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], Aggelos Kiayias[7,8], and William J. Knottenbelt[1]

[1] Imperial College London
[2] SBA Research
[3] University College London
[4] University of Athens
[5] Ecole polytechnique fédérale de Lausanne
[6] TU Wien
[7] IOHK
[8] University of Edinburgh

Abstract. Communication across distributed systems, where each system runs its own consensus, is a problem previously studied only within a single trust domain (e.g., a datacenter). With the appearance of distributed ledgers or *blockchains*, numerous protocols requiring robustness against *adversarial* behavior have emerged. Cross-chain communication thereby plays a fundamental role in cryptocurrency exchanges, sharding, as well as the bootstrapping and migration of distributed ledgers. Unfortunately, existing proposals are designed ad-hoc for specific use-cases, making it hard to gain confidence on their correctness and to use them as building blocks for new systems.
In this paper, we provide the first systematic exposition of protocols for cross-chain communication. Through formalization of the underlying research question, which reduces to the fair exchange problem, we identify threat and network model assumptions, necessary for designing correct cross-chain communication protocols. We overview main applications, derive a classification and provide a comparative analysis of existing approaches. Further, we survey and classify techniques for verifying state cross-chain and mechanisms for constructing locks.

1 Introduction

Why Cross-Chain Communication is Worthy of Research. Since the introduction of Bitcoin [141] as the first decentralized ledger currency in 2008, the topic of blockchains (or distributed ledgers) has evolved into a well studied field in both industry and academia. Nevertheless, developments are still largely driven by community effort, resulting in a plethora of blockchain-based digital currencies being created. Taking into account the heterogeneous nature of these systems in terms of design and purpose, it is unlikely that there shall emerge a "coin to rule them all", yielding interoperability an important research problem. Thereby, cross-chain communication is not only found in currency transfers and exchanges [18, 19, 94–96], but is a critical component of scalabilty solutions (synchronization in sharded systems [22, 23, 26, 117, 177]), feature extensions (sidechains [36, 84, 112, 125] and cryptocurrency-backed assets [168, 179]), as well as bootstrapping of new and migration between existing systems [3, 51, 105, 162]. In addition, numerous competing projects aiming to create a single platform for cross-chain communication have recently emerged [20, 98, 121, 161, 169, 170, 172]. However, in spite of the vast number of use cases and solution attempts, the underlying problem of cross-chain communication has neither been clearly defined, nor have the associated challenges been studied or related to existing research.
Historical Background and Difference to Databases. The need for communication among distributed processes is fundamental to any distributed computing algorithm. In databases, to ensure the atomicity of a distributed transaction, an agreement problem must be solved among the set of participating processes. Referred to as the Atomic Commit problem (AC) [44], it requires the processes to agree on a common outcome for the transaction: commit or abort. If there is a strong requirement that every correct process should eventually reach an outcome despite the failure of other processes, the problem is called Non-Blocking Atomic Commit (NB-AC) [35]. Solving this problem enables correct processes to relinquish locks without

waiting for crashed processes to recover. As such, we can relate the core ideas of communication across distributed ledgers to NB-AC. The key difference hereby lies within the security model of the interconnected systems. While in classic distributed databases all processes are expected to *adhere to protocol rules* and, in the worst case, may crash, distributed ledgers must also consider and handle *Byzantine failures*.

Contributions. In this work, we provide the first systematic analysis of communication across distributed ledgers. We start with an overview of the main scenarios for cross-chain communication observed in practice (Section 3). We then formalize the underlying problem of Correct Cross-Chain Communication (CCC) and show a reduction to the Fair Exchange problem, known to be *impossible without a trusted third party* (Section 4). Consequently, we provide a classification of CCC protocols (Section 5), and systematically analyze mechanisms for cross-chain state verification (Section 6) and locking techniques necessary for atomicity (Section 7). Finally, we discuss implications for the blockchain, network, and threat models, and discuss implications for privacy (Section 8). We overview the related work in Section 9 and conclude this paper in Section 10.

2 Preliminaries

In literature, the terms blockchain and *distributed ledger* are often used as synonyms. We adopt this nomenclature as we proceed to introduce some notation, on the basis of [84] with minor alterations. For background reading on the basics of distributed ledgers, we refer the reader to [54, 81, 141].

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

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

Global Clock. The state evolution of two distinct ledgers L_x and L_y may progress at different *time* intervals: In the time that L_x progresses *one* s_x slot, L_y may, for example, progress *fourty* s_y slots (on average, as in the case of Bitcoin [141] and Ethereum [58]). To capture the order of transactions when synchronizing across L_x and L_y, we introduce a clock function τ which maps a given slot on any ledger to the time on a global, synchronized clock $\tau : s \to t$. For simplicity and readability, we use this conversion implicitly, i.e., given $\tau(i) = t$ we write $\mathsf{L}[t] = \mathsf{L}[i]$ instead of $\mathsf{L}[t] = \mathsf{L}[\tau(i)]$.

Persistence and Liveness. Each participant P adopts and maintains a local ledger state $\mathsf{L}^P[t]$ at time t ($\mathsf{L}_x^P[t]$ and $\mathsf{L}_y^P[t]$ respectively), i.e., her current view of the ledger. The views of two distinct participants P and Q on the same ledger L may differ at time t (e.g., due to network delay or message loss): $\mathsf{L}^P[t] = \mathsf{L}[i]$, $\mathsf{L}^Q[t] = \mathsf{L}[i-1]$, $\mathsf{L}^P[t] \neq \mathsf{L}^Q[t]$. However, eventually, all honest parties in the ledger will have the same view. This is captured by the persistence and liveness properties of distributed ledgers [81]:

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 time $t \in \mathbb{N}$, then TX will appear in its ledger at time t', i.e., $\exists t' \in \mathbb{N} : t' \geq t \wedge \mathrm{TX} \in \mathsf{L}^P[t']$. Additionally, we require the interval $t' - t$ to be *upper bound* by u.

[1] In practice often referred to as the blockchain *height*.

Persistence: Consider two honest parties P, Q of a ledger L and a *persistence delay parameter* k. 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 $t' > t$. Concretely, for all honest parties P and Q, we have that $\forall t \in \mathbb{N} : \exists t' \geq t : \mathsf{L}^Q[t'] = \mathsf{L}^P[t]$. Additionally, we require that the interval $t' - t$ is *upper bound* by k.
Note: when a participant P writes a transaction TX_P to L, any participant Q can verify TX_P was written by P.
Assets/Objects. We denote assets/objects on ledgers L_x and L_y as x and y, respectively.
Full nodes and Light Clients. We differentiate between two types of nodes in the networks: (i) those that store the entire transaction history and validate the full ledger state (*full nodes*), (ii) and those that only store transactions relevant to them and verify their inclusion in the underlying ledger, but do not check their validity (*light clients*).

3 Main Scenarios for Cross-Chain Communication

We proceed to give an overview of the main scenarios for cross-chain communication, as observed in practice.
Transfers and Exchanges Across Ledgers. The likely most well known use case for communication across distributed ledgers is the transfer of asset ownership between users. In practice, this implies two parties (i) agree on exchange terms, e.g. amount, conversion rate and time, and (ii) update the ownership of the involved assets, i.e., alter the state of the respective ledgers. Typically, exchange protocols are designed as two-phase lock/unlock processes, where either both parties receive control over the agreed upon assets *atomically*, or no exchange is executed at all. Historically, protocols for achieving a such *fair exchange* [80, 136] required trust in third parties. Blockchains allow us to reduce these trust assumptions [73, 111, 116] by making use of so called *smart contracts*, i.e., distributed programs, the result of which is enforced though consensus of the underlying system [58].

The first approach towards cross-chain asset exchange outside of centralized exchanges is dated back to a forum discussion in 2013 [18], which later became known as *atomic cross-chain swaps* [19, 95]. The initially proposed protocol relied on symmetric (i.e., same on both chains) *hash(ed) timelock contracts* [2, 150]: assets x and y on L_x and L_y are locked such that they can be spent atomically when a secret (hash pre-image) is revealed. Timelocks thereby prevent assets from remaining locked indefinitely in case one of the parties crashes or abandons the exchange (cf. Section 7). However, atomic swaps using symmetric locks are vulnerable to race conditions: exchange parties must be online throughout the entire exchange or risk losing funds [19]. Later adaptions improve on this requirement by using *proofs of transaction inclusion*[2] and escrow smart contracts: asset y is released by the smart contract on L_y if the counterparty can prove it made the agreed upon transfer of x on L_x [8, 59, 96, 112, 179].

A recently proposed alternative to atomic cross-chain swaps are *cryptocurrency-backed assets* [26, 84, 125, 179]. Thereby, an asset x is locked on L_x, while a corresponding representation of this asset $y(x)$ is unlocked on L_y. This asset can then be used just like any other native asset y on L_y, i.e., transferred, exchanged, and used with smart contracts. The latter can be particularly useful if L_x itself only has limited scripting capabilities (such as e.g. Bitcoin). After use, $y(x)$ can be redeemed for the corresponding amount of x. A benefit of this approach over atomic cross-chain swaps is that synchronization of L_x and L_y is performed only twice for any number of swaps, reducing risk of failures. However, this comes at the cost of creating a long term economic dependency between the two systems and the necessity to mitigate exchange rate fluctuations.
Synchronization in Sharded Systems. The goal of sharding [23, 117, 177] is to increase the concurrency of committing transactions by instantiating multiple blockchains, each of which is authoritative over a subset of the state. Assuming that the workload of the system is parallelizable, sharding has the potential of a linear increase (to the number of participating nodes) in throughput, a property named "scale-out" in traditional literature [66]. Concretely, in sharded distributed ledgers different sets of nodes only maintain the state and the ledger of a specific shard. For sharding to be practical, there must be a way to perform transactions

[2] We referred to this as *verification of state evolution*, cf. Section 6.

between shards which guarantees atomic execution. In order for this to be possible, shards must be able to verify proofs about the state of all other shards.

