# Consensus for Crosschain Communications

**Preprint** · April 2020

CITATIONS
0

READS
1,444

**1 author:**

Peter Robinson
ConsenSys
**29** PUBLICATIONS **114** CITATIONS

SEE PROFILE

# Consensus for Crosschain Communications

Peter Robinson
Protocol Engineering Group and Systems (PegaSys), ConsenSys
School of Information Technology and Electrical Engineering, University of Queensland, Australia
peter.robinson@consensys.net



***Abstract*—Crosschain communications allows information to be communicated between blockchains. Consensus, in the context of crosschain communications, relates to how participants are able to agree on the state of one blockchain and communicate that information to another blockchain such that the information is trusted. This chapter reviews the usage scenarios of crosschain communications: value transfer and atomic swaps, reading, writing, and state pinning. It analyses what attributes must be maintained for permissionless and permissioned blockchains using a crosschain consensus mechanism such that the properties of each blockchain are not compromised. Finally, this chapter reviews the main categories of crosschain communications techniques: Hash Time Locked Contracts, block header relaying (BTC Relay, Pegged Sidechains and Ion), relay chains (Polkadot and Cosmos), and coordination chains with threshold schemes (Atomic Crosschain Transaction).**

## I. Introduction

Crosschain Communications [1] refers to the transferring of information between one or more blockchains. Crosschain Communications is motivated by two requirements common in distributed systems: accessing data and accessing functionality which is available in other systems. The first requirement, accessing data in other systems, has been previously achieved by use of distributed query languages, for example SPARQL Federated Query 1.1 [2] and the Resource Description Framework 1.1 [3]. The second requirement, accessing functionality in other systems, has been achieved by use of Remote Procedure Calls (RPC) [4].

Whereas previous distributed systems have operated with implicit trust, blockchain systems operate in partially trusted or untrusted environments. That is, consumers of distributed query language results issue queries to single servers and implicitly trust the results. The results are trusted by virtue of the fact that the results were returned by the server. Similarly, entities issuing remote procedure calls assume that if the system the call is issued to indicates the call has executed, then it has. Additionally, systems that execute remote procedure calls often execute the calls based on little or no authentication.

Blockchain systems are designed to be Byzantine fault tolerant. This means that the system can tolerate node failures, network failures, and malicious actors. Rather than relying on a single entity, blockchain system nodes come to consensus on the validity of proposed transactions. In the context of Crosschain Communications, this means that cross-blockchain systems need to be Byzantine fault tolerant, and not rely on single parties for trust.

The properties of *liveness* and *safety* are as core to cross-chain consensus as they are to other consensus protocols.

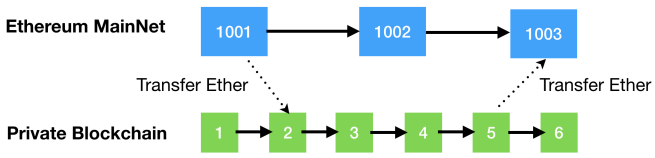



Fig. 1. Transferring Ether between Ethereum MainNet and a Private Blockchain.

Communications must only result in known outcomes and the system must be able to continue to operate, regardless of system failures, network failures or active adversaries.

The remainder of this chapter is divided into four sections. Section II *Scenarios* describes the usage scenarios that crosschain communications is used in. Section. III *Required Attributes* explains the core aspects of Permissionless and Permissioned Blockchains that need to be maintained for the Crosschain Communications technique to be useful in those environments. The *Techniques* section, Section IV, lists the main crosschain communications methodologies and explains how implementations of those technologies work, which scenarios they are able to cater for, whether they meet the requirements of Permissionless or Permissioned Blockchains, and analyses the *liveness* and *safety* properties of each technique. The chapter closes with a comparison of the Crosschain Communications techniques and their related consensus properties in Section V.

## II. Scenarios

At a high level Crosschain Communications can be viewed as reading and writing between blockchains. This section reviews the scenarios in which this reading and writing occurs.

### A. Value Transfer and Atomic Swaps

Users may have value on one blockchain and wish to move the value to another blockchain. For example, as shown in Fig. 1, a user may wish to move Ether from Ethereum MainNet to a private Ethereum blockchain, and then at a later point move the Ether back to Ethereum MainNet. While the Ether is on the Private Blockchain the Ether should not be accessible on Ethereum MainNet. Similarly, once the Ether has been transferred to Ethereum MainNet the Ether should not be accessible on the Private Blockchain.

Value transfer could be of the native currency of a blockchain such as Ether for Ethereum MainNet or Bitcoins

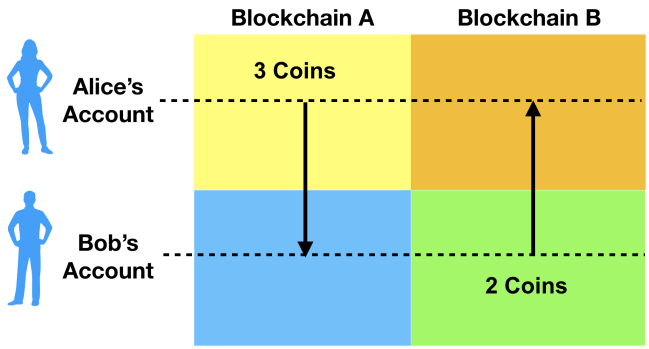



Fig. 2. Atomic Swap between two blockchains.

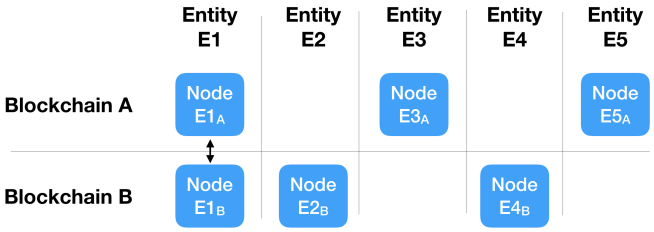



Fig. 3. Reading Values Across Blockchains.

for the Bitcoin blockchain. Alternatively, the value transfer could be for fungible tokens such as ERC 20 tokens [5], [6] or for non-fungible tokens such as ERC 721 tokens [7] which are represented as allocations within Ethereum smart contracts.

A common way to effect a value transfer is to do an Atomic Swap. In Fig. 2 Alice and Bob have accounts on both Blockchain A and B. Alice has three coins on Blockchain A. Bob has two coins on Blockchain B. Alice is prepared to swap her coins on Blockchain A for Bob's coins on Blockchain B. In an Atomic Swap, the coins are switched between Alice and Bob simultaneously. Importantly, if the transfer fails on either Blockchain A or B, then the transfer on the other blockchain fails too.

Value transfers between blockchains are typically Atomic Swaps in which the value on the first blockchain is moved such that it is inaccessible. In this situation, the value on the second blockchain needs to only be created if the value on the first blockchain can be proven to be inaccessible.

### B. Function Calls that Return Values (Reading Data)

Blockchain systems allow the reading of values from the distributed ledger at a certain block height from a blockchain node's local copy of the distributed ledger. This functionality is typically provided by functions that process the data in the ledger without updating it, and then return a result. As all of the computation occurs on the one node and the node is implicitly trusted by the application calling it, no consensus between nodes is required prior to returning the value. The nodes return values upon which the nodes in the blockchain have previously come to consensus on.

In a crosschain scenario, a node on a blockchain must convince other nodes on the blockchain that the value from another blockchain is correct. For example, consider the two blockchains in Fig 3. Imagine that there is an application that executes a transaction on Blockchain A which includes a function call to return a value from Blockchain B. The application starts the transaction on Node $E1_A$. Node $E1_A$ communicates with Node $E1_B$, which may also communicate with nodes on Blockchain B, to return a value. Node $E1_A$ must then convince nodes Node $E3_A$ and Node $E5_A$ that the value returned from Blockchain B can be trusted and used as an input to the transaction to be executed on Blockchain A.

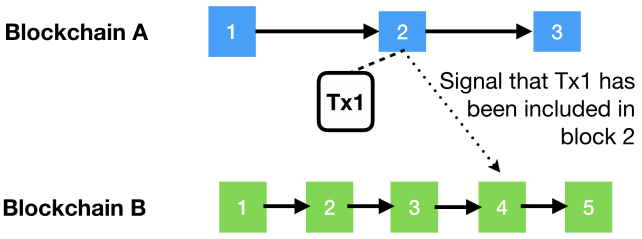



Fig. 4. Transaction Signalling.

Reading values from one blockchain to another blockchain may require both consensus of nodes of the blockchain the value is being read from and consensus on the blockchain using the value read, for example Blockchain A.

A scenario similar to reading values across blockchains, but with reduced requirements is *Transaction Signalling*. Users may wish to indicate that a transaction has occurred on a blockchain to an application on another blockchain. Fig 4 shows two blockchains. Transaction Tx1 is included in block number two on Blockchain A. A user may wish to signal to a smart contract on Blockchain B that the transaction Tx1 has occurred. They may use this as a way of validating that some action should occur on Blockchain B, as a result of the transaction Tx1 occurring on Blockchain A. Users of Blockchain B need to have some degree of certainty that the signalling can be trusted.

### C. Transaction Function Calls (Writing Data)

Blockchain systems allow distributed ledger values to be updated via transactions. Applications submit signed transactions to a node in the blockchain. Nodes gossip the transaction to other nodes and may propose a block containing the transaction. Nodes on the blockchain form consensus on which block should be added to the end of the blockchain. In this way, information is written to a blockchain system's distributed ledger.

In a crosschain system, an application may wish to submit a transaction which calls out to another blockchain. For example, in Fig 3 an application could submit a transaction to Node $E1_A$. The node could then submit the nested transaction to Node $E1_B$. In this scenario, some form of consensus is required for Blockchain A to be sure that the nested transaction has been submitted to Blockchain B.

