Cross-chain Deals and Adversarial Commerce

Presenters - Kalyan Valiveti, Rajat Jalal, Baadal Bhojak, Sadaf Arshad

Original Presentation - Keyi Wu, Jingyuan Li

Authors - Maurice Herlihy, Barbara Liskov, and Liuba Shrira

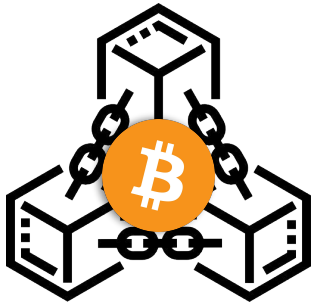
2019 May 23

RoadMap

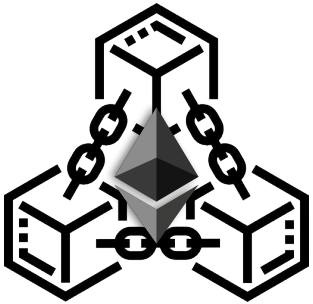
- What is a cross-chain deal and Why do we need it?
- How It Works
- Two Protocols
- Cost Analysis
- Discussion

Introduction

Imagine a world where everything’s ownership is documented in its own blockchain...

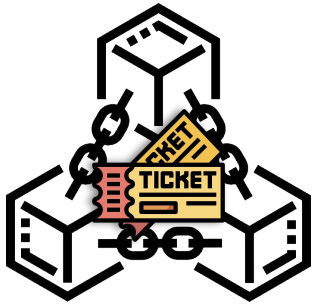


Bitcoin



Ethereum

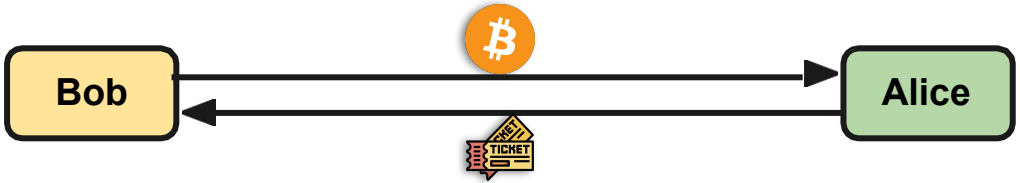
Milk



Film ticket

Introduction

How do we make a ***(cross-chain) transaction*** in such a world?



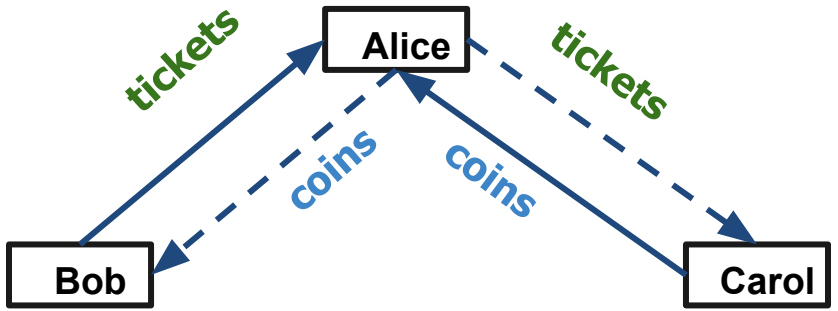
Introduction - Cross-chain Deal

- A new way to structure complex distributed computations that manage assets in an adversarial setting.
- Multi-step transfers are OK.
- Multi-blockchain transfers are OK.
- There is no size limit of faulty parties, every party can be a faulty party.
- All parties that follow the protocol end up “no worse off” than when they started, even in the presence of faulty or malicious behavior by an arbitrary number of other parties.

Other Approach: Cross-chain Swap

- Existing proposals for cross-chain swaps set up a collection of ***unconditional*** transfers from one party to another.
- Each party checks that it is satisfied with its own transfers in and out, and then when all approve, the transfers take place.

Incapable of expressing many kinds of standard financial transactions: broker, auction...