Bootstrapping and Migration. Leveraging the security properties of existing blockchains to bootstrap new systems is another well studied field of cross-chain communication. This idea was first put to practice in *merged mining* [3,105,147,178], where (partial) Proof-of-Work solutions from a parent "chain" are reused as valid proofs-of-work in "child" chains, i.e., miners can mine on the parent chain and the child chain(s) at the same time. Instead of directly re-using solutions, Proof-of-Work can be used to sample committees [68,113, 146,148] to bootstrap other systems, such as blockchains running (permissioned) Byzantine fault tolerant agreement protocols. Alternatively, blockchains can also serve as a public source of randomness [52,57], which in turn can be leveraged as input to new systems, e.g. Proof-of-Stake [65,72]. Proof-of-burn [162], on the other hand, is a technique to migrate between chains, where new assets are created on L_y by destroying assets on L_x.

(Pegged) Sidechains. The definition of the term "sidechain" has seen heated debate in the past years. The word has evolved to a generic term for cross-chain communication, used to refer to (i) ways for blockchains to communicate with each other [16,84], (ii) blockchains which are capable of verifying other blockchains [36,84, 112], and (iii) blockchains which are considered "children" of another blockchain, i.e., whose security model relies on the secure operation of a "parent" chain (at least for some period of time) [36,84]. The capability of verifying the state of another blockchain is also captured by the notion of *cross-chain state verification* via so called "chain-relays" (light clients for L_x deployed on L_y) [59,179]. Similarly, the definitions of "pegged sidechains" [36] and "cryptocurrency-backed assets" [179] both refer to the concept of moving assets from one blockchain to another. We attempt to resolve this misdemeanor in this paper.

4 The Cross-Chain Communication Problem

In this section, we formally define the problem of Correct Cross-Chain Communication (CCC) and provide a reduction to the Fair Exchange problem [30,144], showing that CCC is *impossible without a trusted third party*.

4.1 Model and Specification

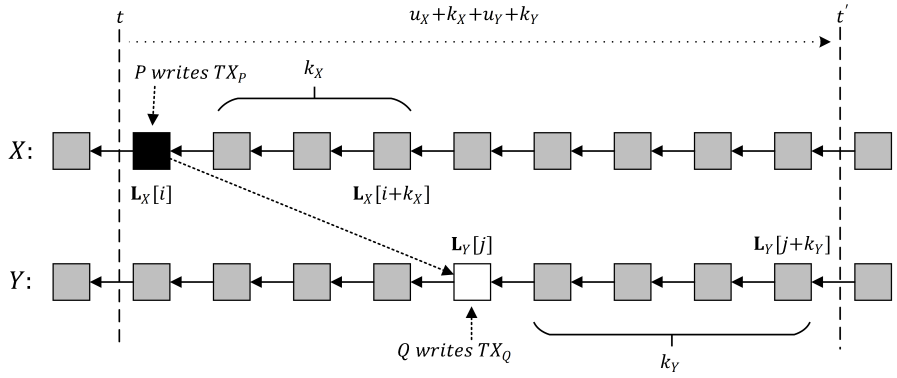
Model. Consider two independent distributed systems X and Y with underlying ledgers L_x and L_y, as defined in Section 2. We assume a *closed* system model as in [123] with a process P running on X and a process Q running on Y. The only way a process can influence the state evolution of the underlying system is by (i) writing a transaction TX to the underlying ledger L, or (ii) by completely 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). Both processes posses descriptions d_P and d_Q of the transactions TX_P and TX_Q they expect. Note: $d_P = desc(\mathrm{TX}_P)$ implies TX_P is valid in X (at time of synchronization), as it cannot be written to L_x otherwise (analogous for d_Q).

We assume P and Q know each other's identity and no (trusted) third party is involved in the communication between the two processes. Further, we assume no bounds on message delay or deviations between local clocks and treat failure to communicate as adversarial behavior. Hence, if P or Q become malicious, we indicate this using boolean "error variables" [82] m_P and m_Q.

Specification. The goal of cross-chain communication can be described as the synchronization of processes P and Q such that Q *writes* TX_Q *to* L_y *if and only if* P *has written* TX_P *to* L_x. Thereby, it must hold that $desc(\mathrm{TX}_P) = d_Q \wedge desc(\mathrm{TX}_Q) = d_P$. The intuition is that TX_P and TX_Q are two transactions which must either both, or neither, be included in L_x and L_y, respectively. For example, they can constitute an exchange of assets which must be completed atomically. A visual representation is provided in Figure 1.

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 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:

Fig. 1: Simplified visualization of CCC. Process Q writes TX_Q only if P has written TX_P. We use k_X and k_Y to denote exemplary persistence delays of X and Y ($k_X = 3$, $k_Y = 4$), and set $u_x = u_y = 0$ for the liveness delays.



Effectiveness: If both P and Q behave correctly and TX_P and TX_Q match the expected descriptions (and are hence 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.

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

Atomicity: There are no outcomes in which P writes TX_P to L_x but Q does not write TX_Q, or Q writes TX_Q to L_y but P did not write TX_P to L_x.

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

Timeliness: Eventually, a process that behaves correctly will write a transaction TX 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 1 (Correct Cross-Chain Communication (CCC)). *Consider two systems X and Y with ledgers L_x and L_y, each of which has Persistence and Liveness. Consider two processes, P on X and Q on Y, with to-be-synchronized transactions TX_P and TX_Q. Then a* correct *cross-chain communication protocol is a protocol which achieves $\text{TX}_P \in \mathsf{L}_x \wedge \text{TX}_Q \in \mathsf{L}_y$ and has Effectiveness, Atomicity, and Timeliness.*

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

4.2 Correct Cross-Chain Communication as Hard as Fair Exchange

Fair Exchange. The fair exchange of digital goods is a well-studied problem in computer science. On a high level, an exchange between two (or more) parties is considered fair if either both parties receive the item they expect, or none do [31]. Fair exchange can be considered a sub-problem of fair secure computation [42], and is known to be impossible without a trusted third party [144, 174]. However, research has circumvented this impossibility under weaker security models, e.g. under the optimistic model where a trusted third party is only consulted in case a party deviates from the correct behavior [31, 62, 120]. Recently, there have been numerous works which use cryptocurrencies such as Bitcoin and Ethereum to construct (optimistic) fair

exchange protocols [27, 42, 73, 111, 115, 119]. Here, smart contracts, i.e., programs or scripts, the result of which is agreed upon and enforced by consensus participants, are leveraged to ensure the exchange happens correctly. Since the actions of a smart contract can be restricted, the latter need not be trusted, as long as the majority of consensus participants in the underlying blockchain behave correctly.

Relating Cross-Chain Communication to Fair Exchange. We proceed to show that Correct Cross-Chain Communication (CCC) is impossible without a trusted third party (TTP) by reducing it to Fair Exchange [30, 31, 144].

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

Proof (Sketch). Consider that the two processes P and Q are parties in a fair exchange. Specifically, P owns an item i_P and wishes to exchange it against an item i_Q owned by Q. Assume TX_P assigns ownership of i_P to Q and TX_Q ownership of i_Q to P ("descriptions" of TX_P and TX_Q). Then, TX_P must be included in L_x and TX_Q in L_y to finalize 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 i_Q and Q receives desired i_P, i.e., P and Q fairly exchange items i_P and i_Q.

A fair exchange protocol must fulfill three properties: *Effectiveness*, *(Strong) Fairness* and *Timeliness* [30, 144]. It is easy to see the Timeliness property of CCC is equivalent to Timeliness of fair exchange as defined in [144]. Effectiveness in fair exchange states that if P and Q behave correctly and do not want to abandon the exchange, and items i_P and i_Q are as expected by Q and P, then at the end of the protocol, P will have the desired i_Q and Q will have the desired i_P [144]. It is easy to see Effectiveness in CCC achieves exactly this property: if P and Q behave correctly and the transaction match the respective descriptions (i.e., assign i_P to Q and i_Q to P), then P will write TX_P and Q will write TX_Q to L_x, resulting in P receiving i_Q and Q receiving i_P. From Persistence and Liveness of L_x and L_y we know both transactions will eventually be written to the local ledgers of P and Q, and consequently appear in the local ledgers of all other participants of X and Y. Strong Fairness in fair exchange states that there is no outcome of the protocol, where P owns i_Q but Q does not own i_P, or Q owns i_P but P does not own i_Q [144]. 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 assume the same asynchronous (explicitly) and deterministic (implicitly) system model (cf. Section 4.1) as [78, 144]. Since P and Q can simply stop participating in the CCC protocol, we also have the same crash failure model as [78, 144]. Hence, we conclude:

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

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

Our result currently holds for the closed model, as in [78, 144]. In the open model, P and Q can be forced to make a decision by the system, i.e., transactions can be written on their behalf if they crash [117]. This can be achieved by leveraging smart contracts, similar to blockchain-based fair exchange protocols, e.g. [73]. As such, we can construct a smart contract on Y which will include TX_Q in L_y as soon as P includes TX_P in L_x, i.e., Q is allowed to crash. A contract, however, can only perform actions based on some prior input. Specifically, before writing TX_Q the contract must receive proof that TX_P was included in L_x. A protocol achieving CCC must hence assume (i) P (or Q) will always submit a proof showing $\text{TX}_P \in \mathsf{L}_x$, i.e., introduce some form of *synchrony* assumptions, or (ii) resort to a TTP. We argue introducing a TTP is equivalent to introducing synchrony assumptions, as discussed in [144], and derive the following corollary:

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

Proof (Sketch). 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. □

5 Classification of Cross-Chain Communication

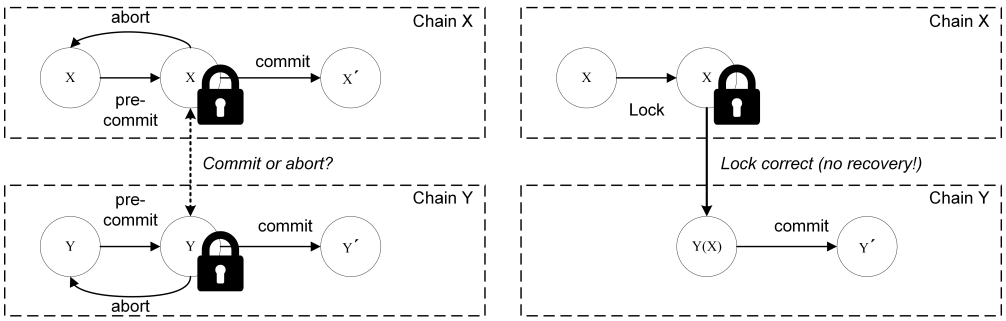
In this section, we define the main properties of cross-chain communication protocols and then present a classification of existing work.

