SoK: Communication Across Distributed Ledgers

Alexei Zamyatin[1,2], Mustafa Al-Bassam[3], Dionysis Zindros[4,7], Eleftherios Kokoris-Kogias[5], Pedro Moreno-Sanchez[6], and Aggelos Kiayias[7,8], and William J. Knottenbelt[1]

[1] Imperial College London
[2] Interlay
[3] University College London
[4] University of Athens
[5] IST Austria
[6] IMDEA Software Institute
[7] IOHK
[8] University of Edinburgh

Abstract. Since the inception of Bitcoin, a plethora of distributed ledgers differing in design and purpose has been created. While by design, blockchains provide no means to securely communicate with external systems, numerous attempts towards trustless cross-chain communication have been proposed over the years. Today, cross-chain communication (CCC) plays a fundamental role in cryptocurrency exchanges, scalability efforts via sharding, extension of existing systems through sidechains, and bootstrapping of new blockchains. Unfortunately, existing proposals are designed ad-hoc for specific use-cases, making it hard to gain confidence in their correctness and composability.

We provide the first systematic exposition of cross-chain communication protocols. 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. With this result in mind, we develop a framework to design new and evaluate existing CCC protocols, focusing on the inherent trust assumptions thereof, and derive a classification covering the field of cross-chain communication to date. We conclude by discussing open challenges for CCC research and the implications of interoperability on the security and privacy of blockchains.

1 Introduction

Since the introduction of Bitcoin [138] 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 there shall emerge a "coin to rule them all", yielding interoperability an important research problem.

Today, cross-chain communication is found not only in research on cryptocurrency transfers and exchanges [20, 19, 94, 93, 173], but is a critical component of scalabilty solutions such as sharding [115, 28, 171, 27, 30], feature extensions via sidechains [40, 111, 86, 122], as well as bootstrapping of new systems [102, 155, 104]. In practice, over $1bn worth of Bitcoin has been moved to other blockchains [22], and numerous competing interoperability projects, attempting to unite independent systems, have been deployed to practice [162, 166, 119, 153, 96, 163, 21], creating a multi-million dollar industry.

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. Early attempts to overview this field offer iterative summaries of mostly community-lead efforts [59, 150, 98], or focus on a subset of this space, such as atomic swaps [44, 136], and support our study.

This work. This Systematization of Knowledge (SoK) offers a comprehensive guide for designing protocols bridging the numerous distributed ledgers available today, aiming to facilitate clearer communication between academia, community, and industry. The contributions of this work are thereby twofold:

– We formalize the underlying problem of Correct Cross-Chain Communication (CCC) (Section 2), relating CCC to existing research and outlining a generic CCC protocol encompassing existing solutions. We then relate CCC to the Fair Exchange problem and show that contrary to common beliefs in the blockchain community, CCC is *impossible without a trusted third party* (Section 3).

– With the impossibility result in mind, we present a framework to design new and evaluate existing CCC protocols, focusing on the inherent trust assumptions thereof (Sections 4). We apply this framework to classify the field of CCC protocols to date (Section 5), highlighting similarities and key differences. Finally, we outline general observations on current developments, provide an outlook on the challenges of CCC research, and discuss the implications of interoperability on the security and privacy of blockchains (Section 6).

2 The Cross-Chain Communication Problem

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

2.1 Historical Background: Distributed Databases

The need for communication among distributed processes is fundamental to any distributed computing algorithm. In databases, to ensure the atomicity of a distributed transaction, an agreement problem must be solved among the set of participating processes. Referred to as the Atomic Commit problem (AC) [48],

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) [39]. 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*.

2.2 Distributed Ledger Model

We use the terms *blockchain* and *distributed ledger* as synonyms and introduce some notation, based on [86] 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. [1]. The data structures underlying X and Y are *blockchains* (or *chains*), i.e., append-only sequences of blocks, where each block contains a reference to its predecessor(s). 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. 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 [138] and Ethereum [58]). 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

[1] More specifically, in the case of Proof-of-Work (Bitcoin) or Proof-of-Stake blockchains, the majority pertains to computational power [138] or stake [109].

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 [82]:

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., $\exists t' \in \mathbb{N} : t' \geq t \wedge \text{TX} \in \mathsf{L}^P[t']$. The interval $t' - t$ is* 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 [167]. For generality, we do not differentiate between specifics transactions models (e.g. UTXO [138] or account-based models [167]).

2.3 Cross-Chain Communication System Model

Consider two independent distributed systems X and Y with underlying ledgers L_x and L_y, as defined in Section 2.2. We assume a *closed* system model as in [121] 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, $d_P = \mathit{desc}(\text{TX}_P)$ 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" [84] 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.

2.4 Formalization of Correct Cross-Chain Communication

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

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

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

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

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

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

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

2.5 The Generic CCC Protocol

We now describe the main phases of a generic CCC protocol, which can represent the transfer of good, assets or objects, between any two blockchain-based distributed systems X and Y. A visual representation is provided in Figure 1.
1) Setup. A CCC protocol is parameterized by the involved distributed systems X and Y and the corresponding ledgers L_x and L_y, the involved parties P and Q, the transactions TX_P and TX_Q as well as their descriptions d_P and d_Q. The latter ensure the validity of TX_P and TX_Q and determine the application-level specification of a CCC protocol. For example, in the case of an exchange of digital assets, d_P and d_Q define the asset types, transferred value, time constraints and any additional conditions agreed by parties P and Q. Typically, the setup occurs out-of-band between the involved parties and we hence omit this step hereby.
2) (Pre-)Commit on X. Upon successful setup, a publicly verifiable commitment to execute the CCC protocol is published on X: P writes[2] 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 $r+u_x+k_x$.
3) Verify. The correctness of the commitment on X by P is verified by Q checking (or receiving a proof from P) that (i) $d_P = \mathit{desc}(\mathrm{TX}_P)$ 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 fails to execute the commitment on Y, a CCC protocol can exhibit an abort step on X, i.e., "reverting" the modifications TX_P made to the state of L_x. As blockchains are append-only data

[2] In off-chain protocols [89], the commitment can be done by exchanging pre-signed transactions or channel states, which will be written to the ledger at a later point.

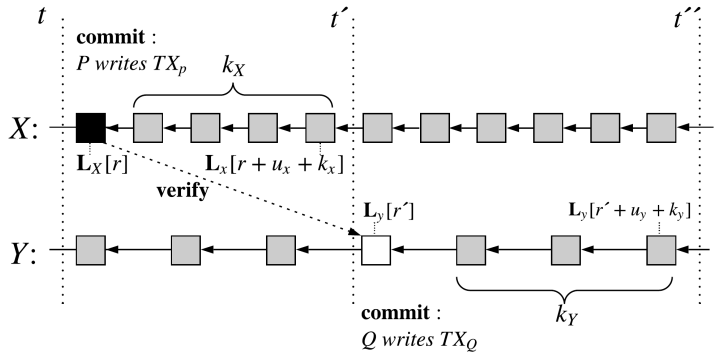

Fig. 1: CCC between X and Y. Process Q writes TX_Q only if P has written TX_P. We set exemplary persistence delays for X and Y as $k_X = 4$ and $k_Y = 3$, and liveness delays as $u_x = u_y = 0$. We omit the optional the abort phase.

structures, reverting requires broadcasting an additional transaction $\text{TX}_{P'}$ which resets X to the state before the commitment of TX_P.