Atomic crosschain transactions are transactions in which there is certainty that the updates across blockchains occurs together. That is, in Fig 3, the transactions and associated

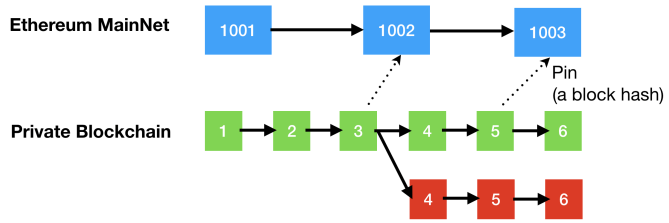



Fig. 5. Pinning state of private blockchain onto Ethereum MainNet.

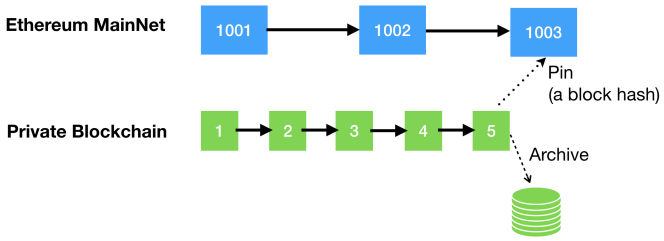



Fig. 6. Pinning state of private blockchain onto Ethereum MainNet.

updates are either committed on both blockchains or are discarded on both blockchains. In this way, the data across blockchains is guaranteed to have a consistent state.

### D. State Pinning

State Pinning [8] is defined as including the state of one blockchain in another blockchain. For example, in Fig. 5 the Block Hash of a Private Blockchain is put into a Smart Contract on Ethereum MainNet. As the Block Hash from the Private Blockchain is included in Ethereum MainNet at a particular block number, it indicates that the state of the Private Blockchain can be represented by that Block Hash at that time. The Block Hash of a block is known as a "Pin".

A majority of participants of the Private Blockchain could collude to alter the historical state of the chain [9]. For example, in Fig. 5, the colluding participants could produce new blocks 4, 5, and 6 on the Private Blockchain. State Pinning allows minority participants of the chain to prove to governmental regulators and others that the state of the chain has been altered, by showing that the correct state matches the pinned Block Hash. That is, the minority participants could demonstrate the valid state of the blockchain at block 5 and the Pin which has been included in Ethereum MainNet block 1003.

When the state of a private blockchain is pinned it is important to maintain the privacy of the blockchain by not revealing the participants of the blockchain, "Participant Privacy", or the blockchain block or transaction rate, "Block Transaction Rate Privacy". Not disclosing the transaction rate of a private blockchain is important as attackers may be able to infer activity based on this. "Anonymous State Pinning" hides the transaction rate of the pinned private blockchain and hides the identities of the participants, except in exceptional circumstances.

### E. Final State Pinning

A specialised use of Pinning is to store the final state of a Private Blockchain prior to the blockchain being archived. Pinning the final state of a blockchain prior to archiving allows the blockchain to be reinstated later if needed at a state which can be verified. For example, in Fig. 6, the Private Blockchain's late block, block 5, is pinned to Ethereum MainNet prior to the blockchain being archived. If at a later date the Private Blockchain needs to be reinstated, the reinstated state can be shown to be correct by comparing the block hash of the reinstated state with the pinned value.

## III. Required Attributes

Different types of blockchains and blockchain deployments have different requirements for a crosschain communications system. That is, in order to meet the goals of the blockchain deployment, the crosschain communications system may have specific requirements or limitations to meet such that they do not compromise the goals of the blockchain system that they are part of.

### A. Permissionless Blockchains

Permissionless Blockchains are blockchains that allow any node to join the network of nodes. Any account can submit a transaction on any node to the blockchain. Permissionless Blockchains are commonly called *Public Blockchains* or *Public Permissionless Blockchains*. Examples of Public Permissionless Blockchains are Bitcoin and Ethereum MainNet.

### B. Permissioned Blockchains

Permissioned Blockchains typically restrict the nodes that can join the network to only authorised nodes. Additionally, Permissioned Blockchains may restrict which accounts are authorised to submit transactions. Permissioned Blockchains are commonly called *Private Permissioned Blockchains*, *Private Blockchains* or *Consortium Blockchains*. Examples of Permissioned Blockchains are blockchains that can be established by Enterprise Ethereum[10] platforms such at Hyperledger Besu or Quorum, or other private blockchains platforms such Hyperledger Fabric.

Permissioned Blockchains may allow any node to join the network, but restrict the nodes that transactions can be submitted on. This type of network is typically known as a *Public Permissioned Blockchain*.

### C. Blockchain Attributes

Permissioned Blockchains[10] need to keep the list of participating nodes private. They need to keep the contents of transactions confidential. They need to keep the rate of transactions private.

Permissionless Blockchains do not have any central control or small number of controlling nodes. Permissioned Blockchains are often between a small number of participants, and as such, by virtue of the small number of participants have some level of centralisation. Some Permissioned Blockchains, such as Hyperledger Fabric and Corda have centralisation points [11][12][13]. Hence, crosschain systems typically need

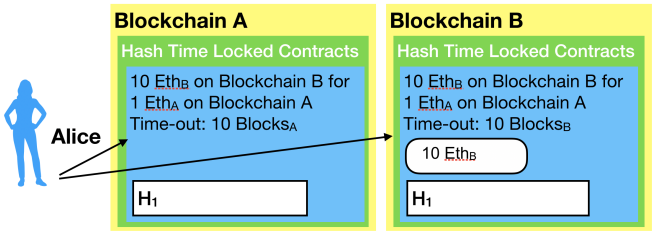



Fig. 7. Hash Time Lock Contract Set-up.

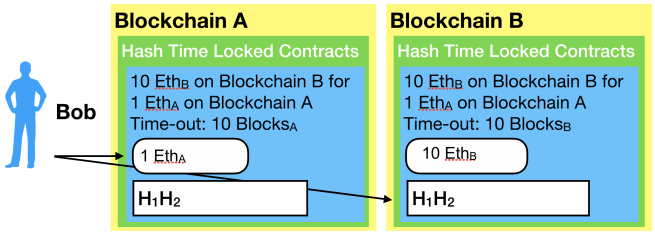



Fig. 8. Hash Time Lock Contract Accept Offer.

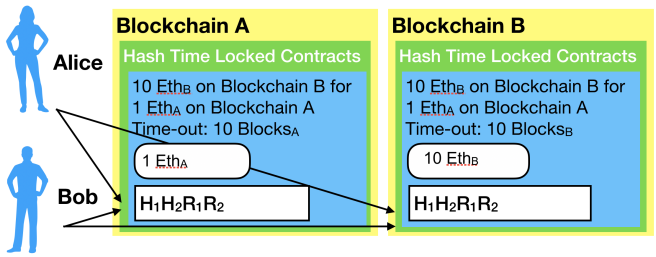



Fig. 9. Hash Time Lock Contract Finalization.

to have no centralisation points if they are to be used with Permissionless Blockchains, whereas some level of centralisation may be acceptable for Permissioned Blockchains.

Crosschain consensus protocols, similar to other consensus protocols, need a methodology to encourage good behaviour. *Bad behaviour* could be due to malicious behaviour, but could also be due to network outages, system failures, mis-configuration, and software defects. Permissionless Blockchains rely on crypto-economic enforcement, such as charging transaction fees, to discourage bad behaviour, whereas Permissioned Blockchains use external enforcement such as taking legal action in a court of law.

## IV. Techniques

### A. Hashed Timelock Contracts

Hashed Timelock Contracts (HTLC) [14] have been put forward as a mechanism for trust-less inter-chain value transfer and payment channel value transfer. This technique allows value on one blockchain to be swapped for value on another blockchain. For example in Fig. 7, Alice wishes to swap ten Ether on Blockchain B for one Ether on Blockchain A.

Alice deploys a Smart Contract to affect the value swap on the two blockchains. She sets-up the deal in the Smart Contracts reflecting the offer. She deposits ten Blockchain B Ether into the contract on Blockchain B. She generates a random number $R_1$ and calculates a *commitment* value $H_1$ using Equation 1. She submits the commitment value $H_1$ to both contracts. She also submits a time-out either in number of blocks or as wall clock time.

$$H_1 = MessageDigest(R_1) \tag{1}$$

Anyone may now accept the offer. In Fig. 8, Bob accepts the offer. Bob deposits one Ether on Blockchain A. He generates a random number $R_2$ and calculates a *commitment* value $H_2$ using Equ. 2. He submits the commitment value $H_2$ to both contracts.

$$H_2 = MessageDigest(R_2) \tag{2}$$

As soon as Bob submits the commitment values $H_2$, both Alice's and Bob's Ether is locked in the contracts in escrow. To withdraw funds, Alice and Bob need to submit the random values $R_1$ and $R_2$, as shown in Fig. 9. That is, Alice can withdraw the one Ether on Blockchain A if the Smart Contract on Blockchain A contains $H_1$, $H_2$, $R_1$ and $R_2$, and equations Equ. 1 and Equ. 2 hold for the values. Similarly, Bob can withdraw the ten Ether on Blockchain B if the Smart Contract on Blockchain B contains $H_1$, $H_2$, $R_1$ and $R_2$, and equations Equ. 1 and Equ. 2 hold for the values.

Alice and Bob can withdraw the Ether they have deposited if the time-out expires and the correct random values $R_1$ and $R_2$ have not been submitted to the contract they deposited their Ether into. If the time-out is specified in blocks, one should consider that different blockchains generate blocks with different block periods. Even if the target block period on the two blockchains is the same, the time-out on the two blockchains can not be precisely the same as block generation is not synchronised across blockchains. As such, blocks which reflect the time-out block number on each blockchain will be produced at different times.

