with different underlying trust models, the simplest and currently most used being custodial (centralized) exchanges [17, 166]. Recent works, such as Tesseract [42], attempt to reduce the risk of theft by the custodian by leveraging trusted hardware, e.g. Intel SGX [98]. 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 [94]. Hashlocks can thereby be replaced with signature locks [73, 129, 139, 145, 162], 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 [166] 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 exchanges to commit theft.

An alternative to interactive swaps via HTLCs or signature locks are SPV atomic swaps [18, 19, 94, 166]. 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, permitting theft through a malicious abort by the initiator.

Recently, hybrid versions of HTLC and signature lock atomic swaps have been introduced, most notably Arwen [93] and A2L [162], 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 [146]) and introduces higher costs in the form of additional fees to prevent malicious lockup of coordinator funds (griefing).

6.2 Asset Migration Protocols

The asset migration protocols 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. Although less common than in exchange protocols, asset migration protocols may implement an explicit abort procedure, if the creation of $y(x)$ fails.

Contrary to exchange protocols, asset migration protocols require explicit involvement of the consensus participants of interlinked chains X and Y, specifically in the verification step. We hence

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 (e.g. [42, 166])**	●	✗	External Custodian (single, pre-defined)	●	✗	External Validator	●	✗	TTP (same)
		HTLC Atomic Swaps [18, 19, 94, 166]	○	-	Hash Lock	○	✗	Direct Observation	○	-	Timelock
		Notarized HTLC Atomic Swaps [166]	○	-	Hash Lock	●	✗	External Validator	◑	-	Timelock + TTP
		SPV Atomic Swaps [8, 57, 95, 112, 177]	●	✓	Smart Contract	●	✗	Smart Contract	-	-	-
		Collateralized SPV Atomic Swaps [112, 177]	●	✓	Smart Contract	●	✓	Smart Contract	-	-	-
		ECDSA/DLSAG Atomic Swaps [129, 139]	○	-	Signature Lock	○	✗	Direct Observation	○	-	Timelock
		A2L [162]	◑	✗	Multisig Escrow + Signature Lock	○	✗	Direct Observation	◑	✗	Timelock + TTP
		Arwen [93]	◑	✗	Multisig Escrow + Hash Lock	○	✗	Direct Observation	◑	✗	Timelock + TTP
Transfer	Heterogeneous	**XCLAIM [177]**	●	✓	External Custodian (single, unrestricted)	●	✓	Smart Contract	-	-	-
		Dogethereum [165]	●	✓	External Custodian (single, unrestricted)	●	✓	Smart Contract	-	-	-
		PoS Sidechains [85]	●	✗	External Custodian (Consensus of Y)	●	✗	Smart Contract	-	-	-
		tBTC [16]	●	✗	External Custodian	●	✓	Smart Contract	-	-	-
		wBTC [17]	●	✗	External Custodian (single, pre-defined)	●	✗	Direct Observation	-	-	-
	Homogeneous	**ATOMIX [116]**	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	●	✗	TTP (same)
		SBAC [23, 156]	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	●	✗	TTP (same)
		Rapidchain [175]	●	✗	Consensus Custodian (shard)	●	✗	Consensus Committee	-	-	-
		Fabric Channels [26]	●	✗	Consensus Custodian	●	✗	Consensus Committee	●	✗	TTP (same)
		Federated Sidechains/Pegs [37, 68]	●	✗	External Custodian (Consensus of Y)	●	✗	Consensus Committee	-	-	-
		PoS Sidechains [85]	●	✗	External Custodian (Consensus of Y)	●	✗	Consensus Committee	-	-	-
		Rootstock [124, 125]	●	✗	External Custodian (Consensus of Y)[†]	●	✗	Consensus Committee	-	-	-
		Proof-of-Burn [106, 158]	●	✓	Smart Contract / Burn address	●	✗	Smart Contract / Consensus Committee	-	-	-
		Merged Mining/Staking [85, 104]	●	✗	Consensus Custodian	●	✗	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.

classify asset migration into *heterogeneous* and *homogeneous*, two subfamilies defined upon the required blockchain security model.

6.2.1 Heterogeneous Asset Migration Heterogeneous asset migration protocols, in most cases, focus on transferring assets to other ledgers via *cryptocurrency-backed assets* [26, 85, 124, 177]. 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. Compared to exchange protocols, where each swap requires to broadcast transactions on all interlinked chains, cryptocurrency-backed assets require synchronization only twice: once to issue and a second time to redeem the transferred assets.