It is worth noting that some CCC protocols, specifically those facilitating *exchange* of assets, follow a two-phase commit design. In this case, steps 2 and 4a are executed in parallel, followed by the verification and (optional) abort steps on *both* X and Y. A further observation is that a CCC protocol necessarily requires a *conditional state transition* to occur on Y, given a state transition on X. As such, we do *not* consider (oracle) protocols which merely relay data across distributed ledgers [161, 54, 57, 5, 59], as CCC protocols by themselves.

3 Impossibility of CCC without a Trusted Third Party

In this section we show that CCC is impossible without a trusted third party by reducing it to the Fair Exchange problem [33, 140].

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 [35]. Fair exchange can be considered a sub-problem of fair secure computation [46], and is known to be impossible without a trusted third party [140, 168, 76, 75]. We recall the definition of Fair Exchange in Appendix A.

3.1 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 [46, 31, 117, 110, 72, 114]. They leverage smart contracts (i.e., programs or scripts), the result of which is agreed upon and enforced by consensus participants, to ensure the correctness of the exchange. These protocols thus use the consensus of the distributed ledgers as an abstraction for a trusted third party. If the majority of consensus

participants are honest, correct behavior of processes/participants of the fair exchange is enforced – typically, the correct release of a_Q to P if Q received a_P.

A CCC protocol aims to achieve synchronization between *two* such distributed ledgers, both of which are inherently trusted to operate correctly. As we show below, a (possibly additional) TTP can be used to (i) 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 $\text{TX}_Q \in \mathsf{L}_y$.

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. Strictly speaking, from the perspective of CCC there is little difference between a TTP consisting of a single individual and a committee where N out of M members must agree to take action (even though a committee is, without question, more resilient against failures) – contrary to the common assumptions made by the blockchain community.

3.2 Relating CCC to Fair Exchange.

We proceed to show that Correct Cross-Chain Communication is impossible without a trusted third party (TTP), under the deterministic system model of distributed ledgers, by reducing CCC to Fair Exchange [35, 33, 140]. We recall, a fair exchange protocol must fulfill three properties: *Effectiveness*, *(Strong) Fairness* and *Timeliness* [140, 33] (cf. Appendix A).

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 TX_P assigns ownership of a_P to Q and TX_Q transfers ownership of a_Q to P (specified in the "descriptions" d_P of TX_P and d_Q of TX_Q). Then, TX_P must be included in L_x and TX_Q must be included in L_y to correctly execute the exchange. In other words, if $\text{TX}_Q \in \mathsf{L}_y$ and $\text{TX}_P \in \mathsf{L}_x$, then P receives desired a_Q and Q receives desired a_P, i.e., P and Q fairly exchange a_P and a_Q.

We observe the definition of Timeliness in CCC is equivalent to the definition of Timeliness in fair exchange protocols, as defined in [140]. 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 [140]. It is easy to see Effectiveness in CCC achieves exactly this property: if P and Q behave correctly and $desc(\text{TX}_P) = d_P$ and $desc(\text{TX}_Q) = d_Q$, i.e., TX_P transfers a_P to Q and TX_Q transfers a_Q to P, then P will write TX_P to L_y at time t and Q will write TX_Q to L_x before time t'. From Persistence and Liveness of L_x and L_y we know both transactions will eventually be written to the local ledgers of P and Q, consequently all other honest participants of X will report $\text{TX}_P \in \mathsf{L}_X$ and honest participants of Y will report $\text{TX}_Q \in \mathsf{L}_Y$. From our

model we know that honest participants constitute majorities in both X and Y. Hence, P will receive a_Q and Q will receive a_P.

Strong Fairness in fair exchange states that there is no outcome of the protocol, where P receives a_Q but Q does not receive a_P, or, vice-versa, Q receives a_P but P does not receive a_Q [140]. In our setting, such an outcome is only possible if $\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \notin \mathsf{L}_y$ or $\text{TX}_P \notin \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y$, which contradicts the Atomicity property of CCC.

We construct a protocol for Fair Exchange using CCC in Appendix B. It is left to show that CCC is defined under the same model as Fair Exchange. The distributed ledger model [82] used in CCC assumes the same asynchronous (explicitly) and deterministic (implicitly) system model (cf. Section 2.3) as [140, 79]. Since P and Q by definition can stop participating in the CCC protocol at any time, CCC exhibits the same crash failure model as Fair Exchange [34, 140] (and in turn Consensus [79]). 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 [140, 79]. 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 [115]. In the case of CCC, this means that distributed system Y, or more precisely, the consensus of Y, can write TX_Q to L_y on behalf of Q (if P wrote TX_P to L_x). We observe that the consensus of Y becomes the TTP in this scenario: both P and Q must agree that the consensus of Y enforce correct execution of CCC. In practice, this can be achieved by leveraging smart contracts, similar to blockchain-based fair exchange protocols, e.g. [72]. As such, we can construct a smart contract, the execution of which is enforced by consensus of Y, that will write TX_Q to L_y if P includes TX_P in L_x, i.e., Q is allowed to crash.

However, it remains the question how the consensus participants of Y become aware that $\text{TX}_P \in \mathsf{L}_x$. In practice, a smart contract, can only perform actions based on some input. As such, before writing TX_Q the contract / consensus of Y must observe and verify that TX_P was included in L_x. A protocol achieving CCC must hence make one of the following assumptions. Either, there exists a TTP that will ensure correct execution of CCC; or the protocol assumes P, or Q, or some other honest, online party (this can again be consensus of Y) will always deliver a proof for $\text{TX}_P \in \mathsf{L}_x$ to Y within a known, upper bounded delay, i.e., the protocol introduces some form of *synchrony* assumption. As argued in [140], we observe that *introducing a TTP and relying on a synchrony assumption 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 $\text{TX}_P \in \mathsf{L}_x$ and informs the smart contract or directly enforces the inclusion of TX_Q in L_y.

3.3 Incentives and Rational CCC

Several workarounds to the fair exchange problem, including gradual release mechanisms, optimistic models, and partially fair secure computation [35, 62, 118, 46], have been suggested in the literature. These workarounds suffer, among others, from a common drawback: they require some form of trusted party that does not collude with the adversary. Further, in case of a adversary-caused abort, honest parties must spend extra efforts to restore fairness, e.g., in the optimistic model the trusted server must be contacted each time fairness is breached.

First suggested in the context of rational exchange protocols [157], 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 [173]. We remark that malicious/altruistic actors can nevertheless breach CCC properties: even if there is no economic damage to parties P or Q, the correct execution of the communication protocol itself still fails.

4 The CCC Design Framework

With the impossibility result 3 and CCC model (Section 2.2) in mind, we now introduce a new framework for creating and evaluating CCC protocols.

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

4.1 (Pre-)Commit Phase

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

– *Custody of Assets.* Custody determines with whom the control over assets of (honest) participants resides. We differentiate between *custodians* and *escrows*. *Custodians* receive *unconditional* control over the participant's funds and are thus *trusted* to release them as instructed by the protocol rules. *Escrows* receive control over the participant's funds *conditional* to certain prearranged constraints being fulfilled. Contrary to custodians, escrows can fail to take action, e.g. freeze assets, but cannot commit theft.