If the blockchains both support the ability of checking wall clock time, then wall clock time could be used. For blockchains that support wall clock time, the time is likely to be the time at which the block containing the transaction accessing the time value was generated. In Ethereum, the node mining the block chooses the block's timestamp. Other nodes in the network will accept the block, as long as the block timestamp is within fifteen seconds of the current time. As such, miners in Ethereum and other similar blockchains have some limited influence as to which block a Hash Timelock Contract expires in.

The system can be subject to misuse. Bob could submit his random value, $R_2$ to both contracts. Alice could wait until the block is about to be generated which would expire the time-out on Blockchain A. She could submit her value $R_1$ and withdraw the Ether on Blockchain A. Bob may not have time to fetch the value $R_1$ and submit it to the contract on Blockchain B. In this case Alice could withdraw her Ether on Blockchain B as the timeout would have expired and her Ether would no longer be held in escrow.

The system could be enhanced such that Alice has a longer time-out to withdraw funds on Blockchain A and Bob has a longer time-out on Blockchain B. For example, the Alice could have a time-out of five minutes on Blockchain B and a one hour on Blockchain A, whereas Bob could have a time-out of five minutes on Blockchain A and a one hour time-out on Blockchain B. In this scenario, if Alice submitted her value $R_1$ to the contract on Blockchain A so that she could withdraw her funds just before the five minute time-out, Bob would have an additional fifty-five minutes to submit the $R_1$ value to Blockchain B to withdraw his funds.

Hash Timelock Contracts have the *safety* issue that parties can loose their funds if they go offline for longer that the time-out period prior to withdrawing their funds and after submitting their secret. Another abuse of the system (though not a *safety* issue) is that a party could only execute the agreement, submitting their R value if blockchain value fluctuations are in their favour [15]. That is, Alice could choose to not submit $R_1$ if the relative value of Ether on Blockchain A relative to Blockchain B drops and submit it if the relative value increases.

Hash Timelock Contracts do not suffer from *liveness* issues. If funds are not transferred prior to the time-out, then the participants are able to withdraw the funds. Even in the case of the *safety* issue described above, where a participant goes offline, the contract is not blocked from use once the time-out has expired.

As Hash Timelock Contracts allow trust-less transfer of value between blockchains, they are suitable for Permissionless Blockchains and Permissioned Blockchains. They form the basis of the value transfer mechanism in Poon and Dryja's Bitcoin Lightning Network [16], Thomas and Schwartz's Interledger Protocol [17], and the Dogecoin to Ethereum bridge [18].

### B. Transferring Block Headers

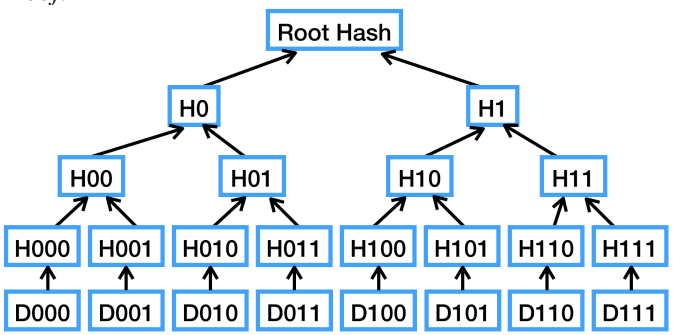
Block Headers transferred between blockchains can be used to verify that a transaction has executed on one blockchain and that some action should occur on another blockchain based on this fact. For example in Fig. 10 Alice observes blocks being produced on Blockchain A. For each block produced on Blockchain A, she submits the block header to Blockchain B by submitting a transaction to a contract. Anyone, in this example Bob in Fig. 11, can then submit a transaction to the contract on Blockchain B that includes the block number, the transaction which is said to have occurred on Blockchain A, and a Merkle Proof (see Info. IV-B) that proves that the transaction occurred on Blockchain A in a certain block. The contract calculates the hash of the transaction said to have occurred on Blockchain A, and calculates the Root Hash using the partial Merkle Tree. If the Root Hash matches the value in the block header indicated by the block number, then the transaction will be deemed to have occurred on Blockchain A in the specified block.

The *safety* of the system relies on the block headers being transferred such that they can be trusted. The *liveness* of the system depends on the block headers actually being transferred. If the entity(ies) transferring block headers stop transferring them, then no transactions can be confirmed on the second blockchain.

BTC Relay [20] uses the block header transferring technique described above to allow users of Ethereum MainNet (the production public deployment of Ethereum) to confirm Bitcoin transactions and do actions based on those transactions. A typical action is to move Ether from one account to another account based on a Bitcoin transaction transferring BTC from one account to another account. Entities called *Relayers* are

A *Merkle Tree* [19] is a tree of message digests, as shown in the figure below. Each node in the tree is a message digest value. Data D000 to D111 are separately message digested to produce H000 to H111. Nodes above the leaf layer are calculated as the message digest of the concatenation of the nodes that feed into them. For example, `H00` is calculated as `message digest(H000 + H001)`, where + indicates concatenation. The node at the head of the tree is known as the *Root Hash* or *Merkle Root*. Systems using Merkle Trees typically have Root Hashes available such that verifiers can confirm information in the Merkle Tree using the Merkle Root.

*Merkle Proofs* provide the ability to prove that a data value is part of a dataset, without having the present the entire dataset. For example, to prove that data value `D110` is a member of the dataset, the data value along with message digests `H111`, `H10`, and `H0` are presented to a verifier. The verifier calculates candidate values for `H110`, `H11`, `H1` and the `Root Hash`. The verifier can confirm the data value is part of the dataset if the candidate Root Hash matches the expected Root Hash value. The data value along with the message digests `H111`, `H10`, and `H0` are known as a *Merkle Proof*.




**Information 1:** Merkle Trees and Merkle Proofs

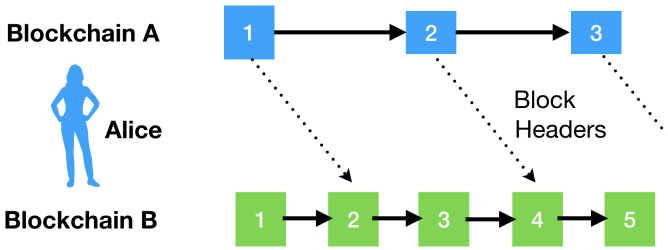



Fig. 10. Block Header Transfer

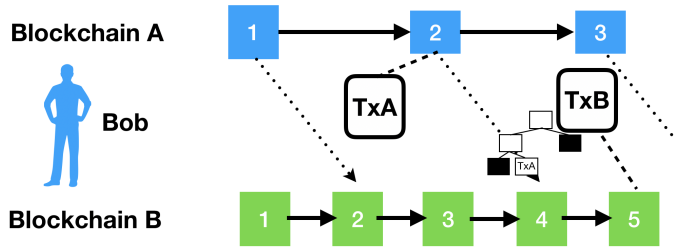



Fig. 11. Transaction Transfer

compensated for posting Bitcoin block headers to a Smart Contract on Ethereum MainNet. BTC Relay relies on Proof of Work (PoW) mining difficulty for its security. Multiple active Relay nodes must be prepared to post the block header for each block. In this way, if one Relay node posts a block header of a fork of the chain, other Relay nodes can post the block header of the longest chain. Transactions can only be validated if the block header they relate to is on the longest chain and if at least six block headers have been posted on top of the block header that the transaction relates to [21]. As attackers can not produce a longer chain than the main Bitcoin blockchain due to the mining difficulty, they are unable to confirm transactions based on a malicious fork.

Users of BTC Relay pay a fee to confirm Bitcoin transactions on Ethereum. This fee is used to compensate Relayers. However, if no users are confirming transactions on BTC Relay, then Relayers are not compensated for submitting blocks to the BTC Relay contract on Ethereum. This means that the Relayers are not incentivised to submit Bitcoin block headers. If no block headers are submitted, then transactions can not be confirmed. This *liveness* issue has occurred in the production deployments of BTC Relay as Relayers are no longer relaying block headers.

A detailed security analysis of BTC Relay was conducted in 2016 by Martin Swende [22]. This analysis identified implementation issues such as being able to avoid fees while confirming Bitcoin transactions, but no *safety* issues relating to crosschain consensus.

Pegged Sidechains [23] proposes Bitcoins be transferred between the Bitcoin blockchain and sidechains, to allow for increased transaction rate and experimentation. The solution relies on publishing a Merkle Proof that a transaction to transfer Bitcoin was included in a block in the source blockchain and publishing the block headers that were produced based on that block, in the *destination* blockchain. The authors of the paper [23] recommend users should consider providing 24 hours of block headers. If the hashing power of the source blockchain is significant, then it would be impossible for an attacker to produce forged blocks. Wood [24] contends that the sidechain hashing power is unlikely to be sufficient to ensure security. Consequently, Bitcoins can be securely transferred to the sidechain from the Bitcoin blockchain, but not back. As such, this methodology should be viewed as having *safety* issues. Given the user affecting the transfer submits their own block headers, there is no requirement to wait for or incentivise relaying nodes to transfer block headers as there was in BTC Relay. As such, this methodology does not suffer from *liveness* issues.

BTC Relay and Pegged Sidechains are not generally applicable as they rely on the source blockchain using PoW consensus algorithm, and the hashing power of the source blockchain being large enough such that attackers can not create forks and affect transfers based on the forks. Additionally, PoW is not an appropriate consensus algorithm for private blockchains as organisations do not wish to allocate resources to mining of blocks [10]. As such, they are inappropriate for Private Blockchains.