5.1 CCC Protocol Models

Communication Model. We differentiate between two main types of cross-chain communication protocols, based how synchronizations of state transitions across chains is handled. An explanatory visualization is provided in Figure 2.

- **Two-Phase Commit** protocols enforce atomicity of CCC by explicitly introducing an *abort* (or *recover*) step to mitigate communication failures. Typically, two-phase commit is used in protocols which facilitate exchanges [18,96,112] (actions on both X and Y by different parties) or communication involving synchronization across more than two chains (e.g., multi-input transactions in UTXO-based sharding [26,117]).
- **Fire-and-Forget** protocols assume a valid state transition on L_x will eventually result in a state transition on L_y, but *no abort or recovery* phase is explicitly introduced to handle failures. This approach is followed in *one-way* asset and information transfers, such as cryptocurrency-backed assets [168, 179], Proof-of-Burn [162] or merged mining [105, 125]. We note that atomic execution of *multiple* one-way protocols again requires explicit recovery handling.

Fig. 2: Difference between Two-Phase Commit and Fire-and-Forget CCC protocols. Two-Phase Commit is typically used in exchange protocols, or when multiple cross-chain transactions need to be coordinated. Fire-and-Forget, or one way, protocols typically transfer assets or information from X, to Y. Thereby, a "write" lock is obtained on X, preventing any changes to the locked asset or object.



Verification Model. A critical component of CCC is the verification of state transitions which occurred on other chains.

- **Local** verification assumes that only the protocol participants must agree on the correctness of the state transition(s). An example are atomic cross-chain swaps using symmetric HTLC locks [18, 19, 95], where only the exchange partners verify the correctness of the counterparty's lock.
- **Global** or consensus-level verification implies the consensus participants of a chain Y verify and agree that some state transition occurred on X. An example are atomic cross-chain swaps realized via cross-chain state verification [96, 112, 179]: assets are unlocked on Y if a proof for the agreed counter-transfer being executed on X is provided (and accepted by Y).

Blockchain Security Model. Another distinction between CCC protocols is the assumptions made with respect to the security models of interlinked chains.

- **Homogeneous** security implies that we can *explicitly* make assumptions regarding the threat and network model of Y when constructing a communication protocol from X. In practice, this is the case in sharding [23, 25, 117, 177], where every shard is secure by design [117, 177] using, for example, an appropriate form of randomness [155, 164]. As a result, shards have "uniform" security, i.e., it is equally difficult for an adversary to compromise any shard. This further applies to (child) chains which have hierarchical dependency to some parent chain, i.e., the child chain's security depends on that of the parent (e.g., merged mining [3, 105, 125, 165]). Since, if the parent fails, so does the child, we are forced to assume the parent's security holds when communicating with it from the child. Note, this relation does not necessarily hold inversely: by definition, a child can fail without affecting the parent (contrary to the critical impact of a failing shard).
- **Heterogeneous**. Interlinked chains cannot be assumed to have identical rule sets/data structures. In practice, this implies *any user can create their own chain* and, in the absence of pre-defined specifications or rules ensuring e.g. sufficient honest participants exist, *no generic security assumptions can be relied on*. Hence, a chain, even if capable of verifying the consensus rules of another chain, cannot be sure that the received information is indeed valid (except if fully validated, cf. Section 6.1) [36]. Typically, CCC protocols in the heterogeneous security model make additional assumptions regarding interlinked chains, restricting applicability to a subset of existing systems.
- **Blockchain agnostic protocols**. CCC protocols which assume a local verification model require only verification and agreement among CCC participants. That is, they do not require direct involvement of the consensus participants of the interlinked chains in the synchronization process. Consequently, such protocols are applicable to both homo- and heterogeneous security models. Notable examples are atomic cross-chain swaps which use HTLCs [18, 19, 95, 128] or similar symmetric cryptographic locks [46, 49, 75, 129, 140, 149, 166] (cf. Section 7), as well as centralized and custodial exchange protocols [17, 41, 169].

5.2 Protocol Classification

We proceed to provide a (non-exhaustive) overview of how existing cross-chain communication protocols fit into the introduced classification. A visual representation is provided in Figure 3.

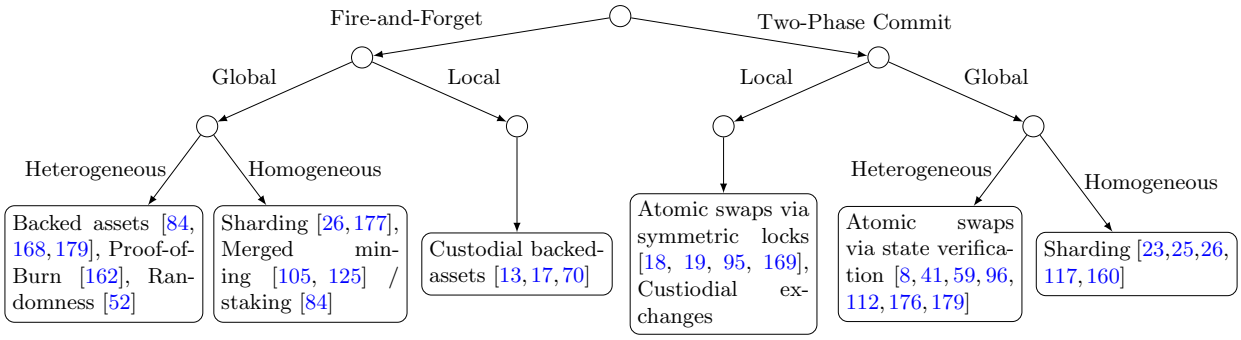


Fig. 3: Classification of cross-chain communication protocols. We provide a non-exhaustive list of example protocols representative for each combination. Protocols following the local verification model do not differentiate between homogeneous and heterogeneous security models.

Fire-and-Forget

– **Global Heterogeneous** one-way protocols facilitate transfers of information and assets without explicit security assumptions for interlinked chains. Notable examples are bootstrapping protocols such as Proof-of-Burn [162], or crypto-currency-backed assets [168,179], where trust in the correct operation of a smart contract on Y is assumed as restriction. Thereby, collateral can be used to mitigate the lack of homogeneous security assumptions.
– **Global Homogeneous** one-way protocols comprise communication (i) in sharded blockchains [26,177] and (ii) between chains in a parent-child relation as in merged mining [3,105] and merged staking [84]. The main difference to heterogeneous protocols here is that both synchronization and verification of state transitions are handled directly by consensus participants, without the need for additional measures such as collateral.
– **Local (Agnostic)**. One-way transfers operating only under a local verification module are observed in *custodial* protocols for cryptocurrency-backed assets [13,17,70]. Thereby, a (set of) coordinator(s) takes full control of the issuing process, and hence participants need only to verify the presence of the correct signature(s). While this verification is often outsourced to a globally verified smart contract, this is not necessary in theory, if no additional security measures such as collateral are introduced.

Two-Phase-Commit

– **Global Heterogeneous** protocols using two-phase commit are mainly observed in atomic cross-chain exchanges, which leverage a coordinator to hold exchanged assets on chain Y and release the latter only upon receiving a proof of the agreed payment on chain X. The coordinator can thereby be realized directly by the consensus participants of involved chains [176], or a non-trusted escrow using smart contracts [8, 59,96,112,179] or trusted hardware [41].
– **Global Homogeneous** two-phase commit protocols are specific to sharding [23,25,26,117,160]. The main difference to one-way (fire-and-forget) cross-shard communication protocols lies in the explicit recovery mode. Two-phase commit protocols can be either client-driven [117], where transaction submitters are responsible for delivering the protocol messages to the respective shards, or shard-driven [23], where shard consensus participants handle this task.
– **Local (Agnostic)** verification in two-phase commit protocols is used in atomic exchanges relying on symmetric cryptographic locks [18,19,95,169]. Thereby, assets are locked with the same release condition across multiple chains (cf. Section 7), as in the case of HTLCs [2,150]. In contrast to global verification, these protocols are typically interactive, i.e., require exchange parties to remain online or resort to services such as watchtowers [34,130].

6 Cross-chain State Verification and Validation

A critical component of cross-chain communication is the verification of the state evolution of a chain X from within another chain Y. We present a classification discussing available (non-interactive) cryptographic proving techniques, i.e., where verification does not require trust in third parties or running fully validating nodes in all involved blockchains (Section 6.1). Consequently, we discuss other verification approaches, loosening these constraints (Section 6.2).

6.1 Classification of State Verification and Validation Proofs

Specifically, 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 if a specific state *exists*, i.e., is reachable but has not necessarily been agreed upon by the consensus participants. A representative example is the verification of Proof-of-Work in merged mining [3,105]: the child chain Y only verifies that the hash of the Y candidate block was included in a X block, the PoW hash of which exceeds the difficulty

target of Y. Note: Y does not care whether the block is actually part of L_x. Another example is the use of blockchains as a public source of randomness [52, 57, 65, 72].