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

– *Election.* An important distinction to make is between *static*, i.e., unchanged over time (usually permissioned), and *dynamic* coordinators. A dynamic coordinator can be chosen by CCC participants for each individual execution, or can be sampled by a pre-defined mechanism, as e.g. studied in [67, 112, 113, 142] for Proof-of-Work and in [134, 109, 66, 47] for Proof-of-Stake blockchains. We consider CCC protocols where any user can become a coordinator as *unrestricted* [173], while protocols that require coordinators to register with some third party (or e.g. first acquire a token) as *restricted* [17].

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

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

– *External Custodians (Committees).* Instead of relying on the availability and honest 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 [63, 112]. External custodians can be both static or dynamic, and collateralization can be added on involved blockchains to incentivize honest behavior.

– *Consensus-level Custodians (Consensus Committee)* are identical to external custodians, except that they are also responsible for agreeing on the state of the underlying ledger. This model is typically used in blockchain sharding [115, 28], where the blockchain X on which the commit step is executed runs a BFT consensus protocol, i.e., there already exists a *static* committee of consensus participants that much be trusted for correctness of CCC (Persistence and Liveness of X). Collateralization of Consensus Custodians is best handled on another blockchain, i.e., where the coordinators have no influence on consensus.

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

– *Consensus-level Escrow (Smart Contracts)* are programs stored in a ledger which are executed and their result agreed upon by consensus participants [58, 61]. As such, trusting in the correct behavior of a smart contract is essentially trusting in the secure operation of the underlying chain, making this a useful construction for Escrows. Contrary to Consensus-level Custodians, who must actively follow the CCC protocol and potentially run additional software, with smart contracts consensus participants typically are not directly involved in the CCC protocol: an interaction with the CCC smart contract is, by default, treated like any other state transition and no additional software/action is required. CCC protocols which rely on smart contracts typically involve cross-chain state verification (cf. Appendix C) to enforce correct execution on both X and Y, and typically do not need additional collateralization, except as potential insurance against software bugs [23].

Model 2: Synchrony 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. In practice, so called *lock contracts* are typically used in CCC protocols that facilitate asset exchanges and implement two-phase commit, where the same (*symmetric*) locks are created on both chains and released atomically.

– *Hash Locks.* A protocol based on hash locks relies on the *preimage resistance* property of hash functions: participants P and Q transfer assets to each other by means of transactions that must be complemented with the preimage

of a hash $h := H(r)$ for a value r chosen by P – the initiator of the protocol – typically uniformly at random [20, 19, 93, 126].

– *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 [52, 50, 127, 158, 74, 143, 137].

– *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 [146] build upon inherently sequential functions where the result is only revealed after a predefined number of operations are performed. Verifiable delay functions [50] 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 [45].

Model 3: Hybrid (Watchtowers) Instead of fully relying on coordinators being available or synchrony assumptions among participants holding, it is possible to employ so called *watchtowers*, i.e., service providers which act as a fallback if CCC participants experience crash failures. We observe strong similarities with optimistic fair exchange protocols [35, 34, 62]. 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 [105, 38, 128, 37]. This construction was first introduced and applied to off-chain payment channels [89].

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. Thereby, it is also of relevance what exactly is being verified: the possibility or the consensus agreement on a state, a specific state transition (e.g. a transaction), or actual validation of a state under underlying consensus rules (we direct the interested reader to Appendix C for a detailed classification).

Model 1: Trusted Third Party (Coordinators). The simplest approach to cross-chain verification is to rely on a trusted third party (also referred to as *validators* [166]) to handle the verification of the state changes on interlinked chains during CCC execution.

– *External Validators.* A simple approach is to outsource the verification step to a (trusted) third party, external to the verifying ledger (in our case Y), as in [162, 18]. The TTP can then be the same as in the commit/abort steps.

– *Consensus Committee / Smart Contracts.* Alternatively, the verification can be handled by the consensus participants of the verifying chain [115, 69, 123], leveraging the assumption that misbehavior of consensus participants indicates a failure of the chain itself.

– *Verification Games.* Finally, rather than fully trusting coordinators, they can be used as a mere optimistic performance improvement by introducing dispute handling mechanisms to the verification process: users can provide (reactive) fraud proofs [29] or accuse coordinators of misbehavior requiring them to prove correct operation [159, 90, 103].

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

– *Direct Observation.* Similar to the commit phase of CCC, one can require all participants of a CCC protocol to execute the verification phase individually: i.e., to run (fully validating) nodes in all involved chains. This is often the case in exchange protocols, such as atomic swaps using symmetric locks such as HTLCs [20, 93], but also in parent-child settings where one chain by design verifies or validates the other [40, 86, 122]. This relies on a synchrony assumption, i.e., requires CCC participants to observe commitments and act within a certain time, in order to complete the CCC.

– *Smart Contracts (Chain Relays).* The verification process can be encoded in smart contracts capable of verifying the commitment on X, so called *chain relays*, as in the case of BTC-Relay [5] – a smart contract on Ethereum which tracks the Bitcoin main chain and verifies BTC payments. Recently, chain relays capable of verifying succinct proofs of knowledge [73, 165] have been proposed, which can (theoretically) enable full validation of commitments on X (i.e., similar properties as full nodes, see Appendix C).

Model 3: Hybrid (Watchtowers). Just like in the commit phase, synchrony and TTP assumptions can be combined in the verification phase, such that a CCC protocol initially relies on a synchrony assumption, but can fall back to a TTP (*watchtowers*, c.f Section 4.1) to ensure correct termination if messages are not delivered within a per-defined period.

4.3 Abort Phase

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

Model 1: Trusted Third Party (Coordinators) Similarly to the commit phase, an abort can be handled by a trusted third party and the possible implementations are the same as in Section 4.1. If a TTP was introduced in the commit phase, the abort phase will be typically handled by the exact same TTP.

Model 2: Synchrony Assumptions (Timelocks) Alternatively, it is possible to enforce synchrony by introducing timelocks, after the expiry of which the pro-

tocol 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. 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* [7] have elapsed since the *inclusion* of another transaction TX_1 in the underlying ledger. Typically, TX_1 and TX_2 are related as TX_2 spends assets transferred in TX_1 [144]. Although more practical than absolute timelocks (no need for external clock), we are not aware of schemes allowing the creation of relative timelocks *across* ledgers.

Model 3: Hybrid (Watchtowers) As an additional measure of security, TTPs can be introduced as a fallback to timelocks in case CCC participants experience crash failures, e.g. in form of a watchtower [105, 38, 128, 37] that recovers otherwise potentially lost assets. This is specifically useful in the case of atomic swaps using Hased Timelock Contracts (HTLCs) [20, 93, 3, 144], when either party crashes after the hashlock's secret has been revealed.

5 Classification of Existing CCC Protocols

We now apply the framework introduced in Section 4 to classify existing CCC protocols. In addition, we split existing proposals into two protocol families, based on their design rationale and use case: (i) *exchange* protocols, which synchronize the exchange of assets on two ledgers, and (ii) *asset migration* protocols, which allow to move an asset or object to a different ledger. We study the trust model each proposed protocol chooses at each step of the CCC process. Our findings are summarized in Table 1.