The Clearmatics Ion project uses the block header transferring technique in the context of Private Permissioned Blockchains. It provides a framework and tools to develop crosschain smart contracts so that they execute if a state transition has occurred on another blockchain [25], [26]. The system works by having a set of validators that wish to transfer block headers from one blockchain to another. At least a threshold number of the validators need to sign each block header for the block headers to be accepted by the receiving blockchain. Validators only transfer a block header once the blocks they relate to are deemed to be final (see Info. IV-B). For the scenario when the source blockchain is Ethereum, a user executes a transaction on the source blockchain that emits an event. The transaction's receipt contains the transaction hash, the block number, transaction number, and includes a list of event information. The event information is feeds into the transaction hash calculation. The user can now execute a transaction on the destination blockchain, passing in as parameters information relating to the source blockchain's transaction: the block number, the Merkle Proof showing the transaction belongs to the block, and the event information. The code on the destination blockchain executes code that verifies the source blockchain's transaction information. If that information is found to be correct, then code can use the information in the event to perform some action. Ion would suffer from *liveness* issues if not enough validators sign block headers or if the multiply signed block headers are not transferred to the destination blockchain.

BTC Relay and Pegged Sidechains leverage the consensus protocol of the source blockchain, Bitcoin's PoW, and rules around how many blocks to wait, to provide crosschain consensus. Ion forms crosschain consensus by registering a set of validators from a source blockchain on a destination blockchain and requiring a threshold subset of the validators to sign block headers for them to be accepted. The threshold and the validators themselves do not need to be related to the source blockchain's consensus protocol. Whereas the crosschain consensus algorithm for BTC Relay and Pegged Sidechains is tied to the underlying blockchain, Ion's crosschain consensus algorithm is independent of the source blockchain's consensus algorithm.

### C. Cosmos: Relay Chain with Block Header Transfer

Cosmos [29] is a multi-blockchain system in which blockchains called Zones communicate transactions via a central blockchain called a Hub, as shown in Fig. 12. The

A block is *final* if it has been added to a blockchain and can not be removed for any reason. A transaction that is included in a block is final if the block it is included in is itself final.

Consensus algorithms such as Istanbul Fault Tolerant (IBFT)[27] have *immediate finality*. In these protocols a block can not be removed once it has been added to the blockchain.

Bitcoin and Ethereum's Proof of Work (PoW) consensus algorithms offer *probabalistic finality*. In these protocols competing mining nodes may build longer (in the case of Bitcoin) or heavier (in the case of Ethereum) chains, thus causing a fork in the blockchain. If a fork occurs, blocks that no longer belong to the canonical chain are known as *stale* and the transactions in them are pushed back into the transaction pool to be added to new blocks if they do not already appear in blocks in the new canonical chain. Due to mining difficulty, the probability of a node finding a heavier chain and a block currently on the canonical chain being discarded decreases as more blocks are added to the blockchain[28].

**Information 2:** Finality

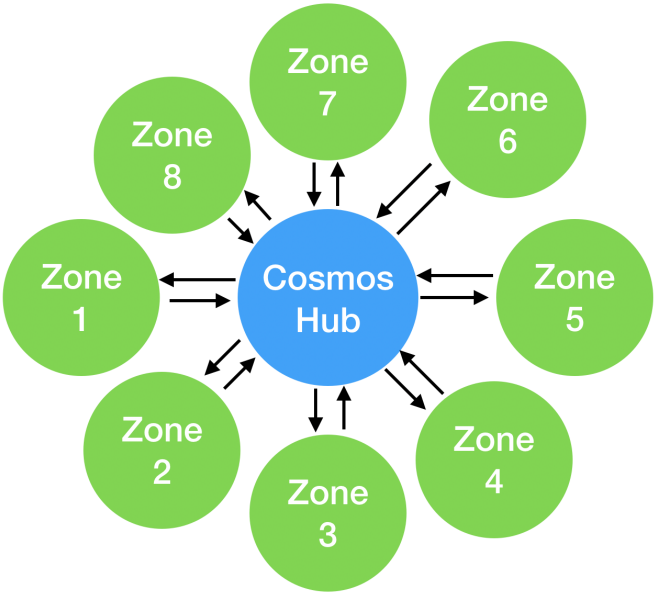



Fig. 12. Cosmos system of blockchains

Zones and the Hub typically use Tendermint [30] a type of Practical Byzantine Fault Tolerance [31][32] consensus algorithm. The system is envisaged to allow heterogeneous blockchain communications, allowing the Zone blockchains to be permissioned or permissionless, to use alternative consensus algorithms, including algorithms that offer probabilistic finality, and allowing for completely different blockchain paradigms.

Validators on each Zone blockchain must have a set of validators. The Zone blockchain validators must trust validators in the Hub blockchain and visa-versa.

Fig. 13 shows an example communication of a datagram

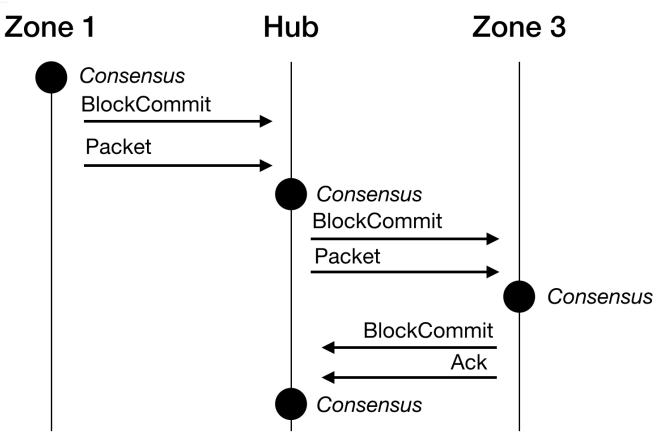



Fig. 13. Cosmos Inter-Blockchain Communications (IBC)

from Zone 1 to Zone 3 using Cosmos' Inter-Blockchain Communications (IBC) system. A transaction on the Zone 1 blockchain generates the datagram to be communicated to Zone 3 blockchain. The datagram is included in a block on Zone 1 blockchain. A *Block Commit* message containing the block header of the block containing the datagram to be communicated is signed by at least a threshold number of validators and sent to the Hub blockchain. A *Packet* message containing the datagram, and a Merkle Proof proving that the datagram relates to the block, are sent to the Hub blockchain. The Hub blockchain then includes the datagram in a block on the Hub blockchain. A node on the Hub blockchain then sends a *Block Commit* message to the Zone 3 blockchain with the Hub blockchain's block header for the block containing the datagram. This message is signed by at least a threshold number of validators on the Hub blockchain. Finally, the Hub blockchain sends the datagram along with a Merkle Proof in a *Packet* message. The validators on Zone 3 blockchain can send a *Block Commit* message and then an acknowledgement message back to the Hub blockchain. Once the Hub blockchain includes the acknowledgement in a block, validators on Zone 1 blockchain can check the acknowledgement and be sure that the transaction has been included in the Zone 3 blockchain.

The datagram sent from the Zone 1 blockchain can contain a time-out defined in terms of a block number on the Hub blockchain. If the Hub blockchain determines that a datagram has timed-out, because validators on the Hub blockchain have not seen an acknowledgement from Zone 3 blockchain, they can generate a *time-out datagram*, include it in the Hub blockchain, send a *Block Commit* and then a *Time-out* message to the Zone 1 blockchain.

Cosmos' crosschain consensus relies on the trust between the Zone and Hub blockchains, and the threshold number of validators that need to sign a message used for each of them. The messages contained in the `Block Commit` messages are trusted based on this. In turn, datagrams are shown to have been included in blocks by presenting Merkle Proofs in *Packet* messages.

The system does not provide guaranteed atomic behaviour. For example, there is no guarantee that a datagram is included

in both the Zone 1 and Zone 3 blockchains. There is also no guarantee that the Zone 3 blockchain will be willing to submit acknowledgements to the Hub blockchain. The Hub blockchain should produce a time-out datagram, though there is no certainty that this will occur.

The liveness of the system depends on the liveness of the individual Zone and Hub blockchains.

The Hub blockchain is provided rate control and protected against Denial of Service (DoS) attacks flooding the system with crosschain transactions by charging for each transaction committed to the Hub blockchain using Cosmos' digital currency, the *Atom*. This usage of the Atom could lead to issues for users who do not have adequate Atoms to pay for blocks to be included in the Hub blockchain. In particular, acknowledgements posted by blockchains receiving crosschain transactions may not be able to be posted to the Hub blockchain if accounts on the receiving blockchain do not have adequate Atoms.

The system relies on Zone blockchain validators fully trusting Hub blockchain validators to act correctly. That is, the Hub blockchain acts as a centralisation point for the entire system of blockchains.

Cosmos [29] documentation indicates that cross-zone messages can be end-to-end encrypted which would provide confidentiality. This might allow the cross-zone message system to be used between two Private Blockchain Zones.

### *D. Polkdot: Shared Consensus*

Polkadot [24], [33] proposes a multi-chain network built on Substrate consisting of *Relay Chains*, *Parachains* and *Bridges*. Relay Chains provide shared consensus for all Parachains. Parachains receive and process transactions. Bridges provide a mechanism for transactions to be routed to non-Polkadot blockchain systems. Note that, despite the name, Relay Chains do not relay messages between Parachains or Bridges.

There are two main roles that participants play in the Polkadot ecosystem: *Collator* and *Validator*. Collators collect transactions on Parachains, propose blocks and provide zero knowledge non-interactive proofs proving the transactions result in valid state changes to the Validators. Groups of Validators ratify Parachain blocks and publish them to the Parachain. The Validators seal the Parachain block headers to the Relay Chain. The Validators are randomly assigned to Parachains, with the assignment changing regularly. Validators use a Proof of Stake consensus algorithm to provide a shared consensus for all Parachains. Supportive roles are performed by *Nominators* and *Fishermen*. Nominators provide funds to Validators they trust to execute the Proof of Stake consensus. Fishermen observe the Parachains and submit fraud proofs to Validators.