In the design of heterogeneous asset migration 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) [37].

As a result, CCC protocols in the heterogeneous model typically make additional assumptions regarding interlinked chains. 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 [177]. 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 x on X. 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 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], with the restriction that the set of custodial coordinators is pre-defined (static).

A drawback of cryptocurrency-backed assets between heterogeneous chains is the necessity to maintain a stable exchange rate between the backing assets x and 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 asset migration protocols is the lack of a dedicated abort phase on 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 Asset Migration Similar to heterogeneous asset migration, 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 observe cross-shard communication protocols [23, 25, 26, 116, 156, 175] 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 is secure by design (e.g., leveraging an appropriate form of randomness [117, 151, 160]), as otherwise the system is considered to fail ("correct by construction" assumption [116]). The main difference observed in this protocol group is the execution of an explicit abort phase in e.g. ATOMIX [116], 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, 156].

Sidechains, as first introduced by Back et al. in 2012 [37] 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[37, 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) [124, 125].

Finally, mechanisms for bootstrapping new blockchains, such as merged mining [3, 104] and Proof-of-Burn [106, 158], 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* [178] 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 [177] and Dogethereum [165], such CCC protocols utilize a pre-defined, 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, 112, 165, 177], 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 3.3). 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, the numerous techniques for constructing watchtowers for off-chain protocols [33, 35, 88, 107, 130] 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, 97, 121, 157, 167, 170]. 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 [85, 165, 177] 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 [126].

7 Challenges and Implications

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 $\alpha \leq 33\%$ [76, 86, 150]. 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 [77, 84] may be lower than that of accumulating computational power, or vice-versa [50]. 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 [32]. However, this task becomes even more difficult in the cross-chain setting: not only can consensus participants (i) "hop" between different chains [122, 135], destabilizing involved systems, but also (ii) be susceptible to bribing attacks, which can be executed cross-chain, making detection unlikely [103, 132].

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 [81]; 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, 131, 156]. Special consideration may be needed in case of permanent blockchain forks, as this may require updating the way verification is performed [131, 177].

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 [177]. 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, 168] 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 [81, 140, 143], 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 [155]. However, following recent works [179], 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 [81], hindering a seamless synchronization across chains [81, 177]. 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 [109, 127]. If an adversary can exclude this data from the chain, these protocols not only become less efficient but may potentially exhibit vulnerabilities [127]. 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 [178]. 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 [99] as its digital signature scheme and a system Y using Schnorr [152] is only possible if both schemes are instantiated over the same elliptic curve [129]. 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* [101].

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 [140, 153], to the (near) absence of scripting functionality altogether [9, 53]. CCC protocols must hence (i) consider the operations supported by both X and Y and leverage the intersecting functionality [129]; or (ii) move assets from chain with limited functionality to those with high(er) language expressiveness [68, 165, 177].

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 [87, 128] show attacks leveraging the locking mechanism and some countermeasures have been proposed [128, 129, 149]; (ii) heuristics to explore blocks from different cryptocurrencies [104] as well as forks [159] to cluster miner and user accounts [96]; (iii) CCC protocols leveraging coordinators, similar to and payment hubs [88, 102], also lead to privacy leakages that enable further account clustering [173]. Recent works [87, 92, 162] propose measures that allow to preserve the anonymity of participants, if added to CCC protocols.

8 Related Work

We believe 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. [37]. 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" [154], focusing mostly on community driven efforts, rather than academic publications. Similarly, Johnson et al. discuss open source interoperability projects related to Ethereum [100], while Robinson evaluates

Ethereum as a coordination platform for communication among other blockchains [148]. Bennik et al. [41] and similarly Miraz et al. [138] 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 [34].

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 early stages of this research. We also wish to thank the anonymous reviewers for their valuable comments that helped improve the presentation of our results.