Verification of State Agreement. In addition to the existence of a state, a proof can provide sufficient data to verify that consensus has been achieved on that state. Typically, the functionality of this verification is identical to that of blockchain light clients [1, 11, 141]: instead of verifying the entire blockchain of X, all block headers but only transactions relevant to the CCC protocol are verified (and stored) on Y. The assumption thereby is that an invalid block will not be included in the verified blockchain under correct operation [127, 141]. Block headers can be understood as the information stored in a block, excluding transactions, which are typically referenced using a vector commitment [64] (or some other form of cryptographic accumulator [39]), e.g. Merkle trees [134]. We discuss how proofs of state agreement differ depending on the underlying consensus mechanism below (non-exhaustive):

- **Proof-of-Work.** To verify agreement in PoW blockchains, a primitive called *(Non-interactive) Proofs of Proof-of-Work* [108, 109], also referred to as SPV (simplified payment verification) [141] is used. Thereby, the verifier of a proof must for each block at least check that (i) the PoW meets the specified difficulty target, (ii) the specified target is in accordance with the difficulty adjustment mechanism and (iii) the block contains a reference to the previous block in the chain [1, 179]. The first known implementation of cross-chain state agreement verification (for PoW blockchains) is BTCRelay [4]: a smart contract which allows to verify the state of Bitcoin on Ethereum[3].
- **Proof-of-Stake.** If the verified chain uses Proof-of-Stake, the proofs represent a dynamic collection of signatures, capturing the underlying stake of the chain. These are referred to as *Proofs of Proof-of-Stake* (PoPos) and a scheme in this direction was put forth in [84].
- **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 the majority of the committee. If the committee membership is dyncamically changing, the verification process need to capture the rotating configuration of the committee, which can incur significant cost for parties that rarely synchronize.

Sub-linear State Agreement Proofs. Verifying all block headers results in proof complexity linear in the size of the blockchain. However, there exist techniques for achieving *sub-linear* (logarithmic in the size of the chain) complexity, which rely on probabilistic verification. For PoW blockchains, we are aware of two approaches: Superblock (Ni)PoPoWs [36, 108, 109, 138] and FlyClient [127]. Both techniques rely on probabilistic sampling but differ in the selection heuristic. Superblock (Ni)PoPoWs sample blocks which exceed the required PoW difficulty[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 [52]). Such probabilistic sampling techniques may be potentially applicable to Proof-of-Stake blockchains [127], however, to the best of our knowledge, no concrete schemes have been put forth at the time of writing. For blockchains maintained by a static BFT committee, the verified signatures can be combined into aggregate signatures [113, 114] for optimization purposes. These signature techniques are well known and invented prior to blockchains, and we hence do not elaborate further on these schemes. In the dynamic setting, skipchains [79, 118, 142], i.e., double-linked skiplists which enable sub-linear crawling of identity chains, can reduce costs from linear to logarithmic (to the number of configurations).

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

Verification of State Validity. Even though a block is believed to have consensus, it may not be a valid block if it contains invalid transactions or state transitions (e.g. a PoW block meeting the difficulty

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

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

requirements, but containing invalid transactions). Fully validating nodes will reject these blocks as they check all included transactions. However, in the case of cross-chain communication where chains only verify state agreement but not that the blocks are valid, detection is not directly possible. We classify two broad categories of techniques to enable such chains, and non-full nodes (i.e., light clients), to reject invalid blocks:

- In **proactive** state validation, nodes ensure that blocks are valid before accepting them. Apart from requiring participants to run fully validating nodes, this can be achieved by leveraging "validity proofs" through succinct proofs of knowledge using systems such as SNARKs [45], STARKs [37] or Bulletproofs [56]. First schemes for blockchains offering such proofs for each state transition are put forth in [38, 48, 133]. Informally speaking, this is a "guilty until proven innocent model": nodes assume blocks that have consensus are invalid until proven otherwise.
- In **reactive** state validation, nodes follow an "innocent until proven guilty" model. It is assumed that blocks that have consensus only contain valid state transition, unless a state transition "fraud proof" [24] is created. Fraud proofs typically are proofs of state evolution, i.e., opening of the transaction vector commitment in the invalid block at the index of the invalid transaction, e.g. via Merkle tree paths. Depending on the observed failure, more data may be necessary to determine inconsistencies (e.g. Merkle tree paths for conflicting transactions in case of a double spend).

Verification of Data Availability. Consensus participants may produce a block header, but not release the included transactions, preventing other participants from validating the correctness of the state transition. To this end, verification of state validity can be complimented by verification of data availability. A scheme for such proofs was put forth in [24], which allows to verify with high probability that all of the data in a block is available, by sampling a constant number of chunks in an erasure-coded version of a block.

6.2 Other Verification Techniques

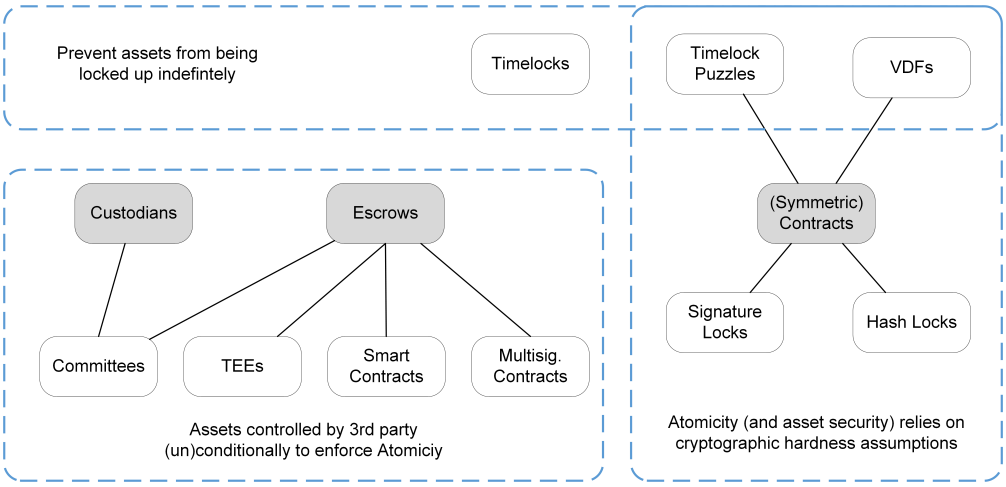
We discussed (non-interactive) cryptographic proofs, involving only communicating parties and consensus participants. We now overview other approaches:

- **Direct Observation.** The simplest approach to cross-chain verification is to require all participants of a CCC protocol to run (fully validating) nodes in all involved chains. This is often the case in *local* verification models, e.g. as in atomic swaps using symmetric locks such as HTLCs [19, 95], but also in parent-child settings where one chain by design verifies or validates the other [36, 84, 125].
- **Watchtowers.** As a fallback mechanism to direct observation it is possible to employ so called *watchtowers*, i.e., service providers which take action in case a party crashes [33, 34, 107, 130].
- **Coordinators.** Another option is to fully trust in a third party to perform the verification. There is, however, a difference between relying on consensus participants [70, 117, 124] and parties external to the verifying blockchain [169]. Specifically, if consensus participants misbehave this, in most cases, indicates a failure of the chain itself, whereas it is not possible to make any assumptions on the operation of external parties.
- **Verification Games.** Finally, rather than fully trusting coordinators, they can rather be used as a pure optimistic performance improvement by introducing dispute handling mechanisms to the verification process: user can provide fraud proofs [24] or accuse coordinators of misbehavior requiring them to prove correct operation [91, 106, 167].

7 Locking Techniques for Atomicity in CCC

The majority of cross-chain communication protocols discussed in this paper focus on the transfer and exchange of digital assets. One of the required properties for CCC is Atomicity (cf. Section 4.1), that is, either assets are exchange across chains successfully, or no asset leaves its original chain/owner. In practice, this is achieved by *locking mechanisms*: assets in different chains are locked until the fulfillment of a certain condition. We discuss possible approaches in the following and provide an accompanying visualization, capturing possible combinations, in Figure 4

Fig. 4: Overview of locking techniques and possible combinations. Custodians can be realized using Committees. Escrows, with restricted and pre-authorized access to assets, can be implemented using Multisignature Contracts, TEEs, or Smart Contracts. (Symmetric) contracts allow to lock assets without requiring a third party and rely on cryptographic hardness assumptions. Timelocks are used to prevent assets from being locked up indefinitely in case of crash failures (similarly, Timelock Puzzles and VDFs). In practice, it is possible to combine techniques from all three groups: e.g. use a Multisig. contract to create an Escrow, but also allow an alternative execution path by using a Hash Lock - and combine with a Timelock to release assets in case of a timeout (e.g. used in HTLC atomic swaps [19,95] and payment channels [150]).



Coordinators. A *coordinator* is a distinguished participant that helps other protocol participants in achieving agreement for either committing or aborting the cross-chain transfer. We differentiate between two types of coordinators depending on their access to (honest) participants' assets.

- **Custodians** receive *unconditional* control over the participant's funds and are thus *trusted* to release the latter as instructed by the protocol rules. A mitigation of this trust assumption can be achieved via collateralization and penalties [91,92,167,168,179].
- **Escrows** receive control over the participant's funds *conditional* to certain prearranged constraints being fulfilled. An Escrow thus cannot steal the locked assets, as long as the underlying ledger is operating correctly, and can only fail to take action, i.e. crash. From game theoretic perspective, Escrows are typically expected to not gain from misbehaving and are hence often referred to as "untrusted" third parties.

We proceed to overview techniques to reduce the trust put in coordinators below.