Cross-Parachain transactions are identical to typical transactions from external accounts. A transaction on one Parachain results in a message being placed in an outbound queue by a Collator on that chain. A Collator for the target Parachain will gather messages destined for the Parachain and submit them to the Parachain's incoming queue. The message is then processed as a transaction on the destination Parachain. The messages are trusted by the destination Parachain as the proof that the message relates to a transaction on the originating Parachain can be submitted, and the transaction can be proven to have been included in a block.

Transactions from Polkadot to Ethereum via a Bridge are achieved by submitting transactions to a special multi-signature Ethereum contract. The signers of the multi-sig wallet are likely to be Validators. Transactions from Ethereum to Polkadot are achieved by calling into a special Ethereum contract which writes an event to the Ethereum event log. This event is interpreted as the outward bound call. As of January 2020, this technology is not documented, marked as *alpha*, and appears to be abandoned.

Despite Polkadot appearing to be a multi-blockchain system, the system is in reality a multi-shard system with shared consensus. As such, for Cross-Parachain transactions, there is no cross blockchain consensus. That is, there is no need for one blockchain to prove to another blockchain that an event occurred or a transaction has been included in a block, as all nodes in the network are using the same blockchain. This usage of a shared consensus algorithm greatly simplifies Cross-Parachain communications.

If a Parachain was to be considered a separate chain, then the requirement for randomly allocated Validators to view information on the Parachain is incompatible with the concepts of Private Blockchains which need to restrict the list of participants.

*1) Plasma and Exits:* Minimum Viable Plasma [34] builds on the concept of Plasma's [35] delegate Ethereum blockchains. Plasma chain operators create a Plasma Smart Contract on Ethereum MainNet and hold value deposited in the contract on a separate Plasma chain as Unspent Transaction Output (UTXO) [36] values in a binary Merkle tree ordered by transaction index. Transactions on the Plasma chain involve proving that an unspent output had not previously been spent. Plasma Cash [37], builds on the Minimum Viable Plasma approach to allow for the exchange of non-fungible assets. Each token has an identifier that represents the token's location in a sparse Merkle Tree. To spend a block, a proof needs to be submitted showing when the token has been used.

For each Plasma variant, information is created on the Plasma chain by a chain operators after a deposit into the Plasma Smart Contract on Ethereum MainNet. To exit the Plasma chain, the owner of a token needs to submit a proof to the Plasma Smart Contract, proving that they own the token. Another user could challenge the *exit* by submitting a fraud proof showing that a subsequent transaction occurred that changed the ownership of the token after the proof submitted by the user purporting to by token owner. The challenge period is typically at least seven days.

A drawback of Plasma is that users need to observe the Plasma Smart Contract continuously, checking for fraudulent exit proofs being submitted, to prevent tokens they rightfully own being converted into Ether. If a user is offline for the entire challenge period, then a nefarious actor could still their tokens. The proofs are likely to be large. As such, exiting and challenging an exit on Ethereum MainNet could be costly. Another major issue is that Ethereum MainNet does not have sufficient capacity to allow for a *mass exit* scenario, in which

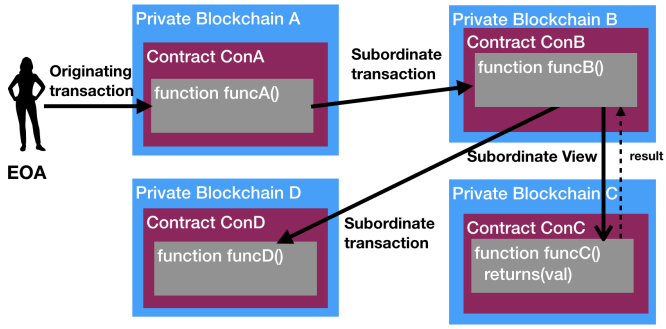



Fig. 14. Originating Transaction containing Two Nested Subordinate Transactions and a Subordinate View

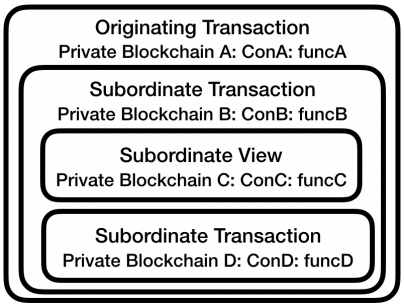



Fig. 15. Nested Transactions and Views

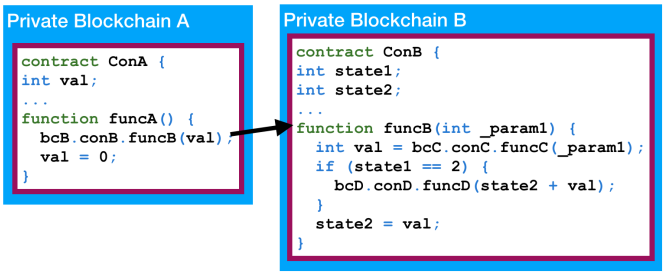



Fig. 16. Function Calls across Blockchains

all members of a Plasma chain choose to exit the chain at the same time.

From a crosschain consensus perspective, consensus is provided by users submitting exit proofs and fraud proofs. Plasma does not provide *safety* as nefarious actors could steal a users tokens if a user is offline during a challenge period or if the nefarious actor could orchestrate a mass exit, thus preventing users from submitting fraud proofs.

### *E. Atomic Crosschain Transactions: Nested Transactions with Coordination Blockchains and Threshold Signatures*

Atomic Crosschain Transactions [1] for Ethereum Private Sidechains [11] and private Ethereum blockchains allow for inter-contract and inter-blockchain function calls that are both synchronous and atomic.

*1) Nested Transactions:* Atomic Crosschain Transactions are special nested Ethereum transactions and views. Transactions are function calls that update state. Views are function calls that return a value but do not update state. Fig. 14 shows an Externally Owned Account (EOA) calling a function `funcA` in contract `ConA` on blockchain `Private Blockchain A`. This function in turn calls function `funcB`, that in turn calls functions `funcC` and `funcD`, each on separate blockchains. The transaction submitted by the EOA is called the *Originating Transaction*. The transactions that the Originating Transaction causes to be submitted are called Subordinate Transactions. Subordinate Views may also be triggered. In Fig. 14, a Subordinate View is used to call `funcC`. This function returns a value to `funcB`.

Fig. 15 shows the nested structure of the Atomic Crosschain Transaction. The EOA user first creates the signed Subordinate View for `Private Blockchain C`, contract `ConC`, function `funcC` and the signed Subordinate Transaction for `Private Blockchain D`, contract `ConD`, function `funcD`. They then create the signed Subordinate Transaction for `Private Blockchain B`, contract `ConB`, function `funcB`, and include the signed Subordinate Transaction and View. Finally, they sign the Originating Transaction for `Private Blockchain A`, contract `ConA`, function `funcA`, including the signed Subordinate Transactions and View.

*2) Actual and Expected Parameter Values:* When a function executes in the Ethereum Virtual Machine [38] function parameter values and stored state combine to form the actual values of variables during execution. For example, consider `funcB` in contract `ConB` on `Private Blockchain B` in Fig. 16. Assuming `_param1` is `1`, `state1` is `2`, `state2` is `4`, and that `funcC` returns the value `6`, then function `funcC` will be called with the parameter value `1`, and function `funcD` will be called with the parameter value `10`. To execute this as part of a Crosschain Transaction, signed Subordinate Transactions and View need to be created with the appropriate parameter values.

Execution of a transaction or view fails if the actual parameter value passed to a function does not match the value in the signed Subordinate Transaction or View. The parameter value in the signed Subordinate Transaction or View is the value the application developer expected to be passed to the function. The expected value can be determined at time of nested transaction creation using dynamic program analysis techniques. In particular, the dynamic analysis needs to consider program flow. For instance, if `state1` was `1`, then `funcD` would not be called, and no Subordinate Transaction should be included in the Crosschain Transaction for this function call.

*3) Per-Node Transaction Processing:* When the EOA submits the Originating Transaction to a node, the node processes the transaction using the algorithm shown in Listing 1. If the transaction includes any Subordinate Views, they are dispatched and their results are cached (Lines 1 to 3). The function is then executed (Lines 4 to 17). If a Subordinate Transaction function call is encountered, the node checks that the parameter values passed to the Subordinate Transaction function call match the parameter values in the signed Subordinate Transaction (Lines 6 to 8). If a Subordinate View function call is encountered, the node checks that the parameters passed

BLS Threshold Signatures [39], [40] combines the ideas of threshold cryptography [41] with Boneh-Lynn-Shacham(BLS) signatures [42], and uses a Pedersen commitment scheme [43] to ensure verifiable secret sharing. The scheme allows any `M` validator nodes of the total `N` validator nodes on a blockchain to sign messages in a distributed way such that the private key shares do not need to be assembled to create a signature. Each validator node creates a signature share by signing the message using their private key share. Any `M` of the total `N` signature shares can be combined to create a valid signature. Importantly, the signature contains no information about which nodes signed, or what the threshold number of signatures (`M`) needed to create the signature is.

**Information 3:** BLS Threshold Signatures

to the Subordinate View function call match the parameter values in the signed Subordinate View (Lines 9 and 10). The cached values of the results of the Subordinate View function calls are then returned to the executing code (Line 11). If the execution has completed without error, then each of the signed Subordinate Transactions is submitted to a node on the appropriate blockchain (Nodes 18 to 20).