Example Deal

Table 1: Alice, Bob, and Carol's deal. Rows represent outgoing transfers, and columns incoming transfers

	Alice	Bob	Carol
Alice		100 coins	tickets
Bob	tickets		
Carol	101 coins		

Cross-chain Deal vs Atomic Transactions

- Transactions perform complex distributed state changes, while deals simply exchange assets among parties
- Transactions use “all-or-nothing” semantics to preserve global invariants, however each autonomous party in a deal can decide independently whether it finds an outcome satisfactory.
- Transactions assume crash failures, whereas deals assume byzantine faults.

How Cross-chain deals work

- An escrow is used for ensuring that a single asset cannot be transferred to different parties at the same time.
- Here is what happens when P places asset ‘a’ in an escrow during deal D:

  Pre: Owns(P, a)

  Post: Owns(D, a) and Owns_C (P, a) and Owns_A(P, a)

- The precondition states that P can escrow A only if P owns a. The postcondition states that ownership of a is transferred from P to D (via the escrow contract), but P remains the owner of a in both Commit and Abort.

How Cross-chain deals work(Cont..)

- When party P tentatively transfers an asset (or assets) a to party Q as part of deal D:

  Pre: Owns(D, a) and Owns_C(P, a)

  Post: Owns_C(Q,a)

Traditional Notions of Correctness

- ***Atomicity***: Either all steps happen, or none do.
- ***Isolation***: No transaction observes other's intermediate state.
- ***Consistency***: ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants
- ***Durability***: once a transaction has been committed, it will remain committed even in the case of a system failure

New Notions of Correctness

- ***Safety***: For every protocol execution, every compliant party ends up with an acceptable payoff.
- ***Weak liveness***: No asset belonging to a compliant party is locked in an escrow forever.
- ***Strong liveness***: If all parties are compliant and willing to accept their proposed payoffs, then all transfers happen.

Timelock Protocol

- Uses commits and timeouts to release assets to new/original owners.
- Parties do not explicitly vote to abort. Instead, timeouts are used to ensure that escrowed assets are not locked up forever if some party crashes or walks away from the deal.
- Assumes a synchronous network model

How It Works - Five Phases

- **Clearing Phase**
- **Escrow Phase**
- **Transfer Phase**
- **Validation Phase**
- **Commit Phase**

Clearing Phase

- All the participating parties are informed about:
  - ❖ Deal identifier D
  - ❖ List of participating parties *plist*
  - ❖ Commit phase starting time t0 and timeout delay Δ

Escrow Phase

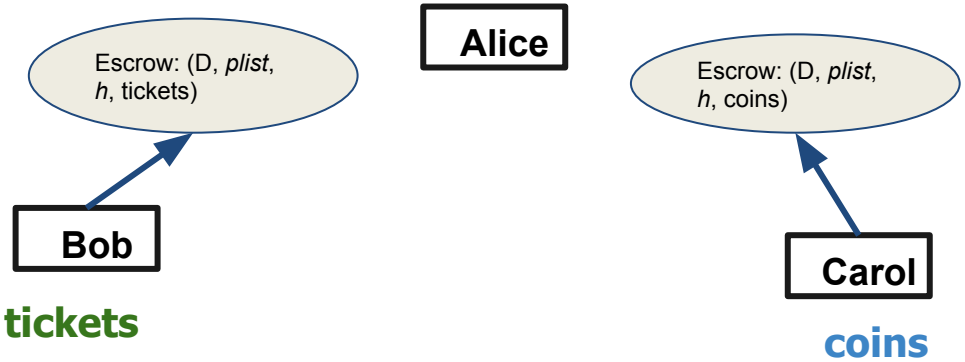
- Each party places its outgoing assets in escrow through an escrow contract

escrow(D,Dinfo,a)

where D is the deal identifier

Dinfo is the rest of the info about the deal (plist, t0, Δ)