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] [n.d.]. Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv. Accessed: 2018-05-16.
[2] [n.d.]. Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts. Accessed: 2018-05-16.
[3] [n.d.]. Bitcoin Wiki: Merged Mining Specification. https://en.bitcoin.it/wiki/Merged_mining_specification. Accessed: 2018-05-03.
[4] [n.d.]. BTCRelay. https://github.com/ethereum/btcrelay. Accessed 2019-08-15.
[5] [n.d.]. Confirmations. https://en.bitcoin.it/wiki/Confirmation. Accessed: 2018-11-28.
[6] [n.d.]. Dogerelay. https://github.com/dogethereum/dogerelay. Accessed 2019-08-15.
[7] [n.d.]. Eth-Eos-Relay. https://github.com/EveripediaNetwork/eth-eos-relay. Accessed 2019-08-15.
[8] [n.d.]. Ethereum contract allowing ether to be obtained with Bitcoin. https://github.com/ethers/EthereumBitcoinSwap. Accessed: 2018-10-30.
[9] [n.d.]. Monero Reference Implementation. https://github.com/monero-project/monero. Accessed: 2018-07-30.
[10] [n.d.]. Parity-Bridge. https://github.com/paritytech/parity-bridge. Accessed 2019-08-15.
[11] [n.d.]. The Parity Light Protocol - Wiki. https://wiki.parity.io/The-Parity-Light-Protocol-(PIP). Accessed: 2018-10-30.
[12] [n.d.]. Peace Relay. https://github.com/loiluu/peacerelay. Accessed 2019-08-15.
[13] [n.d.]. PoA Bridge. https://github.com/poanetwork/poa-bridge. Accessed: 2018-05-23.
[14] [n.d.]. Project Alchemy. https://github.com/ConsenSys/Project-Alchemy. Accessed 2019-08-15.
[15] [n.d.]. Project Waterloo. https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum-1c230ac65524. Accessed 2019-08-15.
[16] [n.d.]. tBTC: A Decentralized Redeemable BTC-backed ERC-20 Token. http://docs.keep.network/tbtc/index.pdf. Accessed: 2019-11-15.
[17] [n.d.]. Wrapped Bitcoin. https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf. Accessed: 2018-05-03.
[18] 2013. Alt chains and atomic transfers. bitcointalk.org. https://bitcointalk.org/index.php?topic=193281.msg2003765#msg2003765
[19] 2013. Atomic Swap. Bitcoin Wiki. https://en.bitcoin.it/wiki/Atomic_swap
[20] 2017. Wanchain Whitepaper. https://www.wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf.
[21] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. 2018. Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil. arXiv:1803.05069. https://arxiv.org/pdf/1803.05069.pdf
[22] Mustafa Al-Bassam. 2019. LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts. arXiv:arXiv:1905.09274 https://arxiv.org/pdf/1905.09274.pdf
[23] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2018. Chainspace: A sharded smart contracts platform. In *2018 Network and Distributed System Security Symposium (NDSS)*.
[24] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. 2018. Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities. *CoRR* abs/1809.09044 (2018). arXiv:1809.09044 http://arxiv.org/abs/1809.09044
[25] Enis Ceyhun Alp, Eleftherios Kokoris-Kogias, Georgia Fragkouli, and Bryan Ford. 2019. Rethinking General-Purpose Decentralized Computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. ACM, 105–112.
[26] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. 2018. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*. Springer, 111–131.
[27] Marcin Andrychowicz. 2015. Multiparty Computation Protocols Based on Cryptocurrencies. https://depotuw.ceon.pl/bitstream/handle/item/1327/dis.pdf Accessed: 2017-02-15.
[28] Dana Angluin, Michael J Fischer, and Hong Jiang. 2006. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems*. Springer, 37–50. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.1040&rep=rep1&type=pdf
[29] Nadarajah Asokan. 1998. Fairness in electronic commerce. (1998).
[30] Nadarajah Asokan, Victor Shoup, and Michael Waidner. 1998. Asynchronous protocols for optimistic fair exchange. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*. IEEE, 86–99.
[31] Nadarajah Asokan, Victor Shoup, and Michael Waidner. 1998. Optimistic fair exchange of digital signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 591–606.
[32] Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. 2019. Bitcoin Security under Temporary Dishonest Majority. In *23rd Financial Cryptography and Data Security (FC)*. https://www.tik.ee.ethz.ch/file/ab83461dc5ca3b739c079a27f3757e94/bitcoin%20security%20under%20temporary%20dishonest%20majority.pdf
[33] Georgia Avarikioti, Eleftherios Kokoris Kogias, and Roger Wattenhofer. 2019. Brick: Asynchronous State Channels. *arXiv preprint arXiv:1905.11360* (2019).
[34] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. 2019. Divide and Scale: Formalization of Distributed Ledger Sharding Protocols. *arXiv preprint arXiv:1910.10434* (2019).
[35] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. 2018. Towards secure and efficient payment channels. *arXiv preprint arXiv:1811.12740* (2018).
[36] Ozalp Babaoglu and Sam Toueg. 1993. Understanding non-blocking atomic commitment. *Distributed systems* (1993), 147–168.
[37] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. 2014. Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf
[38] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive* 2018 (2018), 46.