Listing 1. Originating or Subordinate Transaction Processing

```
1  For All Subordinate Views {
2    Dispatch Subordinate Views & cache results
3  }
4  Trial Execution of Function Call {
5    While Executing Code {
6      If Subordinate Transaction function called {
7        check expected & actual parameters match.
8      }
9      Else If Subordinate View function is called {
10       check expected & actual parameters match
11       return cached results to code
12     }
13     Else {
14       Execute Code As Usual
15     }
16   }
17 }
18 For All Subordinate Transactions {
19   Submit Subordinate Transactions
20 }
```

*4) Blockchain Signing and Threshold Signatures:* The Atomic Crosschain Transaction system uses BLS Threshold Signatures (see Info. IV-E4) to prove that information came from a specific blockchain. For example, in Fig. 14, nodes on `Private Blockchain B` can be certain of results returned by a node on `Private Blockchain C` for the function call to `funcC`, as the results are threshold signed by the validator nodes on `Private Blockchain C`. Similarly, validator nodes on `Private Blockchain A` can be certain that validator nodes on `Private Blockchain B` have mined the Subordinate Transaction, locked contract `ConB` and are holding the updated state as a provisional update because validator nodes sign a *Subordinate Transaction Ready* message indicating that the Subordinate Transaction is ready to be committed.

*5) Multichain Nodes:* A Multichain Node is a logical grouping of one or more blockchain validator nodes, where each node is on a different blockchain. The blockchain nodes

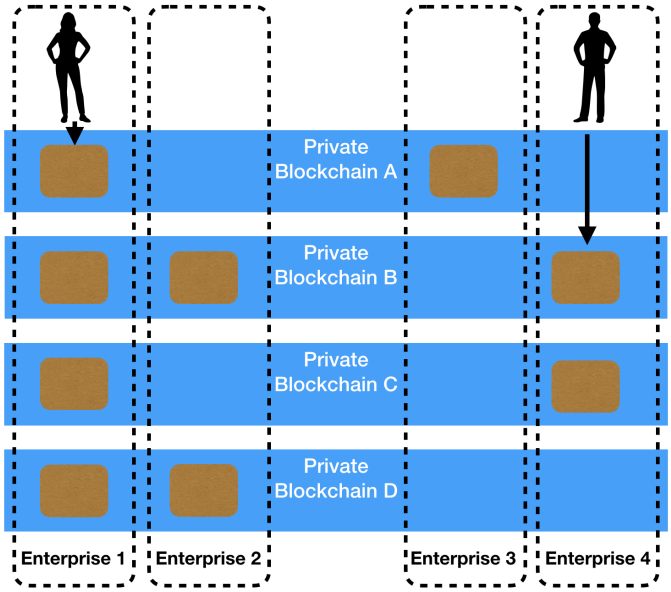



Fig. 17. Multichain Nodes

operate together to allow Crosschain Transactions. The Multichain Node on which the transaction is submitted must have Validator Nodes on all of the blockchains on which the Originating Transaction and Subordinate Transactions and Views take place.

Fig. 17 shows four enterprises that have validator nodes on `Private Blockchain A` to `Private Blockchain D`. Alice who works in Enterprise 1 can submit Atomic Crosschain Transactions that span `Private Blockchain A` to `Private Blockchain D` as Enterprise 1 has a Multichain Node that includes validator nodes on each blockchain. However, Bob who works in Enterprise 4 can only submit Atomic Crosschain Transactions that span `Private Blockchain B` and `Private Blockchain C` as Enterprise 4 only has validator nodes on `Private Blockchain B` and `Private Blockchain C`.

*6) Crosschain Coordination: Crosschain Coordination Contracts* exist on *Coordination Blockchains*. They allow validator nodes to determine whether the provisional state updates related to the Originating Transaction and Subordinate Transactions should be committed or discarded. The contract is also used to determine a common time-out for all blockchains, and as a repository of Blockchain Public Keys.

When a user creates a Crosschain Transaction, they specify the Coordination Blockchain and Crosschain Coordination Contract to be used for the transaction, and the time-out for the transaction in terms of a block number on the Coordination Blockchain. The validator node that they submit the Originating Transaction to (the *Originating Node*) works with other validator nodes on the blockchain to sign a *Crosschain Transaction Start* message. This message is submitted to the Crosschain Coordination Contract to indicate to all nodes on all blockchains that the Crosschain Transaction has commenced.

When the Originating Node has received Subordinate Transaction Ready messages for all Subordinate Transactions, it

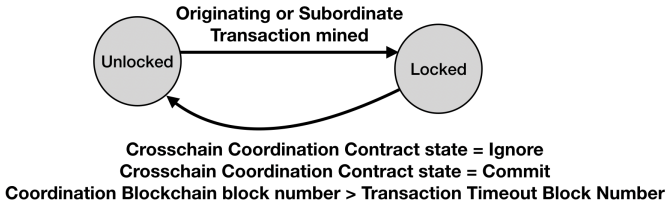



Fig. 18. Contract Locking States

works with other validator nodes to create a *Crosschain Transaction Commit* message. This message is submitted to the Crosschain Coordination Contract to indicate to all nodes on all blockchains that the Crosschain Transaction has completed and all provisional updates should be committed. If an error is detected, then a *Crosschain Transaction Ignore* message is created and submitted to the Crosschain Coordination Contract to indicate to all nodes on all blockchains that the Crosschain Transaction has failed and all provisional updates should be discarded. Similarly, if the transaction times-out, all provisional updates will be discarded.

*7) Contract Locking and Provisional State Updates:* When a contract is first deployed it is marked as a Lockable Contract or a Nonlockable Contract. A Nonlockable Contract, the default, is one which can not be locked. When a node attempts to update the state of a contract given an Originating or Subordinate Transaction, it checks whether the contract is *Lockable* and whether it is *locked*. The transaction fails if the contract is Nonlockable or if the contract is Lockable but is locked.

Careful design is needed to determine whether a contract should be Lockable or Nonlockable. Typically, an individual item, for instance a hotel room, might be modelled as a Lockable contract. The hotel contract would hold references to each hotel room contract and be NonLockable. In this way, many Atomic Crosschain Transaction can occur via the hotel contract, without encountering locked hotel room contracts.

Figure 18 shows the locking state transitions for a contract. The Crosschain Coordination Contract will be in *Started* state. The act of mining an Originating Transaction or Subordinate Transaction and including it in a blockchain locks the contract. The contract is unlocked when the Crosschain Coordination Contract is in the *Committed* or *Ignored* state, or when the block number on the Coordination Blockchain is greater than the Transaction Timeout Block Number. The Crosschain Coordination Contract will change from the *Started* state to the *Committed* state when a valid Crosschain Transaction Commit message is submitted to it, and it will change from the *Started* state to the *Ignored* state when a valid Crosschain Transaction Ignore message is submitted to it.

Ordinarily, all nodes will receive a message indicating that they should check the Crosschain Coordination Contract when the contract can be unlocked. When a node first processes a transaction, it will set a local timer which should expire when the Transaction Timeout Block Number is exceeded. If the node has not received the message by the time the local timer expires, the node checks the Crosschain Coordination Contract to see if the Transaction Timeout Block Number has been exceeded and whether the updates should be committed or ignored.

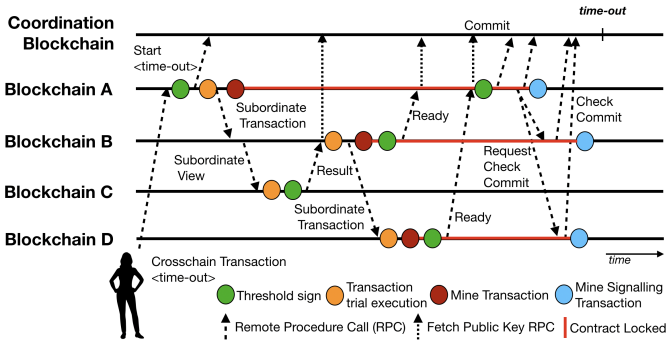



Fig. 19. Sequence Diagram

*8) Sequence Diagram:* Fig. 19 shows a sequence diagram showing an example sequence of execution, based on the calls in Fig. 14. The sequence starts with an Externally Owned Account (EOA) submitting a Crosschain Transaction to a node on Blockchain A. This nested transaction contains the Originating Transaction, and all Subordinate Transactions and Views, plus a time-out in terms of Coordination Blockchain block numbers. The node is known as the Coordinating Node for this transaction.

The nodes on Blockchain A cooperate to threshold sign the *Start* message. The signed *Start* message is submitted to the Coordination Contract on the Coordination Blockchain specified in the Originating Transaction (and all Subordinate Transactions and Views). The Coordination Contract on the Coordination Blockchain accepts the *Start* message if its signature can be verified with the public key registered for Blockchain A. The *Start* message contains the time-out block number, which the Coordinating Contract registers when it accepts the *Start* message. Once the *Start* is registered, the Crosschain Transaction is in *Start* state.

The Coordinating Node then *Trial Executes* the Originating Transaction. Assuming the code executes correctly, the Subordinate Transaction for Blockchain B is dispatched. The Originating Transaction is then submitted to Blockchain A's transaction pool. When the transaction is included in a block, the contract's updated state is stored as provisional state, and the contract state is locked.

The Subordinate Transaction for Blockchain B contains a Subordinate View for Blockchain C. The Subordinate View is dispatched to a node on Blockchain C. The node on Blockchain C completes a *Trial Execution* of the Subordinate View at a certain block number. The node then cooperates with other Blockchain C nodes to have the result of the Subordinate View at the specific block number threshold signed. The node returns the signed result to the calling node on Blockchain B.

The nodes on Blockchain B that do not contain the correct version of Blockchain C's public key fetch it from the Coordination Contract. The node *Trial Executes* the Subordinate Transaction, returning the value from the Subordinate View to the code when the function corresponding to the Subordinate View is called. Assuming the *Trial Execution* executes correctly, then the Subordinate Transaction to be executed

on Blockchain D is dispatched to a node on Blockchain D. The Subordinate Transaction for Blockchain B is submitted to Blockchain B's transaction pool, with the signed Subordinate View result from Blockchain C attached. When the transaction is included in a block, the contract's updated state is stored as provisional state, and the contract state is locked. The nodes on Blockchain B then cooperate to threshold sign a *Ready* message, indicating that the blockchain is ready to commit the Atomic Crosschain Transaction. The message is submitted to the Coordinating Node.