a is the asset



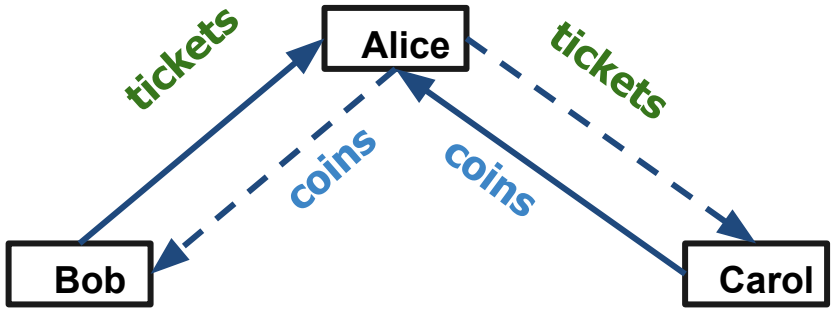
Transfer Phase

- Party transfer an asset tentatively to another owner by sending

  transfer(D,a,Q)

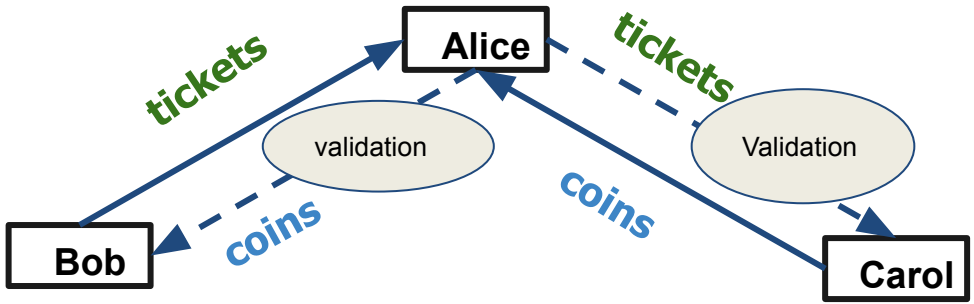
  to the escrow contract

  where D is the deal identifier
  a is the asset
  Q is the new owner



Validation Phase

- Each party examines its escrowed incoming assets to see if they represent an acceptable payoff.
- The deal information provided is also verified in the validation phase.

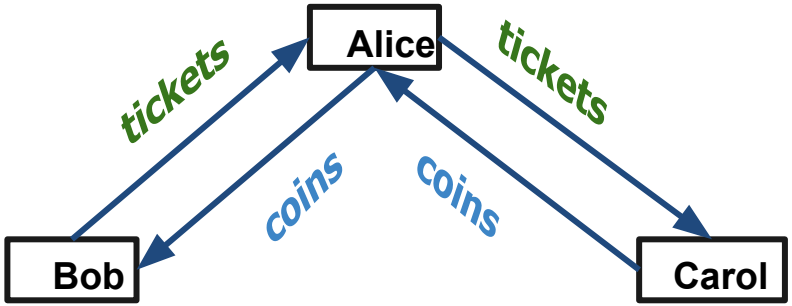


Commit Phase

- Each party sends a commit vote to the escrow contract for each incoming asset.
- In order to send a commit message a party uses:

  commit(D,v,p)

  where D is the deal identifier
  v is the voter
  p is the path signature for v's vote

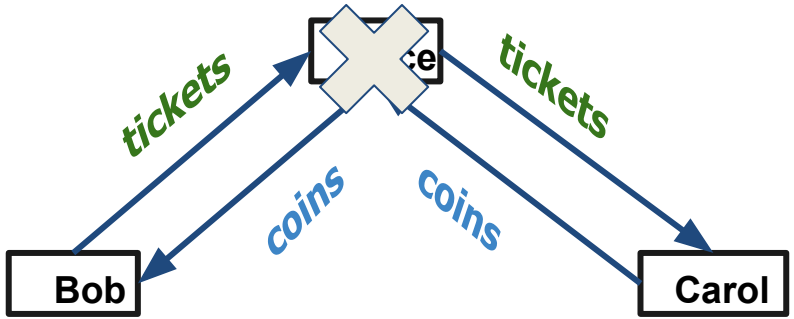


Time Out

- A commit is only accepted by a contract if it is arrives within the commit phase starting time t0 and is well formed.
- Once a commit is accepted, that contract has now accepted a vote from the party.
- Escrowed assets are released to the new owners by the contract once it has received a commit vote from every party.
- If the contract does not receive a commit vote from every party within t0+NΔ time, the missing votes are not at all accepted and the contract will time out.
- The escrowed assets are then rolled back to their original owners.