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

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

[41] Peter Bennink, Lennart van Gijtenbeek, Oskar van Deventer, and Maarten Everts. 2018. An Analysis of Atomic Swaps on and between Ethereum Blockchains using Smart Contracts. Tech. report. https://work.delaat.net/rp/2017-2018/p42/report.pdf.

[42] Iddo Bentov, Yan Ji, Fan Zhang, Yunqi Li, Xueyuan Zhao, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2017. Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware. Cryptology ePrint Archive, Report 2017/1153. https://eprint.iacr.org/2017/1153.pdf Accessed:2017-12-04.

[43] Iddo Bentov and Ranjit Kumaresan. 2014. How to use bitcoin to design fair protocols. In *Advances in Cryptology–CRYPTO 2014*. Springer, 421–439. http://eprint.iacr.org/2014/129.pdf

[44] Iddo Bentov, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. https://eprint.iacr.org/2016/919.pdf Accessed: 2016-11-08.

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

[46] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. 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*. ACM, 326–349.

[47] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable Delay Functions. In *CRYPTO*.

[48] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2018. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. Cryptology ePrint Archive, Report 2018/1188. https://eprint.iacr.org/2018/1188.pdf https://eprint.iacr.org/2018/1188.

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

[50] Joseph Bonneau. 2016. Why buy when you can rent? Bribery attacks on Bitcoin consensus. In *BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research*. http://fc16.ifca.ai/bitcoin/papers/Bon16b.pdf

[51] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. On Bitcoin as a public randomness source. https://eprint.iacr.org/2015/1015.pdf Accessed: 2015-10-25.

[52] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive* 2015 (2015), 1015.

[53] Joseph Bonneau and Andrew Miller. 2014. Fawkescoin: Bitcoin without public-key crypto. In *Security Protocols XXII*. Springer, 350–358. http://www.jbonneau.com/doc/BM14-SPW-fawkescoin.pdf

[54] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2017. Bulletproofs: Efficient Range Proofs for Confidential Transactions. http://web.stanford.edu/~buenz/pubs/bulletproofs.pdf Accessed:2017-11-10.

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

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

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

[58] Vitalik Buterin. 2018. Cross-shard contract yanking. https://ethresear.ch/t/cross-shard-contract-yanking/1450.

[59] Christian Cachin. 2016. Architecture of the Hyperledger Blockchain Fabric. https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf Accessed: 2016-08-10.

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

[61] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186. http://pmg.csail.mit.edu/papers/osdi99.pdf

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

[63] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)* 43, 2, 225–267. https://ecommons.cornell.edu/bitstream/handle/1813/7192/95-1535.pdf?sequence=1

[64] Alexander Chepurnoy, Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2017. TwinsCoin: A Cryptocurrency via Proof-of-Work and Proof-of-Stake. http://eprint.iacr.org/2017/232.pdf Accessed: 2017-03-22.

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

[66] Christian Decker and Roger Wattenhofer. 2014. Bitcoin transaction malleability and MtGox. In *Computer Security-ESORICS 2014*. Springer, 313–326. http://www.tik.ee.ethz.ch/file/7e4a7f3f2991784786037285f4876f5c/malleability.pdf

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

[68] Johnny Dilley, Andrew Poelstra, Jonathan Wilkins, Marta Piekarska, Ben Gorlick, and Mark Friedenbach. 2016. Strong Federations: An Interoperable Blockchain Solution to Centralized Third Party Risks. *arXiv preprint arXiv:1612.05491* (2016).

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

[70] Tuyet Duong, Lei Fan, and Hong-Sheng Zhou. 2016. 2-hop Blockchain: Combining Proof-of-Work and Proof-of-Stake Securely. Cryptology ePrint Archive, Report 2016/716. https://eprint.iacr.org/2016/716.pdf Accessed: 2017-02-06.

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

[72] Jacob Eberhardt and Stefan Tai. [n.d.]. ZoKrates-Scalable Privacy-Preserving Off-Chain Computations. http://www.ise.tu-berlin.de/fileadmin/fg308/publications/2018/2018_eberhardt_ZoKrates.pdf. ([n. d.]).

[73] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *CCS*.

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

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

[76] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*. Springer, 436–454. http://arxiv.org/pdf/1311.0243

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

[78] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2, 374–382. http://macs.citadel.edu/rudolphg/csci604/ImpossibilityofConsensus.pdf

[79] Bryan Ford, Linus Gasser, Eleftherios Kokoris Kogias, and Philipp Jovanovic. 2018. Cryptographically Verifiable Data Structure Having Multi-Hop Forward and Backwards Links and Associated Systems and Methods. US Patent App. 15/618,653.