The Subordinate Transaction for Blockchain D is trial executed. There are no Subordinate Transactions or Views for the transaction that need to be dispatched. The transaction is mined. When the transaction has been included in a block, the updated state is stored in provisional state, the contract is locked, and the nodes on Blockchain D cooperate to threshold sign a *Ready* message, indicating that the blockchain is ready to commit the Atomic Crosschain Transaction. The message is submitted to the Coordinating Node.

When the Coordinating Node (on Blockchain A) has received *Ready* messages for Blockchain B and D, and the Originating Transaction has been mined, then the nodes on Blockchain A then cooperate to threshold sign a *Commit* message. This message is uploaded to the Coordination Contract on the Coordination Blockchain. Assuming this message is uploaded prior to the time-out, the Crosschain Transaction is then in the commit state.

The Coordinating Node requests nodes on other Blockchains commit a Signalling Transaction. When a node on a blockchain encounters a Signalling Transaction for a certain Crosschain Transaction, it checks the Coordination Contract for the Crosschain Transaction to check to see if the transaction should be committed or ignored. If the transaction is to be committed, then all contract provisional state related to the transaction is converted to normal state, unlocking the contracts. If the transaction is to be ignored, then the contract provisional state is discarded and the contract state is updated, unlocking the contracts.

*9) Failure Cases:* If an error is detected then nodes on Blockchain A can cooperate to create a threshold signed *Ignore* message which is uploaded to the Coordination Contract to indicate to all nodes that the Crosschain Transaction should be abandoned, all provisional state updates should be discarded and contracts should be unlocked. If an *Ignore* message can not be created, or if the Crosschain Transaction times-out, then the state returned by the Coordination Contract indicates the Crosschain Transaction should be abandoned.

When nodes on each blockchain first become a part of a Crosschain Transaction, they set a timer to expire when they expect the Crosschain Transaction to time-out, plus a random additional wait period. The timer is cancelled when a Signalling Transaction is received. If a node's timer expires prior to receiving a Signalling Transaction, the node checks the Coordination Contract and then creates and submits a Signalling Transaction to indicate to other nodes that the updates should be committed or ignored, and that the contract should be unlocked. The additional random wait period is used so that all nodes on the blockchain do not simultaneously send Signalling Transactions.

*10) Analysis:* Atomic Crosschain Transactions require a set of validators to threshold sign information. The threshold signing mechanism does not reveal any information about the validators who signed or the threshold number of validators that was needed to sign.

The identity of the node that submits the *Start*, *Commit*, and *Ignore* messages is exposed when the messages are submitted to the Crosschain Coordination Contract. This exposure could be mitigated by using a new randomly generated identity for each submission to the contract. The Crosschain Coordination Contract trusts the messages based on the threshold signatures, and not on the transaction signature.

Due to the simple fail-if-locked locking scheme, the system can not have *dead locks*, though could suffer from *live lock*.

Due to the global per-transaction time-out offered by the Coordination Blockchain, the system will not have *liveness* issues. Once a Crosschain Transaction is committed, ignored, or times-out, all contracts are unlocked. If nodes on a blockchain refused to process a *Signalling Transaction*, then contracts that were part of the crosschain transaction on the blockchain would remain locked. However, this situation is analogous to nodes refusing to produce any more blocks, and hence not related to crosschain consensus.

The system does not allow for partial updates. Errors are returned if a user attempts to read values from a locked contract. As such, this methodology is a *safe* crosschain consensus mechanism.

### *F. State Pinning*

As per Sec. II-D and Sec. II-E, State Pinning and Final State Pinning techniques aims to lock the overall state of one blockchain to another blockchain.

*1) Merge Mining:* Merge Mining [44][45][46] is a technique in which the Block Hash of a low hashing power public blockchain, such as NameCoin, is included in a more secure higher hashing power blockchain, such as Bitcoin. In this scenario, the Bitcoin miners must validate NameCoin transactions prior to including the Block Hash in a transaction on the Bitcoin network. The mined transaction can then be included in both the Bitcoin and NameCoin blockchain.

Merged Mining relies on both blockchains using the same consensus algorithm, and assumes that all transactions can be viewed by both blockchains. As such, this technique is not usable in a private blockchain scenario where transactions must not be revealed outside the blockchain.

The liveness of a Merge Mined blockchain relies on the miners being sufficiently incentivised to include blocks from the Merge Mined blockchain in the higher hashing power blockchain. The safety of the technique relies on a variety of miners on higher hashing power blockchain choosing to include blocks. For example, as of 2015, NameCoin has only been mined by one miner [44].

*2) Block Hash Posting:* The Kaledio team developed Tethered Permissioned Private Chains [47][9] to reduce the risk of state reversion occurring by posting the state of the blockchain onto Ethereum MainNet. In Kaleido's system, a trusted entity

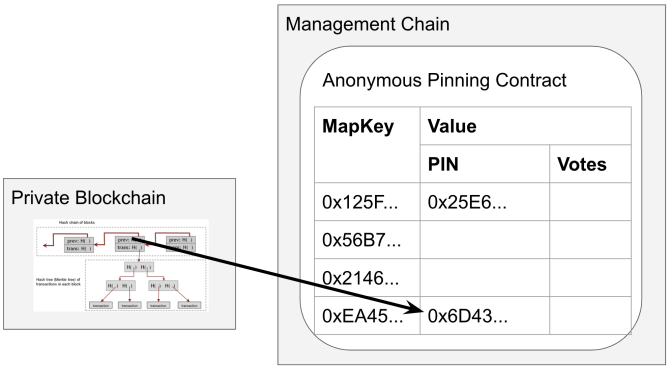



Fig. 20. Anonymous Pinning Map

may be used to submit a block hash on behalf of blockchain participants, thus keeping the participants secret.

The rate of transactions on the consortium chain is revealed on Ethereum MainNet due to the number of posted Pins. Additionally, the solution lacks a method of contesting a Pin posted to Ethereum MainNet.

*3) Anonymous Block Hash Posting:* Robinson and Brainard proposed Anonymous State Pinning [8] as a method of posting block hashes of a private blockchain to a *Management* blockchain without revealing the identities of participants of the private blockchain or the rate of transactions, while allowing posted block hashes to be contested. Pins (block hashes) are posted to certain entries in a map as shown in Fig. 20.

The value of the `MapKey` value is calculated using equation Equ. 3, where `KECCAK` is a message digest algorithm, Private Blockchain Identifier (`PBI`) identifies the blockchain, $\texttt{Pin}_{\texttt{t-1}}$ is the previous Pin value, `PRF` is a pseudo random function which is seeded with a Private Blockchain Secret.

$$MapKey_{\mathrm{t}} = KECCAK\text{-}256(PBI, Pin_{\mathrm{t\text{-}1}}, PRF(t)) \quad (3)$$

Participants of a private blockchain observe the pinning map at the $\mathrm{MapKey_t}$ address corresponding to the next Pin, waiting for the next Pin to be posted to that entry in the map. When the Pin value is posted, they check that the posted Pin matches their understanding of the most recent Block Hash of the private blockchain. If the values do not match, then participants should contest the Pin. To contest the Pin, $\mathrm{Pin_t}$, which is at $\mathrm{MapKey_t}$, they submit to the contract: $\mathrm{MapKey_{t\text{-}1}}$, PRF(t), and the PBI.

Submitting the previous value of the MapKey, $\mathrm{MapKey_{t\text{-}1}}$, allows the contract to fetch from its own storage the value of the previous Pin, $\mathrm{Pin_{t\text{-}1}}$. The contract can then calculate the MapKey of the contested Pin, $\mathrm{MapKey_t}$, by combining $\mathrm{Pin_{t\text{-}1}}$, PRF(t) and the PBI using Equation 3. Given the submitter of the transaction knows the PRF(t) which combined with the PBI links the previous MapKey, $\mathrm{MapKey_{t\text{-}1}}$, and the calculated MapKey, $\mathrm{MapKey_t}$, it implies that both of the MapKeys correspond to Pins for the private blockchain denoted by PBI. The further implication of knowing PRF(t) is that the transaction submitter has access to the Private Blockchain Secret, which implies that they are a member of the private blockchain.

To hide the identity of participants, *masked participants* are represented by a salted hash or their account number. *masked participants* may unmask themselves at any time to become *unmasked participants* by presenting their secret salt value. Only *unmasked participants* can vote on whether a contested Pin is valid.

As multiple blockchains can use the same map, it is difficult for observers to determine the rate of pinning for a particular blockchain, assuming one account is used to submit transactions for multiple blockchains.

The *liveness* of the system depends on Pins being posted when needed. The *safety* of the system depends on all participants being able to observe the management chain at all times so that they can contest Pins when needed. This requirement of being online all the time presents is an onerous requirement which may not be able to be fulfilled in all deployments.

*4) BLS Signed Block Hash Posting:* The validators of a blockchain could collaborate to use BLS Threshold Signatures (see Info. IV-E4) to threshold sign block hashes that are submitted to a *management* blockchain. In this scheme, the public key related to the private key shares would be stored on the *management* blockchain. The signed Pins are verified when they are submitted to the contract used to store the Pins, thus ensuring only valid Pins signed by a majority of validators are accepted.

The advantage of this technique over the *Anonymous Block Hash Posting* is that validators do not need to monitor the management blockchain as only valid Pins are accepted. A disadvantage of this technique is that it is more computationally expensive both on the private blockchain side, having to threshold sign the Pin, and in the smart contract on the management blockchain, where a BLS signature needs to be verified.

## V. Summary

The table below summarises the approaches to crosschain consensus discussed in this chapter. The table shows the scenarios and type of blockchain each technique is suitable for, plus provides a short summary of how the technique provides cross-blockchain consensus.