What can go wrong?

- **Any deviating party may not benefit more than normal protocol execution -** it can have no extra profit or be penalized.
- **No conforming party is worse off -** it might get normal value it agreed to in the deal or get extra profit/value.



Correctness

- The timelock protocol satisfies safety.
- The timelock protocol satisfies weak liveness.
- The timelock protocol satisfies strong liveness.

CBC Protocol

- Semi-synchronous
- No bound on message delivery times
- We use a special blockchain, the *Certified blockchain*, CBC which is kind of like a shared log
  - The CBC can be a preexisting blockchain or a new unique one
- Parties vote on the entire deal and record the vote on the CBC
- A Party claiming an asset(or refund) presents a proof of commit(proof of abort) to contract managing the asset.
  - Proof of commit- proof that every party voted to commit the deal
  - Proof of abort- proof that *some* party voted to abort the deal

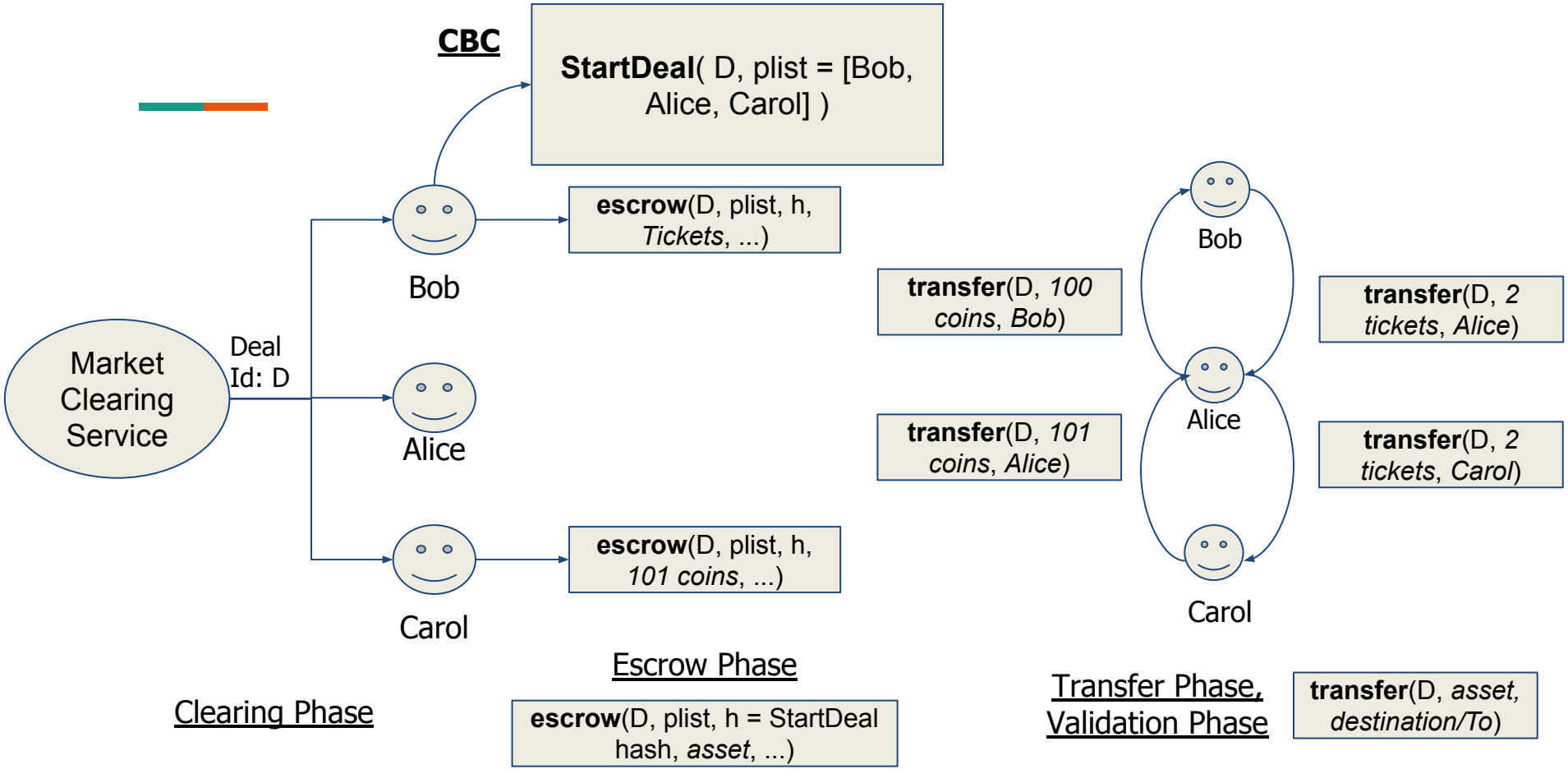
Phases of a CBC protocol

- Clearing Phase:
  - The market-clearing service broadcasts a unique identifier D and a list of participating parties plist
  - Doesn’t require t or Δ
  - One party adds the startDeal(D, plist) on the CBC
- Escrow Phase: escrow(D, plist, h, a, . . .)
  - Each party places its outgoing assets in escrow
  - h is the hash that identifies a unique startDeal entry on the CBC that started the deal