5.1 Exchange Protocols

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

(Pre-)Commit. For the commit phase of the exchange protocol, we observe a variety of different trust models applied in practice, ranging from single and restricted custodians of traditional exchanges (including such that use trusted hardware [45]) to protocols fully relying on synchrony and cryptographic hardness assumptions, such as HTLC atomic swaps [20, 93, 162]. HTLC swaps use hash locks to commit assets before executing cross-chain trades and are the longest-standing alternative to centralized solutions, having been first introduced in 2012 [19]. More recently, hash locks have been replaced with signature

Table 1: Classification of existing of Cross-Chain Communication protocols, in consideration of the selected TTP model (cf. Section 4) at each protocol step (commit, verify, abort). Notation for non-binary TTP values: ● uses a TTP, ○ fully relies on synchrony and availability of participants, ◑ hybrid. We also highlight if the TTP (committee) is static or changes dynamically, and whether collateral is utilzed to incentivize correct behavior of TTPs. We use the following abbreviations: **EC** for External Custodian, **CC** for Consensus Custodian, **EE** for External Escrow, **SC** for Smart Contract, **EV** for External Validator, **CM** for Consensus Committee, and **DO** for Direct Observation.

		Protocol	Trust Model at each CCC Protocol Phase							
			Commit on chain X				Verify & Commit on chain Y		Abort on chain X (optinal)	
			TTP	Dynamic?	Collateral?	Type	TTP	Type	TTP	Type
Exchange Protocols	(Atomic Swaps)	**Traditional Custodial Exchanges (e.g., [1, 45])**	●	✗	✗	EC (single, restricted)	●	EV	●	EC (single, restricted)
		A2L [158]	◑	✗	✓	EE (multisig + signature Lock)	○	DO	◑	EE + Timelock
		Arwen [92]	◑	✗	✗	EE (multisig + Hash Lock)	○	DO	◑	EE + Timelock
		Notarized HTLC Atomic Swaps [162]	○	-	-	Hash Lock	●	EV	◑	EE + Timelock
		HTLC Atomic Swaps [20, 93, 19, 162]	○	-	-	Hash Lock	○	DO	○	Timelock
		ECDSA/DLSAG Atomic Swaps [127, 137]	○	-	-	Signature Lock	○	DO	○	Timelock
		SPV Atomic Swaps [10, 111, 173, 94]	○	-	-	Standard payment	○	SC(chain relay)	-	-
Asset Migration Protocols	(Cryptocurrency-backed Assets)	**Bidirectional Chain Relays [111, 86]**	○	-	-	SC (chain relay)	○	SC (chain relay)	-	-
		XCLAIM [173], Dogethereum [160]	●	✓	✓	EC (single, unrestricted)	○	SC (chain relay)	-	-
		tBTC [17]	●	✗	✓	EC (committee, restricted)	○	SC (chain relay)	-	-
		RenVM [24]	●	✗	✓	EC (committee, restricted)	●	CM	-	-
		Custodial Wrapped Assets (e.g., [18])	●	✗	✗	EC (single, restricted)	●	EV	-	-
	Sharding	**ATOMIX [115]**	●	✗	✗	CC (shard X)	●	CM	●	CC (shard X)
		SBAC [28, 152]	●	✗	✗	CC (shard X)	●	CM	●	CC (shard X)
		Rapidchain [171]	●	✗	✗	CC (shard X)	●	CM	-	-
		Fabric Channels [30]	●	✗	✗	CC (shard X)	●	CM	●	CC (shard X)
	Side-chains	**Federated Sidechains/Pegs [40, 69, 86]**	●	✗	✗	EC (consensus of Y)	●	CM	-	-
		RSK [123, 122]	●	✗	✗	CC (consensus of X)	●	CM	-	-
	Boot-strapping	**Proof-of-Burn (Federated) [155, 104]**	○	-	-	SC / Burn address	●	CM	-	-
		Proof-of-Burn (SPV) [104]	○	-	-	SC / Burn address	○	SC (chain relay)	-	-
		Merged Mining/Staking [102, 86]	●	✗	✗	CC (consensus of X)	●	CM	-	-

locks in [127, 158, 74, 143, 137], improving privacy and cross-platform support. On blockchains which support (near) Turing complete programming languages (e.g., Ethereum [58]) the commitment can be handled via smart contracts – which can be easily extended to support collateralization to penalize misbehaving counterparties (e.g., to prevent griefing [173]). Finally, hybrid versions of HTLC and signature lock atomic swaps have been introduced, most notably Arwen [92] and A2L [158], where users commit assets into (HTLC) multisignature escrows with an exchange coordinator, improving usability and reducing online requirements (see abort). However, this requires a more complex setup process (similar to payment channels [144]) and necessitates pre-paid fees (Arwen) or collateral (A2L) to prevent malicious lockup of coordinator funds.

Verify. Contrary to traditional exchanges, where the custodial (operator) is also responsible for the verification phase, most atomic swap protocols (incl. HTLC, ECDSA/DSLAG, A2L and Arwen) require users to *directly observe* all chains involved in the CCC to check correctness of the protocol. This online requirement can be relinquished on chains with smart contract support by utilizing *chain relays* (cross-chain SPV clients) [10, 111, 173, 94]. In such SPV atomic swaps, the verification of the commitment phase and the correct finalization of the CCC protocol, is enforced by the consensus of the chain running the chain relay.

Abort. Protocols using direct observation for the verification phase (i.e., assuming synchrony over using a dedicated TTP), for example HTLC swaps, utilize timelocks to ensure an automatic abort of the CCC protocol, in case one of the two parties aborts or crashes. However, users of such protocols face strict online requirements: the initiator of a swap can steal funds of the receiving party, if the later does not claim the initiator's committed assets before the abort timelock expires (i.e., after the secret to the hashlock was released in the case of HTLCs). Protocols including A2L and Arwen outsource this to TTPs, at the cost of the risk of TTPs colluding with exchange parties to commit theft [162, 92, 158].

5.2 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 chain X, and creating a *representation* $y(x)$ on Y. Now, the state of x can only be updated by modifying $y(x)$, comparable to the concept of *mutual exclusion* in concurrency control [68]. 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. Thereby, our evaluation suggests a correlation between specific TTP implementations and CCC use cases. Concretely, we differentiate between *cryptocurrency-backed assets* (across independent blockchains, e.g. Bitcoin on Ethereum), *sharding*, *sidechains* and *bootstrapping* of new blockchains.

(Pre-)Commit. We observe protocols creating *cryptocurrency-backed assets* (e.g. "wrapped" Bitcoin on Ethereum) generally rely on TTPs in the form of External Custodians. In the simplest case, custodians are be single, centralized providers (e.g. wBTC [18]). More resilient approaches such as tBTC [17] rely

on some pre-defined committee (holders of some cryptocurrency token), which can also be collateralized. A dynamic approach, on the other hand, is followed by XCLAIM [173] and Dogethereum [160]: here, *anyone* can become a coordinator by putting down collateral to secure locked assets. Collateral in this setting is introduced to protect users against financial losses in case of CCC failure. When communicating between two chains with smart contract support, TTPs can be replaced by a synchrony assumption. Such CCC protocols implement (bidirectional) chain relay smart contracts on both involved chains X and Y: CCC participants prove their commitment on chain X to a chain relay on Y (and vice-versa). As long as the proof is delivered, the CCC protocol will succeed.

While similar conceptually, CCC protocols in *sharding* rely on the consensus participants of the "source" shard X to correctly execute the commitment. Since all shards by design have the same security model, it is acceptable for chain Y to explicitly trust consensus of X for correctness cross-shard transfers.