- **Committees.** The trust assumptions are distributed among a set of N committee members. Decisions require the acknowledgement (e.g. digital signature) of at least $M \leq N$ members, whereby consensus can be achieved via Byzantine Fault Tolerante (BFT) agreement protocols such as PBFT [63]. An important distinction to make here is between *static*, i.e., unchanged over time (usually permissioned), and *dynamic* committees, where a pre-defined mechanism is responsible for member election. The latter is a well studied problem, e.g. in Proof-of-Work [69,113,114,146] and Proof-of-Stake [43,67,110,137] blockchains. Practical examples for such CCC protocols relying on committees include [13,70,117].
- **Multisignature Contracts.** A special case of committees, allowing to transform a Custodian into an Escrow, are multisignature contracts, which require the signature of the participant P in addition to those of the (subset of) committee members, i.e., $P + M, M \leq N$. As a result, the committee can only execute actions pre-authorized by the participant.
- **Trusted Execution Environments (TEEs).** TEE is based on a piece hardware trusted for integrity of code execution and confidentiality of handled data. While TEEs do not support arbitrary long code and

data, they can be leveraged to provide confidentiality and integrity guarantees within an untrusted part of the system [29, 99]. As such, TEEs can be entrusted to store private keys and perform computations, i.e., be used as Escrows. However, vulnerabilities to side-channel attacks [86, 173] require the use of dedicated libraries [55, 88, 143] limiting performance and available functionality.
- **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 system, making this a useful construction for Escrows. Depending on the system properties, smart contracts can be (near) Turing complete [58], or limited to a subset of operations [141, 157]. In addition, smart contracts can be used to collateralize CCC participants, penalize misbehavior [91, 92, 167] or pay premium for correct participation [90] – even if the custodians are located on different chains, which may have no support for smart contracts themselves [179]. Recently, smart contracts have been also used to verify succinct proofs of knowledge [74, 180].

(Symmetric) Contracts. An alternative to coordinators consists in using locks stemming security from cryptographic hardness assumptions. Observations in practice show these techniques are typically used in two-phase commit CCC, where the same (*symmetric*) locks are created on both chains and released atomically. We provide an overview, differentiating between the cryptographic primitives relied upon.

- **Hash Locks** rely 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 uniformly at random by P (i.e., the initiator of the protocol) [18, 19, 95, 128].
- **Signature-based Locks**. Protocols based on hash locks have limited interoperability as they require that both cryptocurrencies support the same hash function within their script language. 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) [46, 49, 75, 129, 140, 149, 166].
- **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 [151] build upon inherently sequential functions where the result is only revealed after a predefined number of operations are performed. Verifiable delay functions [47] improve upon timelock puzzles on that the correctness of the result for the challenge is publicly verifiable. This functionality can also be simulated by releasing parts of the preimage of a hash lock interactively bit by bit, until it can be brute forced [41].

Timelocks. To ensure that assets are *not locked up indefinitely* in case of a crash failure, all of the above locking mechanisms can be complimented with *timelocks*: after expiry of the timelock, assets are returned to their original owner. Thereby, we differentiate between two types of timelocks:

- **Absolute timelocks**, where a transaction becomes valid only after a specified delay. The latter can be defined in absolute time units or by specifying a timestamp located in the future, after which the transaction becomes valid. Alternatively, the delay can be defined in *confirmations* [5], i.e., assuming transaction TX was included in $\mathsf{L}[i]$, then TX becomes valid when the chain reaches ledger state $\mathsf{L}[j]$ where $j = i + c$, which c denoting the number of required confirmations ($i, j, c \in \mathbb{N}$).
- **Relative timelocks**, where a transaction TX_2 becomes valid only after a given time value or number of confirmations 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 [150]. While more practical than absolute timelocks (no need for external clock), as of this writing, we are not aware of schemes allowing the creation of relative timelocks across ledgers.

8 Implications for Blockchain, Threat, Network, and Privacy Models

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

8.1 Threat Model

Security and Adversary Model. Both X and Y can have a well defined security model on their own. However, these security models are not necessarily the same and even further, it might not be trivial to compare the guarantees they provide. For instance, X may rely on PoW and thus assume that adversarial hash computation is bound by $\alpha \leq 33\%$ [76, 85, 154]. On the other hand, Y may use PoS and similarly assume that the adversary's stake in the system is bound by $\beta \leq 33\%$. While similar at first glance, the cost of accumulating stake [77, 83] may be lower than that of accumulating computational power, or vice-versa [50]. Since permissionless distributed ledgers (such as PoW or PoS) are not Sybil resistant [71], i.e., provide weak identities at best, quantifying adversary strength is nearly impossible, even within a single ledger [32]. However, this task becomes even more difficult in the cross-chain setting: not only can consensus participants (i) "hop" between different chains [122, 135], destabilizing involved systems, but also (ii) be susceptible to bribing attacks, which can be executed cross-chain, making detection unlikely [104, 132].
Consensus Finality Guarantees. Interlinked chains X and Y may assume different finality guarantees in their ledgers. Consider the following: X accepts a transaction as valid when confirmed by k subsequent blocks, e.g. as in PoW blockchains [81]; Y, on the other hand, deems transactions valid as soon as they are written to the ledger ($k = 1$, e.g. [21]). A CCC protocol triggers a state transition on Y conditioned on a transaction included in X, however, later an (accidental) fork occurs on X (perhaps deeper than k). While the state of X will be reverted, this may not be possible in Y according to consensus rules - although the Atomicity property of CCC would require such measures.
Replay Attacks. Replay attacks on state verification, i.e., where proofs are re-submitted multiple times or on multiple chains, can result in failures such as double spending. Protections involve the use of sequence numbers, or chains keeping track of previously processed proofs [60, 131, 160]. Special consideration may be needed in case of permanent blockchain forks, as this may require updating the way verification is performed [131, 179].
Increasing Verification Cost. An adversary can increase the cost of the verification of a transaction across chains. For instance, a spam attack in a chain makes the ledger grow in size, increasing both verification time and cost. This in turn may impair cross-chain state verification, especially in heterogeneous settings, where verification of external consensus is typically an expensive operation [179]. In sharded systems, an adversary can attempt to create transactions, the validation of which requires information from (almost) all shards, significantly increasing cost and defeating the purpose of sharding in the first place.
Composability Attacks. We recall, in blockchains with *stablizing* [28, 171] consensus, a security parameter k is used to denote the number of blocks or *confirmation* [5] a transaction should have, before being accepted as secure [81, 141, 145], i.e., with the probability of a reversion being negligible. The same applies to state agreement and state evolution proofs. In addition, the value linked a verified transaction must be considered: the higher the potential gain by an adversary, the higher the risk of an attack, and the more confirmations should be required [159]. However, following recent works [182], we argue *at least* the entire *composition* of a block must be considered, as this is the total value an adversary can gain from executing a successful attack[5].

8.2 Network model

Synchrony Across Chains. The absence of a global clock across chains requires to either agree and trust a third party as external clock, or rely on chain-dependent time definition, such as the block generation rate [81], hindering a seamless synchronization across chains [81, 179]. Several factors, such as the instance

[5] Recall: for a transaction to be reversed or modified, the entire block must be altered.

of the consensus algorithm, computation and communication capabilities of consensus participants or peer-to-peer network delay must be considered for a correct operation of cross-chain communication protocols, especially if timelocks are used.

Exchange Rates. In economically driven cross-chain communication, the exchange rate of assets is crucial for the economic viability of a protocol. For instance, while the exchange of assets can be beneficial for both users given the exchange rate at time t, this may no longer be the case at time $t+1$ provided that exchange rates have changed [90]. Finally, there exists protocols that rely on collateral (i.e., coins locked in a smart contract) to make third parties accountable [92, 168, 179]. Here, the exchange rate is crucial to ensure that collateral has sufficient value to punish misbehavior, and stabilization measures are necessary to mitigate both short and long term fluctuations.

Data Availability. Protocols leveraging verification of state agreement or validity across chains typically rely on timely arrival of proofs and accompanying data (block headers, transactions, ...). Furthermore, existing sub-linear state verification techniques relying on probabilistic sampling require additional data to be included in the verified blockchain [109, 127]. If an adversary can exclude this data from the chain, these protocols not only become less efficient but may potentially exhibit vulnerabilities [127]. This is a particular problem in heterogeneous settings, if data availability is not enforced by consensus, e.g. if protocols are deployed via velvet forks [181]. One possible solution is to include data availability proofs [23], at the cost of increasing complexity of the process.

8.3 Blockchain Model

Cryptographic Primitives. Interconnected chains X and Y may leverage different cryptographic schemes, or different instances of the same scheme. Thereby, cross-chain communication often requires compatible cryptographic primitives: a CCC protocol between a system X using ECDSA [100] as its digital signature scheme and a system Y using Schnorr [156] is only possible if both schemes are instantiated over the same elliptic curve [129]. Similarly, HTLC-based protocols require that the domain of the hash function has the same size in both X and Y- otherwise the protocol is prone to *oversize preimage attacks* [102].

Language Expressiveness. The functionality of CCC protocols is typically restricted by the computational operations supported by the involved chains. These can reach from near Turing complete environments [58], over restricted operation sets [141, 157], to the (near) absence of scripting functionality altogether [9, 53]. CCC protocols must hence (i) consider the operations supported by both X and Y and leverage the intersecting functionality [129]; or (ii) move assets from chain with limited functionality to those with high(er) language expressiveness [70, 168, 179].