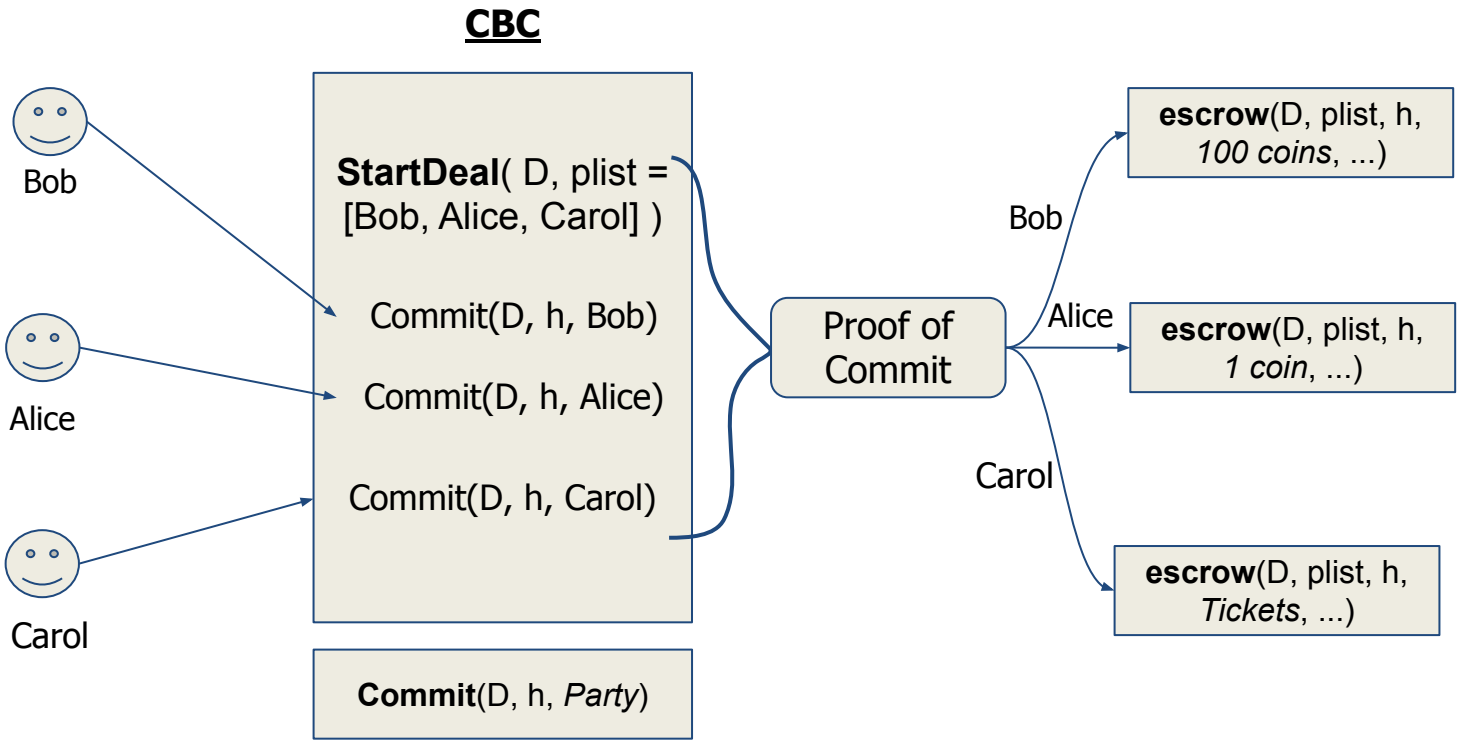
Phases of a CBC protocol

- Transfer Phase: transfer (D, a, Q). ~similar to Timelock
- Validation Phase ~similar to Timelock
- Commit Phase: commit(D, h, P) or abort(D, h, P)
  - Each party P publishes either a commit or abort vote for D **on the CBC**
  - D is the deal identifier
  - h is the startDeal
  - If all parties commit, they receive proof of commit, parties submit that to escrow to get paid assets
  - If some parties abort, they receive proof of commit, parties submit that to escrow to get back original assets