Sidechains [40, 69, 86], which extend existing blockchains with new features and create a parent-child dependency between involved chains in terms of security, typically make the consensus participants of the "child" chain Y responsible for the commitment on X. For example, the consensus committee ("federation") of the Liquid sidechain operates a multisig lock Bitcoin [69]. RSK [123, 122] poses an exception thereof, as it assumes that the parent X and child Y chain share consensus: Bitcoin miners merge-mine [102] the RSK sidehcain.

Finally, Proof-of-Burn [155, 104] protocols avoid the need for explicit TTPs, implementing the commit phase using a smart contract or "burn" address which ensure that locked assets can never be recovered.

Verify. Asset migration protocols - with the exception of centralized, custodial services - rely the consensus of chain Y to correctly verify the commitment on X. This can be implemented (i) under synchrony assumptions by using *chain relay* smart contracts, which cryptographically verify the correctness of the commitment on X, or (ii) by requesting the consensus committee of Y to explicitly sign off on the CCC execution.

Abort. We observe that asset migration protocols generally do not implement an explicit abort phase. Instead, they assume that if the commitment on X is executed correctly, then it will eventually be verified by chain Y, which in turn will commit. An exception are cross-shard transfer protocols ATOMIX [115], SBAC [28], and (Fabric) Channels [30] which implement two-phase commit and allow the consensus of shard X to manually abort a cross-shard transfer.

6 CCC Outlook and Implications

We provide an outlook on the challenges faced by CCC protocols and the implications of interoperability in terms of security and privacy for blockchains.

6.1 Outlook and Challenges for CCC Protocol Designs

Synchrony Across Chains. The absence of a global clock across chains requires to either agree and trust a third party as a clock, or rely on chain-

dependent time definition, such as block generation rate [82], hindering a seamless synchronization across chains [82, 173]. Many factors, such as 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. Failing to take precautions by setting conservative lower bounds on communication delays, or falsely relying on UNIX timestamps can result in severe protocol flaws due to race conditions.

Data Availability. Protocols employing cross-chain verification via chain relays typically rely on timely arrival of proofs and metadata (block headers, transactions, ...). However, if an adversary can exclude this data from the chain, these protocols not only become less efficient but potentially vulnerable [29]. This is a particular problem in settings where data availability is not enforced by consensus, e.g. if CCC protocols rely on additional data being included in blockchains in the form of velvet forks [174, 108]. First attempts to mitigate this problem have been suggested in [29, 27] – yet at the cost of higher protocol complexity and potential communication overhead.

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 [97] as its digital signature scheme and a system Y using Schnorr [149] is only possible if both schemes are instantiated over the same elliptic curve [127]. 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* [99].

Collateralization and Exchange Rates. In recent works [173, 160, 111, 17], there is a trend towards collateralizing coordinators to prevent financial damage to users and incentive correct behavior of TTPs - potentially achieving *economically trustless* CCC protocols (cf. Section 3.3). Thereby, it is crucial to ensure that the provided collateral has sufficient value to outweigh potential gains from misbehavior. However, in the cross-chain setting, where insured asset and collateral are typically different, collateralized CCC protocols are forced to (i) implement measures against exchange rate fluctuations such as over-collateralization which incur capital inefficiencies for participants, and (ii) rely on price oracles. While the latter are an active field of research [145, 26, 161, 6, 175], price oracles so far remain a single point of failure of collateralized CCC protocols.

Industrial Trends. 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 [166, 119, 153, 163, 96, 21]. Individual shards, which are coordinated via a parent chain running a BFT agreement protocol, connect to and import assets from existing blockchains like Bitcoin [14]. These projects have in common that they rely on existing techniques such as cryptocurrency-backed assets [173, 160, 86] to bridge the gap to existing systems (hence not included in the classification). A formal treatment of this design, also considering distributed computations, is presented in [124].

6.2 Implications for Interoperable Blockchains

Security and Adversary Model. Interconnected chains X and Y can have a well defined security model on their own, and it may 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 $\alpha \leq 33\%$ [77, 87, 148]. 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 [85, 78] may be lower than that of accumulating computational power, or vice-versa [53]. Since permissionless ledgers (such as PoW or PoS) are not Sybil resistant [70], i.e., provide weak identities at best, quantifying adversary strength is nearly impossible, even within a single ledger [36]. This task becomes even more difficult in the cross-chain setting: not only can consensus participants (i) "hop" between different chains [133, 120], destabilizing involved systems, but also (ii) be susceptible to bribing attacks, executed cross-chain, where detection and countermeasures are close to impossible [130, 101].

Consensus Finality Guarantees. Interlinked chains X and Y may assume different finality [154] 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 [82]; instead, Y deems transactions valid as soon as they are written to the ledger ($k = 1$, e.g. [25]). 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 - potentially resulting in an inconsistent state on Y and financial damage to users.

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 [129, 152, 60]. Special consideration may be needed in case of permanent blockchain forks, as this may require updating the way verification is performed [129, 173].

Composability Attacks. We recall, in blockchains with *stablizing* [164, 32] consensus, a security parameter k is used to denote the number of blocks or *confirmations* [7] a transaction should have, before being accepted as secure [82, 141, 138], i.e., with the probability of a reversion being negligible. In addition, the value of a verified transaction should be considered: the higher the potential gain, the higher the risk of an attack, and the more confirmations should be required [151]. However, following recent works [176, 101], we argue that it may be insufficient to consider the *composition* of a block within a single blockchain: interactions of financial applications across different blockchains make the potential extracted value from attacks unpredictable and may require even more conservative bounds on transaction stabilization.

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 ad-

vantages, this improves the *fungibility* of payments. However, there exist several privacy attack vectors in cross-chain communication: (i) recent works [126, 88] show attacks leveraging the locking mechanism and some countermeasures have been proposed [126, 127, 147]; (ii) heuristics to explore blocks from different cryptocurrencies [102] as well as forks [156] to cluster miner and user accounts [95]; (iii) CCC protocols leveraging coordinators, similar to and payment hubs [89, 100], also lead to privacy leakages that enable further account clustering [169]. Recent works [88, 91, 158] propose measures that allow to preserve the anonymity of participants, if added to CCC protocols.

7 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. Following this result, we introduce a framework for evaluating existing and designing new cross-chain communication protocols, based on the inherent trust assumptions. We then provide a classification and comparative evaluation, taking into account both academic research and the vast number of online resources, allowing us to better understand the similarities and differences between existing cross-chain communication approaches. Finally, by discussing implications and open challenges faced by cross-chain communication protocols, as well as the implications of interoperability on the security and privacy of blockchains, we offer a comprehensive guide for designing protocols, bridging multiple distributed ledgers.

References

1. Binance exchange. Online. https://www.binance.com/en, Accessed: 2020-09-19.
2. Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv. Accessed: 2018-05-16.
3. Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts. Accessed: 2018-05-16.
4. Bitcoin wiki: Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification . Accessed: 2018-05-03.
5. Btcrelay. https://github.com/ethereum/btcrelay. Accessed 2019-08-15.
6. Chainlink: A decentralized oracle network. Online. https://link.smartcontract.com/whitepaper , Accessed: 2020-09-19.
7. Confirmations. https://en.bitcoin.it/wiki/Confirmation. Accessed: 2018-11-28.
8. Dogerelay. https://github.com/dogethereum/dogerelay. Accessed 2019-08-15.
9. Eth-eos-relay. https://github.com/EveripediaNetwork/eth-eos-relay . Accessed 2019-08-15.