[80] Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, Michael Straka, and Eran Tromer. 2020. Plumo: Towards Scalable Interoperable Blockchains Using Ultra Light Validation Systems. (2020).

[81] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2016. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. http://eprint.iacr.org/2016/1048.pdf Accessed: 2017-02-06.

[82] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. 2020. Zendoo: a zk-SNARK Verifiable Cross-Chain Transfer Protocol Enabling Decoupled and Decentralized Sidechains. *arXiv preprint arXiv:2002.01847* (2020).

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

[84] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-Bleeding Attacks on Proof-of-Stake Blockchains. Cryptology ePrint Archive, Report 2018/248. https://eprint.iacr.org/2018/248.pdf Accessed:2018-03-12.

[85] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Stake Sidechains. *IEEE Security and Privacy. IEEE* (2019).

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

[87] Matthew Green and Ian Miers. 2016. Bolt: Anonymous Payment Channels for Decentralized Currencies. Cryptology ePrint Archive, Report 2016/701. http://eprint.iacr.org/2016/701 Accessed: 2017-08-07.

[88] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2019. SoK: Off The Chain Transactions. Cryptology ePrint Archive, Report 2019/360. https://eprint.iacr.org/2019/360.pdf https://eprint.iacr.org/2019/360.

[89] Runchao Han, Haoyu Lin, and Jiangshan Yu. 2019. On the optionality and fairness of Atomic Swaps. Cryptology ePrint Archive, Report 2019/896. https://eprint.iacr.org/2019/896.

[90] Dominik Harz and Magnus Boman. 2018. The Scalability of Trustless Trust. arXiv:1801.09535. https://arxiv.org/pdf/1801.09535.pdf Accessed:2018-01-31.

[91] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J. Knottenbelt. 2019. Balance: Dynamic Adjustment of Cryptocurrency Deposits. Cryptology ePrint Archive, Report 2019/675. https://eprint.iacr.org/2019/675.pdf https://eprint.iacr.org/2019/675.

[92] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2016. TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. https://eprint.iacr.org/2016/575.pdf Accessed: 2017-09-29.

[93] Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. [n.d.]. The Arwen Trading Protocols. Whitepaper. https://www.arwen.io/whitepaper.pdf.

[94] Maurice Herlihy. 2018. Atomic Cross-Chain Swaps. arXiv:1801.09515. https://arxiv.org/pdf/1801.09515.pdf Accessed:2018-01-31.

[95] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. 2019. Cross-chain Deals and Adversarial Commerce. *arXiv preprint arXiv:1905.09743* (2019).

[96] Abraham Hinteregger and Bernhard Haslhofer. 2018. An empirical analysis of monero cross-chain traceability. *arXiv preprint arXiv:1812.02808* (2018).

[97] Dr Hosp, Toby Hoenisch, Paul Kittiwongsunthorn, et al. 2018. COMIT-Cryptographically-secure Off-chain Multi-asset Instant Transaction Network. *arXiv preprint arXiv:1810.02174* (2018).

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

[99] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

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

[101] John Jones and abitmore. [n.d.]. Optional HTLC preimage length and HASH160 addition. BSIP 64, blog post. https://github.com/bitshares/bsips/issues/163.

[102] Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. 2019. SoK: A Taxonomy for Layer-2 Scalability Related Protocols for Cryptocurrencies. Cryptology ePrint Archive, Report 2019/352. https://eprint.iacr.org/2019/352.pdf https://eprint.iacr.org/2019/352.

[103] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar Weippl. 2019. Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies. Cryptology ePrint Archive, Report 2019/775. https://eprint.iacr.org/2019/775.pdf https://eprint.iacr.org/2019/775.

[104] Aljosha Judmayer, Alexei Zamyatin, Nicholas Stifter, Artemios G. Voyiatzis, and Edgar Weippl. 2017. Merged Mining: Curse or Cure?. In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*. https://eprint.iacr.org/2017/791.pdf

[105] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, 1353–1370.

[106] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Burn. International Conference on Financial Cryptography and Data Security.

[107] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. 2019. Outpost: A Responsive Lightweight Watchtower. (2019).

[108] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. 2016. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer, 61–78.

[109] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. 2017. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963. https://eprint.iacr.org/2017/963.pdf Accessed:2017-10-03.