Phases of a CBC protocol

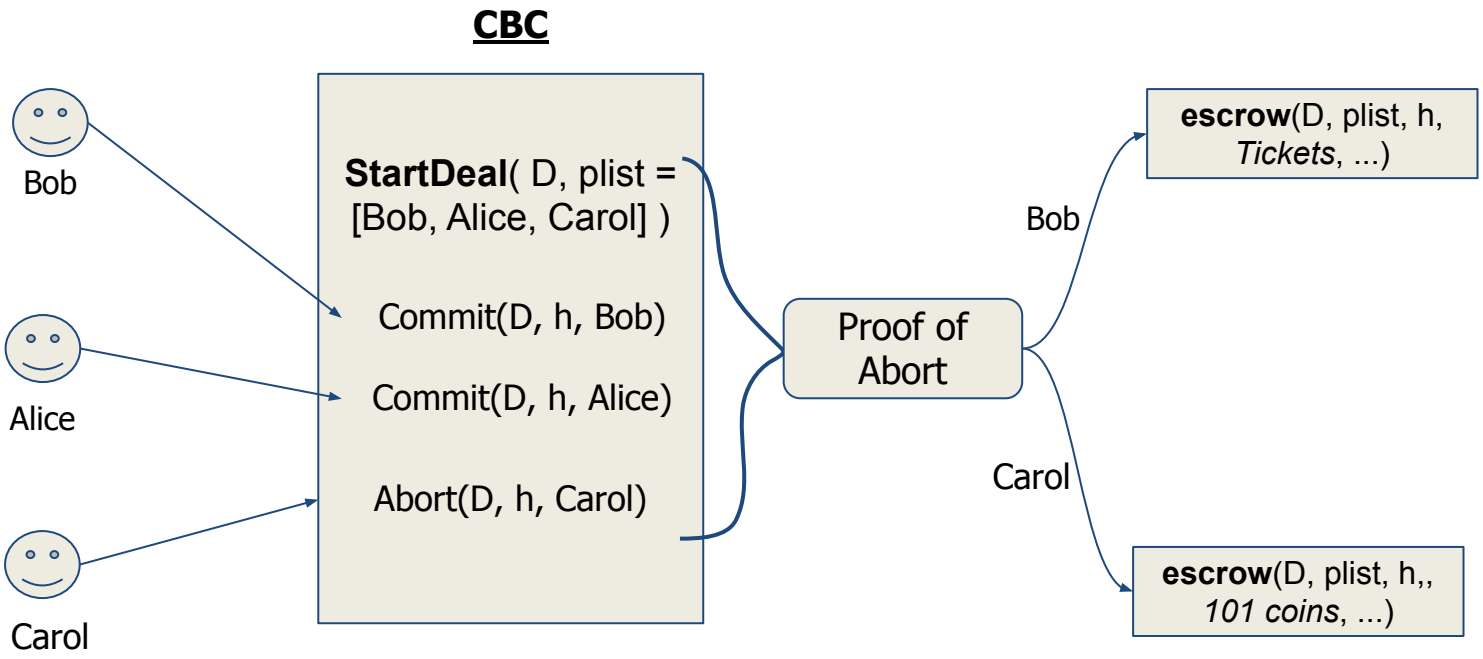


Commit Phase - Proof of Commit



- Each party waits for all other parties to post commit or abort on the CBC
- If a party is waiting too long, after *x* amount of time it can vote to abort the protocol, satisfying weak liveness.
- A party can change its vote after voting commit after *x* amount of time.

Commit Phase - Proof of Abort



What can go wrong?

- **What if:** In previous example, if Carol sends Alice 1001 coins instead of 101 coins, and all parties commit?
  - Since all parties committed, the deal will still execute.
  - In this case, Carol now is a non protocol-conforming party since they did not follow the deal/did some random stuff.
- **No conforming party is worse off -** it might get normal value it agreed to in the deal or get extra profit/value.
- **Any deviating party may not benefit more than normal protocol execution -** it can have no extra profit or be penalized.