10. Ethereum contract allowing ether to be obtained with bitcoin. https://github.com/ethers/EthereumBitcoinSwap. Accessed: 2018-10-30.
11. Parity-Bridge. https://github.com/paritytech/parity-bridge. Accessed 2019-08-15.
12. The parity light protocol - wiki. https://wiki.parity.io/The-Parity-Light-Protocol-(PIP). Accessed: 2018-10-30.
13. Peace relay. https://github.com/loiluu/peacerelay. Accessed 2019-08-15.
14. Polkabtc: Trustless bitcoin on polkadot. Online. https://github.com/interlay/BTC-Parachain , Accessed: 2020-09-19.
15. Project alchemy. https://github.com/ConsenSys/Project-Alchemy. Accessed 2019-08-15.
16. Project waterloo. https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum-1c230ac65524. Accessed 2019-08-15.
17. tbtc: A decentralized redeemable btc-backed erc-20 token. http://docs.keep.network/tbtc/index.pdf . Accessed: 2019-11-15.
18. Wrapped bitcoin. https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf . Accessed: 2018-05-03.
19. Alt chains and atomic transfers. bitcointalk.org, 2013.
20. Atomic swap. Bitcoin Wiki, 2013.
21. Wanchain whitepaper. https://www.wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf, 2017.
22. Bitcoin supply on ethereum tops $1b. Coindesk, September 2020. https://www.coindesk.com/bitcoin-supply-on-ethereum-tops-1b .
23. Nexus mutual: A peer-to-peer discretionary mutual on the ethereum blockchain. Online, 2020. https://nexusmutual.io/assets/docs/nmx_white_paperv2_3.pdf , Accessed: 2020-09-19.
24. Renvm. Online, 2020. https://renproject.io/renvm , Accessed: 2020-09-19.
25. I. Abraham, G. Gueta, and D. Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. arXiv:1803.05069, 2018.
26. J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania. Astraea: A decentralized blockchain oracle. *arXiv preprint arXiv:1808.00528*, 2018.
27. M. Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts, 2019.
28. M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis. Chainspace: A sharded smart contracts platform. In *2018 Network and Distributed System Security Symposium (NDSS)*, 2018.
29. M. Al-Bassam, A. Sonnino, and V. Buterin. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *CoRR*, abs/1809.09044, 2018.
30. E. Androulaki, C. Cachin, A. De Caro, and E. Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In *European Symposium on Research in Computer Security*, pages 111–131. Springer, 2018.
31. M. Andrychowicz. Multiparty computation protocols based on cryptocurrencies, 2015. Accessed: 2017-02-15.
32. D. Angluin, M. J. Fischer, and H. Jiang. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems*, pages 37–50. Springer, 2006.
33. N. Asokan. Fairness in electronic commerce. 1998.
34. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, pages 86–99. IEEE, 1998.

35. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 591–606. Springer, 1998.
36. G. Avarikioti, L. Käppeli, Y. Wang, and R. Wattenhofer. Bitcoin security under temporary dishonest majority. In *23rd Financial Cryptography and Data Security (FC)*, 2019.
37. G. Avarikioti, E. K. Kogias, and R. Wattenhofer. Brick: Asynchronous state channels. *arXiv preprint arXiv:1905.11360*, 2019.
38. G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer. Towards secure and efficient payment channels. *arXiv preprint arXiv:1811.12740*, 2018.
39. O. Babaoglu and S. Toueg. Understanding non-blocking atomic commitment. *Distributed systems*, pages 147–168, 1993.
40. A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling blockchain innovations with pegged sidechains, 2014.
41. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.
42. E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2016.
43. J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.
44. P. Bennink, L. v. Gijtenbeek, O. v. Deventer, and M. Everts. An analysis of atomic swaps on and between ethereum blockchains using smart contracts. Tech. report, 2018. https://work.delaat.net/rp/2017-2018/p42/report.pdf.
45. I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. Cryptology ePrint Archive, Report 2017/1153, 2017. Accessed:2017-12-04.
46. I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology–CRYPTO 2014*, pages 421–439. Springer, 2014.
47. I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake, 2016. Accessed: 2016-11-08.
48. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*, volume 370. Addison-wesley New York, 1987.
49. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.
50. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO*, 2018.
51. D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188, 2018. https://eprint.iacr.org/2018/1188.
52. D. Boneh and M. Naor. Timed commitments. In *Annual International Cryptology Conference*, pages 236–254. Springer, 2000.
53. J. Bonneau. Why buy when you can rent? bribery attacks on bitcoin consensus. In *BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research*, February 2016.
54. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source, 2015. Accessed: 2015-10-25.

55. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.
56. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Efficient range proofs for confidential transactions, 2017. Accessed:2017-11-10.
57. B. Bünz, S. Goldfeder, and J. Bonneau. Proofs-of-delay and randomness beacons in ethereum. 2017.
58. V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014. Accessed: 2016-08-22.
59. V. Buterin. Chain interoperability. Tech. report, 2016. Accessed: 2017-03-25.
60. V. Buterin. Cross-shard contract yanking. https://ethresear.ch/t/cross-shard-contract-yanking/1450, 2018.
61. C. Cachin. Architecture of the hyperledger blockchain fabric, 2016. Accessed: 2016-08-10.
62. C. Cachin and J. Camenisch. Optimistic fair secure computation. In *Annual International Cryptology Conference*, pages 93–111. Springer, 2000.
63. M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
64. D. Catalano and D. Fiore. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*, pages 55–72. Springer, 2013.
65. A. Chepurnoy, T. Duong, L. Fan, and H.-S. Zhou. Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake, 2017. Accessed: 2017-03-22.
66. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.
67. C. Decker and R. Wattenhofer. Bitcoin transaction malleability and mtgox. In *Computer Security-ESORICS 2014*, pages 313–326. Springer, 2014.
68. E. W. Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.
69. J. Dilley, A. Poelstra, J. Wilkins, M. Piekarska, B. Gorlick, and M. Friedenbach. Strong federations: An interoperable blockchain solution to centralized third party risks. *arXiv preprint arXiv:1612.05491*, 2016.
70. J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
71. T. Duong, L. Fan, and H.-S. Zhou. 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716, 2016. Accessed: 2017-02-06.
72. S. Dziembowski, L. Eckey, and S. Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984. ACM, 2018.
73. J. Eberhardt and S. Tai. Zokrates-scalable privacy-preserving off-chain computations.
74. C. Egger, P. Moreno-Sanchez, and M. Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *CCS*, 2019.
75. S. Even. A protocol for signing contracts. Technical report, Computer Science Department, Technion. Presented at CRYPTO'81, 1982.
76. S. Even and Y. Yacobi. Relations among public key signature systems. Technical report, Computer Science Department, Technion, 1980.