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

[111] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. 2016. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 705–734. https://eprint.iacr.org/2015/574.pdf

[112] Aggelos Kiayias and Dionysis Zindros. 2018. Proof-of-Work Sidechains.. In *International Conference on Financial Cryptography and Data Security*. Springer.

[113] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX. http://arxiv.org/pdf/1602.06997.pdf

[114] Eleftherios Kokoris-Kogias. 2019. *Robust and Scalable Consensus for Sharded Distributed Ledgers*. Technical Report. Cryptology ePrint Archive, Report 2019/676.

[115] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gailly, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. 2018. *Calypso: Auditable sharing of private data over blockchains*. Technical Report. Cryptology ePrint Archive, Report 2018/209.

[116] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[117] Eleftherios Kokoris-Kogias, Alexander Spiegelman, Dahlia Malkhi, and Ittai Abraham. 2019. *Bootstrapping Consensus Without Trusted Setup: Fully Asynchronous Distributed Key Generation*. Technical Report. https://eprint.iacr.org/2019/1015.pdf https://eprint.iacr.org/2019/1015.

[118] Lefteris Kokoris-Kogias, Linus Gasser, Ismail Khoffi, Philipp Jovanovic, Nicolas Gailly, and Bryan Ford. 2016. Managing identities using blockchains and CoSi. In *9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016)*.

[119] Ranjit Kumaresan and Iddo Bentov. 2016. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 418–429.

[120] Alptekin Küpçü and Anna Lysyanskaya. 2012. Usable optimistic fair exchange. *Computer Networks* 56, 1 (2012), 50–63.

[121] Jae Kwon and Ethan Buchman. 2015. Cosmos: A Network of Distributed Ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md.

[122] Yujin Kwon, Hyoungshick Kim, Jinwoo Shin, and Yongdae Kim. 2019. Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash? arXiv:1902.11064. https://arxiv.org/pdf/1902.11064.pdf

[123] Leslie Lamport. 1989. A simple approach to specifying concurrent systems. *Commun. ACM* 32, 1 (1989), 32–45.

[124] Sergio Demian Lerner. 2018. *Drivechains, Sidechains and Hybrid 2-Way Peg Designs*. Technical Report. Tech. Rep. [Online]. https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf

[125] S Demian Lerner. 2015. Rootstock: Bitcoin powered smart contracts. https://docs.rsk.co/RSK_White_Paper-Overview.pdf.

[126] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, and Yih-Chun Hu. 2019. HyperService: Interoperability and Programmability Across Heterogeneous Blockchains. *arXiv preprint arXiv:1908.09343* (2019).

[127] Loi Luu, Benedikt Buenz, and Mahdi Zamani. [n.d.]. Flyclient Super light client for cryptocurrencies. ([n. d.]). https://stanford2017.scalingbitcoin.org/files/Day1/flyclientscalingbitcoin.pptx.pdf Accessed 2018-04-17.

[128] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *CCS*. 455–471.

[129] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *NDSS*.

[130] Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. 2018. Pisa: Arbitration Outsourcing for State Channels. *IACR Cryptology ePrint Archive* 2018 (2018), 582.

[131] Patrick McCorry, Ethan Heilman, and Andrew Miller. 2017. Atomically Trading with Roger: Gambling on the success of a hardfork. In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*. http://homepages.cs.ncl.ac.uk/patrick.mc-corry/atomically-trading-roger.pdf

[132] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. 2018. Smart Contracts for Bribing Miners. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*. http://fc18.ifca.ai/bitcoin/papers/bitcoin18-final14.pdf

[133] Izaak Meckler and Evan Shapiro. 2018. Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf.

[134] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 369–378.

[135] Dmitry Meshkov, Alexander Chepurnoy, and Marc Jansen. 2017. Revisiting Difficulty Control for Blockchain Systems. Cryptology ePrint Archive, Report 2017/731. http://eprint.iacr.org/2017/731.pdf Accessed: 2017-08-03.

[136] Silvio Micali. 2016. ALGORAND: The Efficient and Democratic Ledger. https://arxiv.org/pdf/1607.01341.pdf Accessed: 2017-02-09.

[137] Andrew Miller. 2012. *The high-value-hash highway, Bitcoin forum post*. https://bitcointalk.org/index.php?topic=98986.0

[138] Mahdi Miraz and David C Donald. 2019. 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).