## Acknowledgment

I thank Dr Catherine Jones for her continued support and astute review of this chapter.

## References

[1] P. Robinson, D. Hyland-Wood, R. Saltini, S. Johnson, and J. Brainard, "Atomic Crosschain Transactions for Ethereum Private Sidechains," 2019. [Online]. Available: https://arxiv.org/abs/1904.12079

[2] E. Prud'hommeaux and C. Buil-Aranda, "Federated Query, W3C Recommendation," 2013. [Online]. Available: https://www.w3.org/TR/sparql11-federated-query/

[3] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation," 2014. [Online]. Available: http://www.w3.org/TR/rdf11-concepts/

[4] B. J. Nelson, "Remote procedure call," Ph.D. dissertation, 1981. [Online]. Available: https://dl.acm.org/citation.cfm?id=910306

[5] F. Vogelsteller and V. Buterin, "EIP 20: ERC-20 Token Standard," 2015. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

| Methodology | Scenarios | | | | Blockchain Attributes | |
|---|---|---|---|---|---|---|
| | Value Swap | Reading Data | Writing Data | State Pinning | Public Permissionless | Private Permissioned |
| Hash Timelocked Contracts | ✓ | | | | ✓ | ✓ |
| | Consensus: Two party agreement with *safety* issues. | | | | | |
| BTC Relay | ✓ | ✓ | | | ✓ | |
| | Consensus: Block header transfer relies on consensus of Bitcoin. | | | | | |
| Pegged Sidechains | ✓ | | | | ✓ | |
| | Consensus: Block header transfer relies on consensus of Bitcoin and a sidechain. Safety issues for transferring from sidechain to Bitcoin. | | | | | |
| Ion | ✓ | ✓ | | | | ✓ |
| | Consensus: Block header transfer relies on threshold number of signatures. | | | | | |
| Cosmos | | ✓ | ✓ | ✓ | ✓ | |
| | Consensus: Block header transfer relies on threshold number of signatures. *Safety* issues due to lack of guarantees around atomic behaviour. | | | | | |
| Polkadot | - | - | - | - | ✓ | |
| | Consensus: Parachains share the consensus of the Relay Chain. Shared consensus technique, not a cross blockchain approach. | | | | | |
| Plasma | ✓ | - | - | - | ✓ | |
| | Consensus: Users present exit proofs and can be challenged with fraud proofs. Has *safety* issues. | | | | | |
| Atomic Crosschain Transactions | ✓ | ✓ | ✓ | | ✓ | ✓ |
| | Consensus: BLS threshold signatures, in combination with a coordination contract and nested transactions provides cross-blockchain consensus and ensures atomic behaviour. | | | | | |
| Merge Mining | | | | ✓ | ✓ | |
| | Consensus: Uses consensus of blockchain merged to. | | | | | |
| Tethered Blockchains | | | | ✓ | ✓ | ✓ |
| | Consensus: One validator posts block hashes. | | | | | |
| Anonymous Pinning | | | | ✓ | ✓ | ✓ |
| | Consensus: Posted block hashes can be contested. | | | | | |
| Threshold Signed State | | | | ✓ | ✓ | ✓ |
| | Consensus: A threshold number of validators sign block hashes. | | | | | |

TABLE I
COMPARISON OF CROSSCHAIN CONSENSUS TECHNIQUES


[6] "ERC20 Token Standard," 2018. [Online]. Available: https://theethereum.wiki/w/index.php/ERC20{_}Token{_}Standard

[7] D. Shirley, "EIPs/eip-721.md at master · ethereum/EIPs," 2018. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

[8] P. Robinson and J. Brainard, "Anonymous State Pinning for Private Blockchains," 2019. [Online]. Available: https://arxiv.org/abs/1903.02752

[9] A. Ahmed and J. Zhang, "Enhanced Immutability of Permissioned Blockchain Networks by Tethering Provenance with a Public Blockchain Network," 2018. [Online]. Available: http://console.kaleido.io/docs/docs/static/TetheredPermissionedPrivateChains.pdf

[10] E. E. Alliance, "Enterprise Ethereum Alliance - Enterprise Ethereum Client Specification V2," 2018. [Online]. Available: http://entethalliance.org/wp-content/uploads/2018/10/EEA{_}Enterprise{_}Ethereum{_}Client{_}Specification{_}V2.pdf

[11] P. Robinson, "Requirements for Ethereum Private Sidechains," 2018. [Online]. Available: http://adsabs.harvard.edu/abs/2018arXiv180609834R

[12] "Hyperledger Fabric 1.1 Documentation," 2018. [Online]. Available: http://hyperledger-fabric.readthedocs.io/en/release-1.1/getting{_}started.html

[13] "Corda Documentation," 2018. [Online]. Available: https://docs.corda.net/

[14] "Hashed Timelock Contracts," 2017. [Online]. Available: https://en.bitcoin.it/wiki/Hashed{_}Timelock{_}Contracts

[15] V. Buterin, "Chain Interoperability," 2016. [Online]. Available: http://andolfatto.blogspot.ca/2015/02/fedcoin-on-desirability-of-government.html

[16] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[17] S. Thomas and E. Schwartz, "A Protocol for Interledger Payments." [Online]. Available: https://interledger.org/interledger.pdf

[18] O. Guindzberg, "Doge Ethereum Bridge," 2018. [Online]. Available: https://github.com/dogethereum/docs

[19] E. I. Sunnyvale, "PROTOCOLS FOR PUBLIC KEY CRYPTOSYSTEMS Ralph C. Merkle," *Distribution*, pp. 122–134, 1980.

[20] J. Chow, "BTC Relay," 2016. [Online]. Available: https://media.readthedocs.org/pdf/btc-relay/latest/btc-relay.pdf

[21] ——, "BTC Relay Source Code," 2016. [Online]. Available: https://github.com/ethereum/btcrelay/blob/develop/btcrelay.se

[22] M. Swende, "Hacking on BTC Relay," 2016. [Online]. Available: https://swende.se/blog/BTCRelay-Auditing.html

[23] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling Blockchain Innovations with Pegged Sidechains," 2014. [Online]. Available: https://blockstream.com/sidechains.pdf

[24] G. Wood, "Polkadot: Vision for a Hetrogeneous Multi-Chain Framework," 2016. [Online]. Available: https://github.com/polkadot-io/polkadotpaper/raw/master/PolkaDotPaper.pdf

[25] Clearmatics, "General interoperability framework for trustless cross-system interaction," 2018. [Online]. Available: https://github.com/clearmatics/ion

[26] S. Johnson, P. Robinson, and J. Brainard, "Sidechains and interoperability," 2019. [Online]. Available: https://arxiv.org/pdf/1903.04077.pdf

[27] Y.-T. Lin, "EIP 650: Istanbul Byzantine Fault Tolerance," 2017. [Online]. Available: https://github.com/ethereum/EIPs/issues/650

[28] P. Robinson, "Forks, Reorgs, Finality, and Selfish Mining," 2019. [Online]. Available: https://www.youtube.com/watch?v=huoUwuOEOm0

[29] E. Buchman and J. Kwon, "Cosmos: A network of distributed ledgers." Github, 2016. [Online]. Available: https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md

[30] J. Kwon, "Tendermint," 2016. [Online]. Available: https://github.com/tendermint/tendermint/wiki

[31] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Third Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186. [Online]. Available: http://pmg.csail.mit.edu/papers/osdi99.pdf

[32] ——, "Practical Byzantine Fault Tolerance and Proactive Recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4 November 2002, pp. 398–461, 2002. [Online]. Available: http://www.pmg.csail.mit.edu/papers/bft-tocs.pdf
[33] "Architecture · Polkadot Wiki," 2019. [Online]. Available: https://wiki.polkadot.network/docs/en/learn-architecture
[34] V. Buterin, "Minimum Viable Plasma," 2018. [Online]. Available: https://ethresear.ch/t/minimal-viable-plasma/426
[35] J. Poon and V. Buterin, "Plasma: Scalable Autonomous Smart Contracts," 2017. [Online]. Available: https://plasma.io/plasma.pdf
[36] Ripper234, "What is an unspent transaction output?" 2012. [Online]. Available: https://bitcoin.stackexchange.com/questions/4301/what-is-an-unspent-output
[37] K. Floersch, "Plasma Cash Simple Spec," 2018. [Online]. Available: https://karl.tech/plasma-cash-simple-spec/
[38] G. Wood, "Ethereum: a secure decentralized generalised transaction ledger," Github, p. Github site to create pdf, 2016. [Online]. Available: https://github.com/ethereum/yellowpaper
[39] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme." [Online]. Available: http://www-cse.ucsd.edu/users/aboldyre
[40] P. Robinson, "Ethereum Engineering Group: BLS Threshold Signatures - YouTube," 2019. [Online]. Available: https://www.youtube.com/watch?v=XZTvBYG9pn4
[41] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979. [Online]. Available: https://cs.jhu.edu/{~}sdoshi/crypto/papers/shamirturing.pdf
[42] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004. [Online]. Available: https://doi.org/10.1007/s00145-004-0314-9
[43] T. P. Pedersen and D. W. Davies, "A Threshold Cryptosystem without a Trusted Party," in *Advances in Cryptology EUROCRYPT '91*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 522–526. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-46416-6{_}47
[44] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design," in *WEIS*, 2015. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.698.4605{&}rep=rep1{&}type=pdf
[45] D. Schwartz, "How does merged mining work?" 2011. [Online]. Available: https://bitcoin.stackexchange.com/questions/273/how-does-merged-mining-work
[46] D. Roberts, "Merged Mining," 2014. [Online]. Available: https://github.com/namecoin/wiki/blob/master/Merged-Mining.mediawiki
[47] "Kaleido Product," 2018. [Online]. Available: https://kaleido.io/product/