77. I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.
78. G. Fanti, L. Kogan, S. Oh, K. Ruan, P. Viswanath, and G. Wang. Compounding of wealth in proof-of-stake cryptocurrencies. *arXiv preprint arXiv:1809.07468*, 2018.
79. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. volume 32, pages 374–382. ACM, 1985.
80. B. Ford, L. Gasser, E. K. Kogias, and P. Jovanovic. Cryptographically verifiable data structure having multi-hop forward and backwards links and associated systems and methods, Dec. 13 2018. US Patent App. 15/618,653.
81. A. Gabizon, K. Gurkan, P. Jovanovic, G. Konstantopoulos, A. Oines, M. Olszewski, M. Straka, and E. Tromer. Plumo: Towards scalable interoperable blockchains using ultra light validation systems. 2020.
82. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty, 2016. Accessed: 2017-02-06.
83. A. Garoffolo, D. Kaidalov, and R. Oliynykov. Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. *arXiv preprint arXiv:2002.01847*, 2020.
84. F. C. Gärtner. Specifications for fault tolerance: A comedy of failures. 1998.
85. P. Gaži, A. Kiayias, and A. Russell. Stake-bleeding attacks on proof-of-stake blockchains. Cryptology ePrint Archive, Report 2018/248, 2018. Accessed:2018-03-12.
86. P. Gazi, A. Kiayias, and D. Zindros. Proof-of-stake sidechains. *IEEE Security and Privacy. IEEE*, 2019.
87. A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdo rf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC*, pages 3–16. ACM, 2016.
88. M. Green and I. Miers. Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701, 2016. Accessed: 2017-08-07.
89. L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Off the chain transactions. Cryptology ePrint Archive, Report 2019/360, 2019. https://eprint.iacr.org/2019/360.
90. D. Harz and M. Boman. The scalability of trustless trust. arXiv:1801.09535, 2018. Accessed:2018-01-31.
91. E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub, 2016. Accessed: 2017-09-29.
92. E. Heilman, S. Lipmann, and S. Goldberg. The arwen trading protocols. Whitepaper. https://www.arwen.io/whitepaper.pdf.
93. M. Herlihy. Atomic cross-chain swaps. arXiv:1801.09515, 2018. Accessed:2018-01-31.
94. M. Herlihy, B. Liskov, and L. Shrira. Cross-chain deals and adversarial commerce. *arXiv preprint arXiv:1905.09743*, 2019.
95. A. Hinteregger and B. Haslhofer. An empirical analysis of monero cross-chain traceability. *arXiv preprint arXiv:1812.02808*, 2018.
96. D. Hosp, T. Hoenisch, P. Kittiwongsunthorn, et al. Comit-cryptographically-secure off-chain multi-asset instant transaction network. *arXiv preprint arXiv:1810.02174*, 2018.
97. D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

98. S. Johnson, P. Robinson, and J. Brainard. Sidechains and interoperability. *arXiv preprint arXiv:1903.04077*, 2019.
99. J. Jones and abitmore. Optional htlc preimage length and hash160 addition. BSIP 64, blog post. https://github.com/bitshares/bsips/issues/163.
100. M. Jourenko, K. Kurazumi, M. Larangeira, and K. Tanaka. Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Report 2019/352, 2019. https://eprint.iacr.org/2019/352.
101. A. Judmayer, N. Stifter, A. Zamyatin, I. Tsabary, I. Eyal, P. Gaži, S. Meiklejohn, and E. Weippl. Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. Cryptology ePrint Archive, Report 2019/775, 2019. https://eprint.iacr.org/2019/775.
102. A. Judmayer, A. Zamyatin, N. Stifter, A. G. Voyiatzis, and E. Weippl. Merged mining: Curse or cure? In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*, Sep 2017.
103. H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*, pages 1353–1370. USENIX Association, 2018.
104. K. Karantias, A. Kiayias, and D. Zindros. Proof-of-burn. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer, 2020.
105. M. Khabbazian, T. Nadahalli, and R. Wattenhofer. Outpost: A responsive lightweight watchtower. 2019.
106. A. Kiayias, N. Lamprou, and A.-P. Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, Springer, 2016.
107. A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963, 2017. Accessed:2017-10-03.
108. A. Kiayias, A. Polydouri, and D. Zindros. The Velvet Path to Superlight Blockchain Clients, 2020.
109. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.
110. A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.
111. A. Kiayias and D. Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.
112. E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, Aug. 2016. USENIX Association.
113. E. Kokoris-Kogias. Robust and scalable consensus for sharded distributed ledgers. Technical report, Cryptology ePrint Archive, Report 2019/676, 2019.
114. E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gailly, L. Gasser, P. Jovanovic, E. Syta, and B. Ford. Calypso: Auditable sharing of private data over blockchains. Technical report, Cryptology ePrint Archive, Report 2018/209, 2018.
115. E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

116. L. Kokoris-Kogias, L. Gasser, I. Khoffi, P. Jovanovic, N. Gailly, and B. Ford. Managing identities using blockchains and CoSi. In *9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016)*, 2016.
117. R. Kumaresan and I. Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 418–429. ACM, 2016.
118. A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50–63, 2012.
119. J. Kwon and E. Buchman. Cosmos: A network of distributed ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, 2015.
120. Y. Kwon, H. Kim, J. Shin, and Y. Kim. Bitcoin vs. bitcoin cash: Coexistence or downfall of bitcoin cash? arXiv:1902.11064, 2019.
121. L. Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, 1989.
122. S. Lerner. Drivechains, sidechains and hybrid 2-way peg designs. Technical report, Tech. Rep. [Online], 2018.
123. S. D. Lerner. Rootstock: Bitcoin powered smart contracts. https://docs.rsk.co/RSK_White_Paper-Overview.pdf, 2015.
124. Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, and Y.-C. Hu. Hyperservice: Interoperability and programmability across heterogeneous blockchains. *arXiv preprint arXiv:1908.09343*, 2019.
125. L. Luu, B. Buenz, and M. Zamani. Flyclient super light client for cryptocurrencies. Accessed 2018-04-17.
126. G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi. Concurrency and privacy with payment-channel networks. In *CCS*, pages 455–471, 2017.
127. G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*, 2019.
128. P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.
129. P. McCorry, E. Heilman, and A. Miller. Atomically trading with roger: Gambling on the success of a hardfork. In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*, Sep 2017.
130. P. McCorry, A. Hicks, and S. Meiklejohn. Smart contracts for bribing miners. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.
131. I. Meckler and E. Shapiro. Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf, 2018.
132. R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
133. D. Meshkov, A. Chepurnoy, and M. Jansen. Revisiting difficulty control for blockchain systems. Cryptology ePrint Archive, Report 2017/731, 2017. Accessed: 2017-08-03.
134. S. Micali. Algorand: The efficient and democratic ledger, 2016. Accessed: 2017-02-09.
135. A. Miller. The high-value-hash highway, bitcoin forum post, 2012.
136. M. Miraz and D. C. Donald. Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. *Annals of Emerging Technologies in Computing (AETiC) Vol*, 3, 2019.