[139] Pedro Moreno-Sanchez, Randomrun, Duc V. Le, Sarang Noether, Brandon Goodell, and Aniket Kate. 2019. DLSAG: Non-Interactive Refund Transactions For Interoperable Payment Channels in Monero. Cryptology ePrint Archive, Report 2019/595. https://eprint.iacr.org/2019/595.

[140] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf Accessed: 2015-07-01.

[141] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. [n.d.]. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds.

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

[143] Rafael Pass, Lior Seeman, and abhi shelat. 2016. Analysis of the Blockchain Protocol in Asynchronous Networks. http://eprint.iacr.org/2016/454.pdf Accessed: 2016-08-01.

[144] Rafael Pass and Elaine Shi. 2016. Hybrid Consensus: Scalable Permissionless Consensus. https://eprint.iacr.org/2016/917.pdf Accessed: 2016-10-17.

[145] Andrew Poelstra. [n.d.]. Scriptless Scripts. Presentation slides. https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf.

[146] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network. https://lightning.network/lightning-network-paper.pdf Accessed: 2016-07-07.

[147] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).

[148] Peter Robinson. 2019. The merits of using Ethereum MainNet as a Coordination Blockchain for Ethereum Private Sidechains. arXiv:arXiv:1906.04421 https://arxiv.org/pdf/1906.04421.pdf

[149] Jeremy Rubin, Manali Naik, and Nitya Subramanian. 2014. Merkelized Abstract Syntax Trees. http://www.mit.edu/~jlrubin/public/pdfs/858report.pdf.

[150] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 2015. Optimal selfish mining strategies in Bitcoin. http://arxiv.org/pdf/1507.06183.pdf Accessed: 2016-08-22.

[151] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2018. HydRand: Practical Continuous Distributed Randomness. Cryptology ePrint Archive, Report 2018/319. https://eprint.iacr.org/2018/319.pdf

[152] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.

[153] Ilya Sergey, Amrit Kumar, and Aquinas Hobor. 2018. Scilla: a Smart Contract Intermediate-Level LAnguage. arXiv:1801.00687. https://arxiv.org/pdf/1801.00687.pdf Accessed:2018-01-08.

[154] Vasilios A. Siris, Dimitrios Dimopoulos, Nikos Fotiou, Spyros Voulgaris, and George C. Polyzos. 2019. Interledger Smart Contracts for Decentralized Authorization to Constrained Things. arXiv:arXiv:1905.01671 https://arxiv.org/pdf/1905.01671.pdf

[155] Yonatan Sompolinsky and Aviv Zohar. 2016. Bitcoin's Security Model Revisited. http://arxiv.org/pdf/1605.09193.pdf Accessed: 2016-07-04.

[156] Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, and George Danezis. 2019. Replay Attacks and Defenses Against Cross-shard Consensus in Sharded Distributed Ledgers. *arXiv preprint arXiv:1901.11218* (2019).

[157] Matthew Spoke and Nuco Engineering Team. [n.d.]. AION: The third-generation blockchain network. https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf. Accessed 2018-04-17.

[158] Iain Stewart. 2012. Proof of burn. https://en.bitcoin.it/wiki/Proof_of_burn Accessed: 2017-05-10.

[159] Nicholas Stifter, Philipp Schindler, Aljosha Judmayer, Alexei Zamyatin, Andreas Kern, and Edgar Weippl. 2019. 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. https://fc19.ifca.ai/preproceedings/41-preproceedings.pdf

[160] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. 2017. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 444–460.

[161] Paul Syverson. 1998. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Proceedings. 11th IEEE Computer Security Foundations Workshop (Cat. No. 98TB100238)*. IEEE, 2–13.

[162] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. A^2L: Anonymous Atomic Locks for Scalability and Interoperability in Payment Channel Hubs. Cryptology ePrint Archive, Report 2019/589. https://eprint.iacr.org/2019/589.

[163] Jason Teutsch and TrueBit Estsblishment. 2017. On decentralized oracles for data availability. (2017).

[164] Jason Teutsch and Christian Reitwießner. 2017. A scalable verification solution for blockchains. https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf Accessed:2017-10-06.

[165] Jason Teutsch, Michael Straka, and Dan Boneh. 2018. *Retrofitting a two-way peg between blockchains*. Technical Report. https://people.cs.uchicago.edu/~teutsch/papers/dogethereum.pdf

[166] Stefan Thomas and Evan Schwartz. 2015. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf* (2015).

[167] Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. 2018. Quant overledger whitepaper. https://www.quant.network/wp-content/uploads/2018/09/Quant_Overledger_Whitepaper-Sep.pdf.