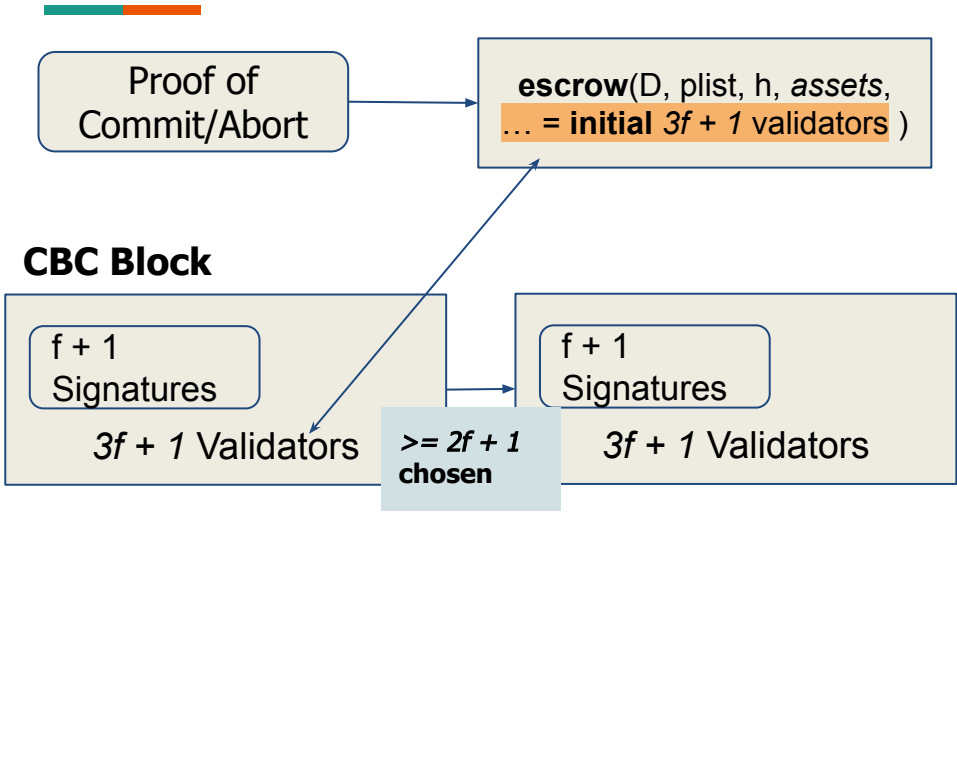
Cross Chain Proofs

- The decisive vote is the vote that determines whether deal commits or aborts.
- The party can submit the entire CBC sequence from startDeal to the decisive vote as proof of commit to the contract.
- **How can a contract in one blockchain trust the proof of commit it is shown by a party which includes other blockchains?**

Byzantine Fault-Tolerant Consensus

- Assume CBC relies on BFT Consensus
  - 3f + 1 validators, at most f malicious
  - To support long term fault tolerance, blockchain is periodically reconfigured by having at least *2f+1* current validators select new set of validators.
  - Each block contains the next block's group of validators
  - Each block is vouched for by a certificate containing at least f + 1 validator signatures of that block's hash
- escrow(D, plist, h, a, ...) ← initial 3f + 1 validators

Byzantine Fault-Tolerant Consensus



- Parties will identify correct initial validators when putting assets in escrow
- Check validators' credentials before voting to commit

Cost Analysis

Two kinds of operations: writing to long-lived storage & each signature verification

Gas costs for a deal with n parties, m assets, and t ≥ n transfers.

Protocol	Escrow	Transfer and Validation	Commit or Abort
Timelock	$O(m)$ writes	$O(t)$ writes	$O(mn^2)$ sig. ver. + $O(m)$ writes
CBC	$O(m)$ writes	$O(t)$ writes	$O(m(f+1))$ sig. ver. + $O(m)$ writes

```
1  contract EscrowManager {
2      ERC20Interface asset;                  // contract holding assets
3      mapping(address => uint) escrow; // escrowed assets
4      mapping(address => uint) onCommit; // result of tentative transfers
5       ...
6      // transfer into escrow account
7      function escrow (uint amount) public {
8          require (asset.transferFrom(msg.sender, this, amount));
9          escrow[msg.sender] = escrow[msg.sender] + amount;
10         onCommit[msg.sender] = onCommit[msg.sender] + amount;