8.4 Privacy and Linkability

Privacy is crucial in any financial interaction and thus in cross-chain communication as well. Ideally it should not be possible for an observer to determine what two events have been synchronized across chains (e.g., what two assets have been exchanged and by whom). Among other advantages, this improves the *fungibility* of payments. However, there exist several privacy attack vectors in cross-chain communication: (i) recent works [87, 128] show attacks leveraging the locking mechanism and some countermeasures have been proposed [128, 129, 153]; (ii) heuristics to explore blocks from different cryptocurrencies [105] as well as forks [163] to cluster miner and user accounts [97]; (iii) CCC protocols leveraging coordinators, similar to and payment hubs [89, 103], also lead to privacy leakages that enable further account clustering [175]. Recent works [87, 93, 166] propose measures that allow to preserve the anonymity of participants, if added to CCC protocols.

9 Related Work

Surveys and Reports. The first work discussing cross-chain communication, excluding forum discussions, is a technical report by Back et al. [36]. The authors introduce the term "sidechain" and present how assets can

be transferred between two chains using a committee of custodians or SPV proofs in a homogeneous security model. A more recent report by Buterin discusses how cross-chain exchanges can be achieved via custodians, escrows, HTLCs and cross-chain state verification, and provides a high level discussion of possible failures in cross-chain communication [59]. Siris et al. provide an iterative overview of protocols for atomic cross-chain swaps and "sidechains" [158], focusing mostly on community driven efforts, rather than academic publications. Similarly, Johnson et al. discuss open source interoperability projects related to Ethereum [101], while Robinson evaluates Ethereum as a coordination platform for communication among other blockchains [152]. Bennik et al. [40] and similarly Miraz et al. [139] summarize technical details of HTLC atomic cross-chain swaps.

Interoprability Platforms. The idea of using a specialized sharded blockchains to interconnect existing distributed ledgers was first introduced in [172]. Thereby, individual shards, which are coordinated via a parent chain running a Byzantine fault tolerant agreement protocol, connect to and import assets from existing blockchains using techniques such as [84, 168, 179]. A formal treatment of such a design, extending the functionality from asset transfer to distributed computation, is presented in [126]. Other projects following a similar mentality include [20, 98, 121, 161, 170].

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

11 Acknowledgements

The authors would like to thank Georgia Avarikioti, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, and Edgar Weippl for helpful comments and insightful discussions. 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), and the competence center SBA-K1 funded by COMET. Mustafa Al-Bassam is funded by a scholarship from the Alan Turing Institute. This research was also funded Chaincode Labs and the Austrian Science Fund (FWF) through the Meitner program.

References

1. Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv, accessed: 2018-05-16
2. Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts, accessed: 2018-05-16
3. Bitcoin wiki: Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification, accessed: 2018-05-03
4. Btcrelay. https://github.com/ethereum/btcrelay, accessed 2019-08-15
5. Confirmations. https://en.bitcoin.it/wiki/Confirmation, accessed: 2018-11-28
6. Dogerelay. https://github.com/dogethereum/dogerelay, accessed 2019-08-15

7. Eth-eos-relay. https://github.com/EveripediaNetwork/eth-eos-relay, accessed 2019-08-15
8. Ethereum contract allowing ether to be obtained with bitcoin. https://github.com/ethers/EthereumBitcoinSwap, accessed: 2018-10-30
9. Monero reference implementation. https://github.com/monero-project/monero, accessed: 2018-07-30
10. Parity-Bridge. https://github.com/paritytech/parity-bridge, accessed 2019-08-15
11. The parity light protocol - wiki. https://wiki.parity.io/The-Parity-Light-Protocol-(PIP), accessed: 2018-10-30
12. Peace relay. https://github.com/loiluu/peacerelay, accessed 2019-08-15
13. Poa bridge. https://github.com/poanetwork/poa-bridge, accessed: 2018-05-23
14. Project alchemy. https://github.com/ConsenSys/Project-Alchemy, accessed 2019-08-15
15. Project waterloo. https://blog.kyber.network/waterloo-a-decentralized-practical-bridge-between-eos-and-ethereum- accessed 2019-08-15
16. Sidechains. https://en.bitcoin.it/wiki/Sidechain, accessed: 2019-08-30
17. Wrapped bitcoin. https://www.wbtc.network/assets/wrapped-tokens-whitepaper.pdf, accessed: 2018-05-03
18. Alt chains and atomic transfers. bitcointalk.org (2013), https://bitcointalk.org/index.php?topic=193281.msg2003765#msg2003765
19. Atomic swap. Bitcoin Wiki (2013), https://en.bitcoin.it/wiki/Atomic_swap
20. Wanchain whitepaper. https://www.wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf (2017)
21. Abraham, I., Gueta, G., Malkhi, D.: Hot-stuff the linear, optimal-resilience, one-message bft devil. arXiv:1803.05069 (2018), https://arxiv.org/pdf/1803.05069.pdf
22. Al-Bassam, M.: Lazyledger: A distributed data availability ledger with client-side smart contracts (2019), https://arxiv.org/pdf/1905.09274.pdf
23. Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., Danezis, G.: Chainspace: A sharded smart contracts platform. In: 2018 Network and Distributed System Security Symposium (NDSS) (2018)
24. Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. CoRR **abs/1809.09044** (2018), http://arxiv.org/abs/1809.09044
25. Alp, E.C., Kokoris-Kogias, E., Fragkouli, G., Ford, B.: Rethinking general-purpose decentralized computing. In: Proceedings of the Workshop on Hot Topics in Operating Systems. pp. 105–112. ACM (2019)
26. Androulaki, E., Cachin, C., De Caro, A., Kokoris-Kogias, E.: Channels: Horizontal scaling and confidentiality on permissioned blockchains. In: European Symposium on Research in Computer Security. pp. 111–131. Springer (2018)
27. Andrychowicz, M.: Multiparty computation protocols based on cryptocurrencies (2015), https://depotuw.ceon.pl/bitstream/handle/item/1327/dis.pdf, accessed: 2017-02-15
28. Angluin, D., Fischer, M.J., Jiang, H.: Stabilizing consensus in mobile networks. In: Distributed Computing in Sensor Systems. pp. 37–50. Springer (2006), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.1040&rep=rep1&type=pdf
29. ARM Ltd.: TrustZone. https://www.arm.com/products/security-on-arm/trustzone. Accessed May 2017 (2017)
30. Asokan, N.: Fairness in electronic commerce (1998)
31. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 591–606. Springer (1998)
32. Avarikioti, G., Käppeli, L., Wang, Y., Wattenhofer, R.: Bitcoin security under temporary dishonest majority. In: 23rd Financial Cryptography and Data Security (FC) (2019), https://www.tik.ee.ethz.ch/file/ab83461dc5ca3b739c079a27f3757e94/bitcoin%20security%20under%20temporary%20dishonest%20majority.pdf
33. Avarikioti, G., Kogias, E.K., Wattenhofer, R.: Brick: Asynchronous state channels. arXiv preprint arXiv:1905.11360 (2019)
34. Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards secure and efficient payment channels. arXiv preprint arXiv:1811.12740 (2018)
35. Babaoglu, O., Toueg, S.: Understanding non-blocking atomic commitment. Distributed systems pp. 147–168 (1993)
36. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014), https://blockstream.com/sidechains.pdf
37. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptology ePrint Archive **2018**, 46 (2018)
38. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Theory of Cryptography Conference. pp. 31–60. Springer (2016)

39. Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 274–285. Springer (1993)
40. Bennink, P., Gijtenbeek, L.v., Deventer, O.v., Everts, M.: An analysis of atomic swaps on and between ethereum blockchains using smart contracts. Tech. report (2018), https://work.delaat.net/rp/2017-2018/p42/report.pdf
41. Bentov, I., Ji, Y., Zhang, F., Li, Y., Zhao, X., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. Cryptology ePrint Archive, Report 2017/1153 (2017), https://eprint.iacr.org/2017/1153.pdf, accessed:2017-12-04
42. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Advances in Cryptology–CRYPTO 2014. pp. 421–439. Springer (2014), http://eprint.iacr.org/2014/129.pdf
43. Bentov, I., Pass, R., Shi, E.: Snow white: Provably secure proofs of stake (2016), https://eprint.iacr.org/2016/919.pdf, accessed: 2016-11-08
44. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems, vol. 370. Addison-wesley New York (1987)
45. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 326–349. ACM (2012)
46. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. Cryptology ePrint Archive, Report 2018/601 (2018), https://eprint.iacr.org/2018/601.pdf
47. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (2018)
48. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. Cryptology ePrint Archive, Report 2018/1188 (2018), https://eprint.iacr.org/2018/1188.pdf, https://eprint.iacr.org/2018/1188
49. Boneh, D., Naor, M.: Timed commitments. In: Annual International Cryptology Conference. pp. 236–254. Springer (2000)
50. Bonneau, J.: Why buy when you can rent? bribery attacks on bitcoin consensus. In: BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin and Blockchain Research (February 2016), http://fc16.ifca.ai/bitcoin/papers/Bon16b.pdf
51. Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source (2015), https://eprint.iacr.org/2015/1015.pdf, accessed: 2015-10-25
52. Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source. IACR Cryptology ePrint Archive **2015**, 1015 (2015)
53. Bonneau, J., Miller, A.: Fawkescoin: Bitcoin without public-key crypto. In: Security Protocols XXII. pp. 350–358. Springer (2014), http://www.jbonneau.com/doc/BM14-SPW-fawkescoin.pdf
54. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: IEEE Symposium on Security and Privacy (2015), http://www.ieee-security.org/TC/SP2015/papers-archived/6949a104.pdf
55. Brasser, F., Capkun, S., Dmitrienko, A., Frassetto, T., Kostiainen, K., Müller, U., Sadeghi, A.R.: Dr. sgx: Hardening sgx enclaves against cache attacks with data location randomization. arXiv preprint arXiv:1709.09917 (2017)
56. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Efficient range proofs for confidential transactions (2017), http://web.stanford.edu/~buenz/pubs/bulletproofs.pdf, accessed:2017-11-10
57. Bünz, B., Goldfeder, S., Bonneau, J.: Proofs-of-delay and randomness beacons in ethereum (2017)
58. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform (2014), https://github.com/ethereum/wiki/wiki/White-Paper, accessed: 2016-08-22
59. Buterin, V.: Chain interoperability. Tech. report (2016), https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf, accessed: 2017-03-25
60. Buterin, V.: Cross-shard contract yanking. https://ethresear.ch/t/cross-shard-contract-yanking/1450 (2018)
61. Cachin, C.: Architecture of the hyperledger blockchain fabric (2016), https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf, accessed: 2016-08-10
62. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Annual International Cryptology Conference. pp. 93–111. Springer (2000)
63. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186 (1999), http://pmg.csail.mit.edu/papers/osdi99.pdf
64. Catalano, D., Fiore, D.: Vector commitments and their applications. In: International Workshop on Public Key Cryptography. pp. 55–72. Springer (2013)