[168] Marko Vukolic. 2016. Eventually Returning to Strong Consistency. https://pdfs.semanticscholar.org/a6a1/b70305b27c556aac779fb65429db9c2e1ef2.pdf Accessed: 2016-08-10.

[169] Martin Westerkamp and Jacob Eberhardt. 2020. zkRelay: Facilitating Sidechains using zkSNARK-based Chain-Relays. *Contract* 1, 2 (2020), 3.

[170] Gavin Wood. 2015. Polkadot: Vision for a heterogeneous multi-chain framework. PolkaDotPaper.pdf. *White Paper* (2015).

[171] Gavin Wood. 2017. Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION (759dccd - 2017-08-07). https://ethereum.github.io/yellowpaper/paper.pdf Accessed: 2018-01-03.

[172] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 162–167.

[173] Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. 2019. Tracing Transactions Across Cryptocurrency Ledgers. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 837–850.

[174] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. 2019. Coded Merkle Tree: Solving Data Availability Attacks in Blockchains. *arXiv preprint arXiv:1910.01247* (2019).

[175] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: A Fast Blockchain Protocol via Full Sharding. Cryptology ePrint Archive, Report 2018/460. https://eprint.iacr.org/2018/460.pdf

[176] Alexei Zamyatin, Zeta Avarikioti, Daniel Perez, and William J Knottenbelt. 2020. TxChain: Efficient Cryptocurrency Light Clients via Contingent Transaction Aggregation. *IACR Cryptology ePrint Archive 2020/580* (2020).

[177] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. 2019. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. *IEEE Security and Privacy. IEEE* (2019).

[178] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, and William J. Knottebelt. 2018. (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.* https://eprint.iacr.org/2018/087.pdf

[179] Dionysis Zindros. 2019. Summa Proofs Are Not Composable. https://medium.com/@dionyziz/summa-proofs-are-not-composable-57b87825f428

A Strong Fair Exchange Definition

This section provides the definition of the strong Fair Exchange problem, as presented in [29–31, 142].

Fair Exchange considers two processes (or parties) P and Q that wish to exchange two items (or asset): a_P owned by P against a_Q owned by Q. There exists a function *desc* that maps any exchangeable item (or asset) to a string describing it in sufficient" detail (e.g. the value and recipient of a payment). The inputs of P to a Fair Exchange protocol are an item a_P and a description d_Q of the desired item. Analogous, the inputs for Q are a_Q and d_Q. To indicate that P is dishonest, an (boolean) error variable m_P is introduced (analogous, m_Q for Q) [83]. A successful Fair Exchange is shown in Table 2 below.

Table 2: A successful Fair Exchange, as defined in [29–31, 142].

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

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

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

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

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

Effectiveness determines the outcome of the exchange if P and Q are willing to perform the exchange and the item's match the expected descriptions, or the items do not match the expected descriptions. (Strong) Fairness restricts the outcomes of the exchange such that neither party is left at a disadvantage. Timeliness ensures eventual termination of the exchange protocol. Note: we do not provide a definition for "Non-repudiability" as this property is not a critical requirement for Fair Exchange protocols, but only becomes relevant in disputes after an exchange [30, 142].

B Fair Exchange using CCC

We provide the intuition of how to construct a Fair Exchange protocol using a generic CCC protocol in Algorithm 1. Specifically, P and Q exchange assets a_P and a_Q, if transaction TX_P is written to L_x and transaction TX_Q is written to L_y (cf. Section 3.1).

Algorithm 1: Fair Exchange using a generic CCC protocol

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

setup(L_x,L_y,TX_P, TX_Q, d_P, d_Q);

if m_P = false **then**
  commit(TX_P, L_x); // P transfers a_P to Q
end
if (*verify*(TX_P, L_x, d_P) = true) $\wedge$ m_Q = false **then**
  commit(TX_Q, L_y); // Q transfers a_Q to P
else
  abort(TX_Q, L_y); // Q does not transfer a_Q to P
end
if *verify*(TX_Q, L_y, d_Q) = false **then**
  abort(TX_P, L_x); // P recovers a_P
end

Algorithm 2: Commit(TX, L)

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

Algorithm 3: Verify(TX, L, d)

if $\text{TX} \in \mathsf{L} \wedge desc(\text{TX}) = d$ **then**
  return true;
end
return false;

Algorithm 4: Abort(TX, L)

if $\text{TX} \in \mathsf{L}$ **then**
  Revert TX; // e.g. using a new transaction
else
  //do nothing
end