137. P. Moreno-Sanchez, Randomrun, D. V. Le, S. Noether, B. Goodell, and A. Kate. Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. Cryptology ePrint Archive, Report 2019/595, 2019. https://eprint.iacr.org/2019/595.
138. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008. Accessed: 2015-07-01.
139. K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford. CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds.
140. H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology . . . , 1999.
141. R. Pass, L. Seeman, and a. shelat. Analysis of the blockchain protocol in asynchronous networks, 2016. Accessed: 2016-08-01.
142. R. Pass and E. Shi. Hybrid consensus: Scalable permissionless consensus, Sep 2016. Accessed: 2016-10-17.
143. A. Poelstra. Scriptless scripts. Presentation slides. https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf.
144. J. Poon and T. Dryja. The bitcoin lightning network, 2016. Accessed: 2016-07-07.
145. H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, et al. Tls-n: Non-repudiation over tls enabling ubiquitous content signing. In *Network and Distributed System Security Symposium (NDSS)*, 2018.
146. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
147. J. Rubin, M. Naik, and N. Subramanian. Merkelized abstract syntax trees. http://www.mit.edu/ jlrubin/public/pdfs/858report.pdf, 2014.
148. A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin, 2015. Accessed: 2016-08-22.
149. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.
150. V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos. Interledger smart contracts for decentralized authorization to constrained things, 2019.
151. Y. Sompolinsky and A. Zohar. Bitcoin's security model revisited, 2016. Accessed: 2016-07-04.
152. A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis. Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. *arXiv preprint arXiv:1901.11218*, 2019.
153. M. Spoke and Nuco Engineering Team. Aion: The third-generation blockchain network. https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf. Accessed 2018-04-17.
154. A. Stewart and E. Kokoris-Kogia. Grandpa: a byzantine finality gadget. *arXiv preprint arXiv:2007.01560*, 2020.
155. I. Stewart. Proof of burn, 2012. Accessed: 2017-05-10.
156. N. Stifter, P. Schindler, A. Judmayer, A. Zamyatin, A. Kern, and E. Weippl. Echoes of the past: Recovering blockchain metrics from merged mining. In *Proceedings of the 23nd International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2019.
157. P. Syverson. Weakly secret bit commitment: Applications to lotteries and fair exchange. In *Proceedings. 11th IEEE Computer Security Foundations Workshop (Cat. No. 98TB100238)*, pages 2–13. IEEE, 1998.

158. E. Tairi, P. Moreno-Sanchez, and M. Maffei. A^{2}l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. Cryptology ePrint Archive, Report 2019/589, 2019. https://eprint.iacr.org/2019/589.
159. J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains, March 2017. Accessed:2017-10-06.
160. J. Teutsch, M. Straka, and D. Boneh. Retrofitting a two-way peg between blockchains. Technical report, 2018.
161. J. Teutsch and TrueBit Establishment. On decentralized oracles for data availability. 2017.
162. S. Thomas and E. Schwartz. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf*, 2015.
163. G. Verdian, P. Tasca, C. Paterson, and G. Mondelli. Quant overledger whitepaper. https://www.quant.network/, 2018.
164. M. Vukolic. Eventually returning to strong consistency, 2016. Accessed: 2016-08-10.
165. M. Westerkamp and J. Eberhardt. zkrelay: Facilitating sidechains using zksnark-based chain-relays. *Contract*, 1(2):3, 2020.
166. G. Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2015.
167. G. Wood. Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07), 2017. Accessed: 2018-01-03.
168. A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.
169. H. Yousaf, G. Kappos, and S. Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 837–850, 2019.
170. M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. *arXiv preprint arXiv:1910.01247*, 2019.
171. M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: A fast blockchain protocol via full sharding. Cryptology ePrint Archive, Report 2018/460, 2018.
172. A. Zamyatin, Z. Avarikioti, D. Perez, and W. J. Knottenbelt. Txchain: Efficient cryptocurrency light clients via contingent transaction aggregation. Sep 2020.
173. A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. *IEEE Security and Privacy. IEEE*, 2019.
174. A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottebelt. (Short Paper) A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.
175. F. Zhang, S. K. D. Maram, H. Malvai, S. Goldfeder, and A. Juels. Deco: Liberating web data using decentralized oracles for tls. *arXiv preprint arXiv:1909.00938*, 2019.
176. D. Zindros. Summa proofs are not composable, 2019.

A Strong Fair Exchange Definition

This section provides the definition of the strong Fair Exchange problem, as presented in [33–35, 140].

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) [84]. A successful Fair Exchange is shown in Table 2 below.

Table 2: A successful Fair Exchange, as defined in [33–35, 140].

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 [34, 140].

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 3.2).

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 $m_P = \mathsf{false}$ **then**
  commit(TX_P, L_x); // P transfers a_P to Q
end
if (*verify*(TX_P, L_x, d_P) $= \mathsf{true}$) $\wedge$ $m_Q = \mathsf{false}$ **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) $= \mathsf{false}$ **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 $\text{TX} \in \mathsf{L} \wedge \mathit{desc}(\text{TX}) = d$ **then**
  return true;
end
return false;

Algorithm 4: Abort(TX, L)

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

C Classification of Cross-Chain State Verification

As described in Section 2.5, a critical component of cross-chain communication is the verification of the state "evolution" of a chain X from within another chain Y, i.e., that X is in a certain state after the commit step. In this section we discuss the different elements of the chain that can be verified during the process, to complement the process of verifying state evolution. We show that there is a classification for what is verified (Section C.1), overview existing techniques for each class, and discuss the relation between the verification classes (Section C.2).

C.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[4, 102]: 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 [55, 57, 71, 65].

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 [138, 2, 12]: 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 [138, 125]. 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 [64] (or some other form of cryptographic accumulator [43]), e.g. Merkle trees[132]. 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* [106, 107], also referred to as SPV (simplified payment verification) [138] 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 [2, 173]. The first known implementation of cross-chain state agreement verification (for PoW blockchains) is BTCRelay [5]: a smart contract which allows to verify the state of Bitcoin on Ethereum[3].
- **Proof-of-Stake.** If the verified chain uses Proof-of-Stake in its consensus, the proofs represent a dynamic collection of signatures, capturing the underlying stake present in the chain. These are referred to as *Proofs of Proof-of-Stake* (PoPoS) and a scheme in this direction was put forth in [86].
- **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 [63]. If the committee membership is dynamically changing, the verification process needs to capture the rotating configuration of the committee [139].

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 [135, 40, 106, 107] and FlyClient [125]. Both techniques rely on probabilistic sampling but differ in the selection heuristic. Superblock (Ni)PoPoWs sample blocks which exceed the required PoW difficulty[4], i.e., randomness is sourced from the unpredictability of the mining process, whereas FlyClient suggests to sample blocks using an optimized heuristic after the chain has been generated (using randomness from the PoW hashes [55]). For blockchains maintained by a static BFT committee, the verified signatures can be combined into aggregate signatures [112, 113] 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 [80, 139, 116], 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 [83, 81, 165].

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.

[3] Similar contracts have been proposed for other chains[15, 13, 8, 16, 11, 9].

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

As such, to verify that $\text{TX} \in \mathsf{L}_x[r]$ 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 [172].

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 [49], STARKs [41] or Bulletproofs [56]. First schemes for blockchains offering such proofs for each state transition are put forth in [131, 51, 42]. 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" [29] 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 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 [29, 170], 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.

C.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.

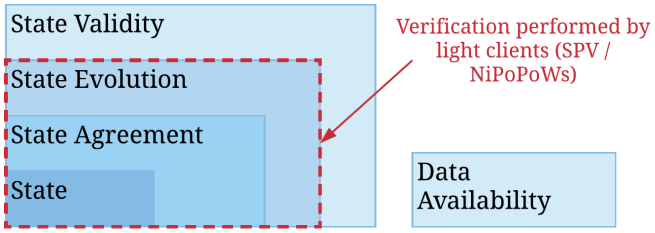

Fig. 2: Venn diagram of cross-chain state verification classes. The red, dotted line highlights the minimum requirement for correctly operating light clients, i.e., verifying SPV / NIPoPoWs in the case of PoW blockchains.

D 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 conducted the early stages of this work during his time at SBA Research, and was supported by a Binance Research Fellowship.