65. Chepurnoy, A., Duong, T., Fan, L., Zhou, H.S.: Twinscoin: A cryptocurrency via proof-of-work and proof-of-stake (2017), http://eprint.iacr.org/2017/232.pdf, accessed: 2017-03-22
66. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al.: Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS) **31**(3), 8 (2013)
67. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573 (2017), http://eprint.iacr.org/2017/573.pdf, accessed: 2017-06-29
68. Decker, C., Seidel, J., Wattenhofer, R.: Bitcoin meets strong consistency. In: Proceedings of the 17th International Conference on Distributed Computing and Networking. p. 13. ACM (2016)
69. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and mtgox. In: Computer Security-ESORICS 2014. pp. 313–326. Springer (2014), http://www.tik.ee.ethz.ch/file/7e4a7f3f2991784786037285f4876f5c/malleability.pdf
70. Dilley, J., Poelstra, A., Wilkins, J., Piekarska, M., Gorlick, B., Friedenbach, M.: Strong federations: An interoperable blockchain solution to centralized third party risks. arXiv preprint arXiv:1612.05491 (2016)
71. Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems. pp. 251–260. Springer (2002), http://www.cs.cornell.edu/people/egs/cs6460-spring10/sybil.pdf
72. Duong, T., Fan, L., Zhou, H.S.: 2-hop blockchain: Combining proof-of-work and proof-of-stake securely. Cryptology ePrint Archive, Report 2016/716 (2016), https://eprint.iacr.org/2016/716.pdf, accessed: 2017-02-06
73. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984. ACM (2018)
74. Eberhardt, J., Tai, S.: Zokrates-scalable privacy-preserving off-chain computations
75. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: CCS (2019)
76. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security. pp. 436–454. Springer (2014), http://arxiv.org/pdf/1311.0243
77. Fanti, G., Kogan, L., Oh, S., Ruan, K., Viswanath, P., Wang, G.: Compounding of wealth in proof-of-stake cryptocurrencies. arXiv preprint arXiv:1809.07468 (2018)
78. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. vol. 32, pp. 374–382. ACM (1985), http://macs.citadel.edu/rudolphg/csci604/ImpossibilityofConsensus.pdf
79. Ford, B., Gasser, L., Kogias, E.K., Jovanovic, P.: Cryptographically verifiable data structure having multi-hop forward and backwards links and associated systems and methods (Dec 13 2018), uS Patent App. 15/618,653
80. Franklin, M., Tsudik, G.: Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In: International Conference on Financial Cryptography. pp. 90–102. Springer (1998)
81. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty (2016), http://eprint.iacr.org/2016/1048.pdf, accessed: 2017-02-06
82. Gärtner, F.C.: Specifications for fault tolerance: A comedy of failures (1998)
83. Gaži, P., Kiayias, A., Russell, A.: Stake-bleeding attacks on proof-of-stake blockchains. Cryptology ePrint Archive, Report 2018/248 (2018), https://eprint.iacr.org/2018/248.pdf, accessed:2018-03-12
84. Gazi, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. IEEE Security and Privacy. IEEE (2019)
85. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdo rf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC. pp. 3–16. ACM (2016)
86. Götzfried, J., Eckert, M., Schinzel, S., Müller, T.: Cache attacks on intel sgx. In: Proceedings of the 10th European Workshop on Systems Security. p. 2. ACM (2017)
87. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701 (2016), http://eprint.iacr.org/2016/701, accessed: 2017-08-07
88. Gruss, D., Lettner, J., Schuster, F., Ohrimenko, O., Haller, I., Costa, M.: Strong and efficient cache side-channel protection using hardware transactional memory. In: USENIX Security Symposium. pp. 217–233 (2017)
89. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Off the chain transactions. Cryptology ePrint Archive, Report 2019/360 (2019), https://eprint.iacr.org/2019/360.pdf, https://eprint.iacr.org/2019/360
90. Han, R., Lin, H., Yu, J.: On the optionality and fairness of atomic swaps. Cryptology ePrint Archive, Report 2019/896 (2019), https://eprint.iacr.org/2019/896
91. Harz, D., Boman, M.: The scalability of trustless trust. arXiv:1801.09535 (2018), https://arxiv.org/pdf/1801.09535.pdf, accessed:2018-01-31
92. Harz, D., Gudgeon, L., Gervais, A., Knottenbelt, W.J.: Balance: Dynamic adjustment of cryptocurrency deposits. Cryptology ePrint Archive, Report 2019/675 (2019), https://eprint.iacr.org/2019/675.pdf, https://eprint.iacr.org/2019/675

93. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub (2016), https://eprint.iacr.org/2016/575.pdf, accessed: 2017-09-29
94. Heilman, E., Lipmann, S., Goldberg, S.: The arwen trading protocols. Whitepaper, https://www.arwen.io/whitepaper.pdf
95. Herlihy, M.: Atomic cross-chain swaps. arXiv:1801.09515 (2018), https://arxiv.org/pdf/1801.09515.pdf, accessed:2018-01-31
96. Herlihy, M., Liskov, B., Shrira, L.: Cross-chain deals and adversarial commerce. arXiv preprint arXiv:1905.09743 (2019)
97. Hinteregger, A., Haslhofer, B.: An empirical analysis of monero cross-chain traceability. arXiv preprint arXiv:1812.02808 (2018)
98. Hosp, D., Hoenisch, T., Kittiwongsunthorn, P., et al.: Comit-cryptographically-secure off-chain multi-asset instant transaction network. arXiv preprint arXiv:1810.02174 (2018)
99. Intel Corp.: Software Guard Extensions Programming Reference, Ref. 329298-002US. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf (2014), https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf
100. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**(1), 36–63 (2001)
101. Johnson, S., Robinson, P., Brainard, J.: Sidechains and interoperability. arXiv preprint arXiv:1903.04077 (2019)
102. Jones, J., abitmore: Optional htlc preimage length and hash160 addition. BSIP 64, blog post, https://github.com/bitshares/bsips/issues/163
103. Jourenko, M., Kurazumi, K., Larangeira, M., Tanaka, K.: Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Report 2019/352 (2019), https://eprint.iacr.org/2019/352.pdf, https://eprint.iacr.org/2019/352
104. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gaži, P., Meiklejohn, S., Weippl, E.: Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. Cryptology ePrint Archive, Report 2019/775 (2019), https://eprint.iacr.org/2019/775.pdf, https://eprint.iacr.org/2019/775
105. Judmayer, A., Zamyatin, A., Stifter, N., Voyiatzis, A.G., Weippl, E.: Merged mining: Curse or cure? In: CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology (Sep 2017), https://eprint.iacr.org/2017/791.pdf
106. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: Proceedings of the 27th USENIX Conference on Security Symposium. pp. 1353–1370. USENIX Association (2018)
107. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: A responsive lightweight watchtower (2019)
108. Kiayias, A., Lamprou, N., Stouka, A.P.: Proofs of proofs of work with sublinear complexity. In: International Conference on Financial Cryptography and Data Security. pp. 61–78. Springer, Springer (2016)
109. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963 (2017), https://eprint.iacr.org/2017/963.pdf, accessed:2017-10-03
110. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol (2016), https://pdfs.semanticscholar.org/a583/3270b14e251f0b16d86438d04652b1b8d7f3.pdf, accessed: 2018-08-19
111. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 705–734. Springer (2016), https://eprint.iacr.org/2015/574.pdf
112. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: International Conference on Financial Cryptography and Data Security. Springer (2018)
113. Kogias, E.K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium (USENIX Security 16). USENIX Association, Austin, TX (Aug 2016), http://arxiv.org/pdf/1602.06997.pdf
114. Kokoris-Kogias, E.: Robust and scalable consensus for sharded distributed ledgers. Tech. rep., Cryptology ePrint Archive, Report 2019/676 (2019)
115. Kokoris-Kogias, E., Alp, E.C., Siby, S.D., Gailly, N., Gasser, L., Jovanovic, P., Syta, E., Ford, B.: Calypso: Auditable sharing of private data over blockchains. Tech. rep., Cryptology ePrint Archive, Report 2018/209 (2018)
116. Kokoris-Kogias, E., Alp, E.C., Siby, S.D., Gailly, N., Gasser, L., Jovanovic, P., Syta, E., Ford, B.: Verifiable management of private data under byzantine failures. Cryptology ePrint Archive, Report 2018/209 (2018), https://eprint.iacr.org/2018/209.pdf, https://eprint.iacr.org/2018/209

117. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 583–598. IEEE (2018)
118. Kokoris-Kogias, L., Gasser, L., Khoffi, I., Jovanovic, P., Gailly, N., Ford, B.: Managing identities using blockchains and CoSi. In: 9th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2016) (2016)
119. Kumaresan, R., Bentov, I.: Amortizing secure computation with penalties. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 418–429. ACM (2016)
120. Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. Computer Networks **56**(1), 50–63 (2012)
121. Kwon, J., Buchman, E.: Cosmos: A network of distributed ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md (2015)
122. Kwon, Y., Kim, H., Shin, J., Kim, Y.: Bitcoin vs. bitcoin cash: Coexistence or downfall of bitcoin cash? arXiv:1902.11064 (2019), https://arxiv.org/pdf/1902.11064.pdf
123. Lamport, L.: A simple approach to specifying concurrent systems. Communications of the ACM **32**(1), 32–45 (1989)
124. Lerner, S.D.: Rootstock: Bitcoin powered smart contracts. https://docs.rsk.co/RSK_White_Paper-Overview.pdf (2015)
125. Lerner, S.: Drivechains, sidechains and hybrid 2-way peg designs. Tech. rep., Tech. Rep. [Online] (2018), https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf
126. Liu, Z., Xiang, Y., Shi, J., Gao, P., Wang, H., Xiao, X., Hu, Y.C.: Hyperservice: Interoperability and programmability across heterogeneous blockchains. arXiv preprint arXiv:1908.09343 (2019)
127. Luu, L., Buenz, B., Zamani, M.: Flyclient super light client for cryptocurrencies https://stanford2017.scalingbitcoin.org/files/Day1/flyclientscalingbitcoin.pptx.pdf, accessed 2018-04-17
128. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks (2017), https://www.cs.purdue.edu/homes/pmorenos/public/paychannels.pdf, accessed: 2017-06-29
129. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Multi-hop locks for secure, privacy-preserving and interoperable payment-channel networks. Cryptology ePrint Archive, Report 2018/472 (2018)
130. McCorry, P., Bakshi, S., Bentov, I., Miller, A., Meiklejohn, S.: Pisa: Arbitration outsourcing for state channels. IACR Cryptology ePrint Archive **2018**, 582 (2018)
131. McCorry, P., Heilman, E., Miller, A.: Atomically trading with roger: Gambling on the success of a hardfork. In: CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology (Sep 2017), http://homepages.cs.ncl.ac.uk/patrick.mc-corry/atomically-trading-roger.pdf
132. McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), http://fc18.ifca.ai/bitcoin/papers/bitcoin18-final14.pdf
133. Meckler, I., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf (2018)
134. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 369–378. Springer (1987)
135. Meshkov, D., Chepurnoy, A., Jansen, M.: Revisiting difficulty control for blockchain systems. Cryptology ePrint Archive, Report 2017/731 (2017), http://eprint.iacr.org/2017/731.pdf, accessed: 2017-08-03
136. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: Proceedings of the twenty-second annual symposium on Principles of distributed computing. pp. 12–19. ACM (2003)
137. Micali, S.: Algorand: The efficient and democratic ledger (2016), https://arxiv.org/pdf/1607.01341.pdf, accessed: 2017-02-09
138. Miller, A.: The high-value-hash highway, bitcoin forum post (2012), https://bitcointalk.org/index.php?topic=98986.0
139. Miraz, M., Donald, D.C.: Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. Annals of Emerging Technologies in Computing (AETiC) Vol **3** (2019)
140. Moreno-Sanchez, P., Randomrun, Le, D.V., Noether, S., Goodell, B., Kate, A.: Dlsag: Non-interactive refund transactions for interoperable payment channels in monero. Cryptology ePrint Archive, Report 2019/595 (2019), https://eprint.iacr.org/2019/595
141. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (Dec 2008), https://bitcoin.org/bitcoin.pdf, accessed: 2015-07-01
142. Nikitin, K., Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Gasser, L., Khoffi, I., Cappos, J., Ford, B.: CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds

143. Oleksenko, O., Trach, B., Krahn, R., Silberstein, M., Fetzer, C.: Varys: Protecting {SGX} enclaves from practical side-channel attacks. In: 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18). pp. 227–240 (2018)
144. Pagnia, H., Gärtner, F.C.: On the impossibility of fair exchange without a trusted third party. Tech. rep., Technical Report TUD-BS-1999-02, Darmstadt University of Technology ... (1999)
145. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks (2016), http://eprint.iacr.org/2016/454.pdf, accessed: 2016-08-01
146. Pass, R., Shi, E.: Hybrid consensus: Scalable permissionless consensus (Sep 2016), https://eprint.iacr.org/2016/917.pdf, accessed: 2016-10-17
147. Paul Sztorc, CryptAxe (Pseudonym), C.S.: Drivechains BIP2: blind-merged-mining. https://github.com/drivechain-project/docs/blob/master/bip1-hashrate-escrow.md, accessed: 2018-05-21
148. Paul Sztorc, C.: Drivechains BIP1: hashrate-escrow. https://github.com/drivechain-project/docs/blob/master/bip1-hashrate-escrow.md, accessed: 2018-05-21
149. Poelstra, A.: Scriptless scripts. Presentation slides, https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf
150. Poon, J., Dryja, T.: The bitcoin lightning network (2016), https://lightning.network/lightning-network-paper.pdf, accessed: 2016-07-07
151. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
152. Robinson, P.: The merits of using ethereum mainnet as a coordination blockchain for ethereum private sidechains (2019), https://arxiv.org/pdf/1906.04421.pdf
153. Rubin, J., Naik, M., Subramanian, N.: Merkelized abstract syntax trees. http://www.mit.edu/~jlrubin/public/pdfs/858report.pdf (2014)
154. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in bitcoin (2015), http://arxiv.org/pdf/1507.06183.pdf, accessed: 2016-08-22
155. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Practical continuous distributed randomness. Cryptology ePrint Archive, Report 2018/319 (2018), https://eprint.iacr.org/2018/319.pdf
156. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
157. Sergey, I., Kumar, A., Hobor, A.: Scilla: a smart contract intermediate-level language. arXiv:1801.00687 (2018), https://arxiv.org/pdf/1801.00687.pdf, accessed:2018-01-08
158. Siris, V.A., Dimopoulos, D., Fotiou, N., Voulgaris, S., Polyzos, G.C.: Interledger smart contracts for decentralized authorization to constrained things (2019), https://arxiv.org/pdf/1905.01671.pdf
159. Sompolinsky, Y., Zohar, A.: Bitcoin's security model revisited (2016), http://arxiv.org/pdf/1605.09193.pdf, accessed: 2016-07-04
160. Sonnino, A., Bano, S., Al-Bassam, M., Danezis, G.: Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. arXiv preprint arXiv:1901.11218 (2019)
161. Spoke, M., Nuco Engineering Team: Aion: The third-generation blockchain network. https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf, accessed 2018-04-17
162. Stewart, I.: Proof of burn (2012), https://en.bitcoin.it/wiki/Proof_of_burn, accessed: 2017-05-10
163. Stifter, N., Schindler, P., Judmayer, A., Zamyatin, A., Kern, A., Weippl, E.: Echoes of the past: Recovering blockchain metrics from merged mining. In: Proceedings of the 23nd International Conference on Financial Cryptography and Data Security (FC). Springer (2019), https://fc19.ifca.ai/preproceedings/41-preproceedings.pdf
164. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 444–460. Ieee (2017)
165. Sztorc, P.: Blind merged mining. http://www.truthcoin.info/blog/blind-merged-mining/, accessed 2018-04-17
166. Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^{2}l: Anonymous atomic locks for scalability and interoperability in payment channel hubs. Cryptology ePrint Archive, Report 2019/589 (2019), https://eprint.iacr.org/2019/589
167. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains (March 2017), https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf, accessed:2017-10-06
168. Teutsch, J., Straka, M., Boneh, D.: Retrofitting a two-way peg between blockchains. Tech. rep. (2018), https://people.cs.uchicago.edu/~teutsch/papers/dogethereum.pdf
169. Thomas, S., Schwartz, E.: A protocol for interledger payments. URL https://interledger. org/interledger. pdf (2015)

170. Verdian, G., Tasca, P., Paterson, C., Mondelli, G.: Quant overledger whitepaper. https://www.quant.network/wp-content/uploads/2018/09/Quant_Overledger_Whitepaper-Sep.pdf (2018)
171. Vukolic, M.: Eventually returning to strong consistency (2016), https://pdfs.semanticscholar.org/a6a1/b70305b27c556aac779fb65429db9c2e1ef2.pdf, accessed: 2016-08-10
172. Wood, G.: Polkadot: Vision for a heterogeneous multi-chain framework. White Paper (2015)
173. Xu, Y., Cui, W., Peinado, M.: Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 640–656. IEEE (2015)
174. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167. IEEE (1986)
175. Yousaf, H., Kappos, G., Meiklejohn, S.: Tracing transactions across cryptocurrency ledgers. In: 28th {USENIX} Security Symposium ({USENIX} Security 19). pp. 837–850 (2019)
176. Zakhary, V., Agrawal, D., Abbadi, A.E.: Atomic commitment across blockchains (2019), https://arxiv.org/pdf/1905.02847.pdf
177. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: A fast blockchain protocol via full sharding. Cryptology ePrint Archive, Report 2018/460 (2018), https://eprint.iacr.org/2018/460.pdf
178. Zamyatin, A.: Merged Mining: Analysis of Effects and Implications. Master's thesis, Vienna University of Technology (2016)
179. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.: Xclaim: Trustless, interoperable, cryptocurrency-backed assets. IEEE Security and Privacy. IEEE (2019)
180. Zamyatin, A., Perez-Hernandez, D., Harz, D.: Snark-relay: Towards bitcoin transaction inclusion proofs via zksnark (2019), https://github.com/dec3ntral/snark-relay
181. Zamyatin, A., Stifter, N., Judmayer, A., Schindler, P., Weippl, E., Knottebelt, W.J.: (Short Paper) A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In: 5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer (2018), https://eprint.iacr.org/2018/087.pdf
182. Zindros, D.: Summa proofs are not composable (2019), https://medium.com/@dionyziz/summa-proofs-are-not-composable-57b87825f428