11     }
12     // tentative transfer
13     function transfer (address to, uint amount) public {
14         require (onCommit[msg.sender] >= amount);
15         onCommit[msg.sender] = onCommit[msg.sender] - amount;
16         onCommit[to] = onCommit[to] + amount;
17     } ... }
```

O(m) writes

O(t) writes

```
1  contract TimelockManager is EscrowManager{
2      address[] parties;              // participating parties
3      address[] voted;                // which parties have voted
4      ...
5      function commit(address voter, address[] signers, bytes32[] sigs) public {
6          require (now < start + (path.length() * DELTA)); // not timed out
7          require (parties.contains(voter));               // legit voters only
8          require (!voted.contains(voter));                // no duplicate votes
9          require (checkUnique(signers));                  // no duplicate signers
10         for (int i = 0; i < signers.length; i++) {
11             require (checkSig(voter, signers[i], sigs[i])); // expensive
12         }
13         voted.push(voter);                               // remember who voted
14     }}
```

O(mn^2) sig. verf. **+ O(m)** writes

```
1  contract CBCManager is EscrowManager{
2      address[] validators;          // CBC validators
3      ...
4      // check commit proof is valid
5      function commit(address[] signers, bytes32[] sigs) public {
6          require(checkUnique(signers));              // no duplicate signers
7          require(validators.contains(signers));      // only validators voted
8          require(signers.length >= f+1);             // enough validators voted
9          for (int i = 0; i < f+1; i++) {
10             require(checkSig(signers[i], sigs[i])); // expensive
11         }
12         outcome = COMMITTED;                        // remember we committed
13     } ... }
```

O(m(f+1)) sig. verf. **+ O(m)** writes

Time cost for synchronous communication

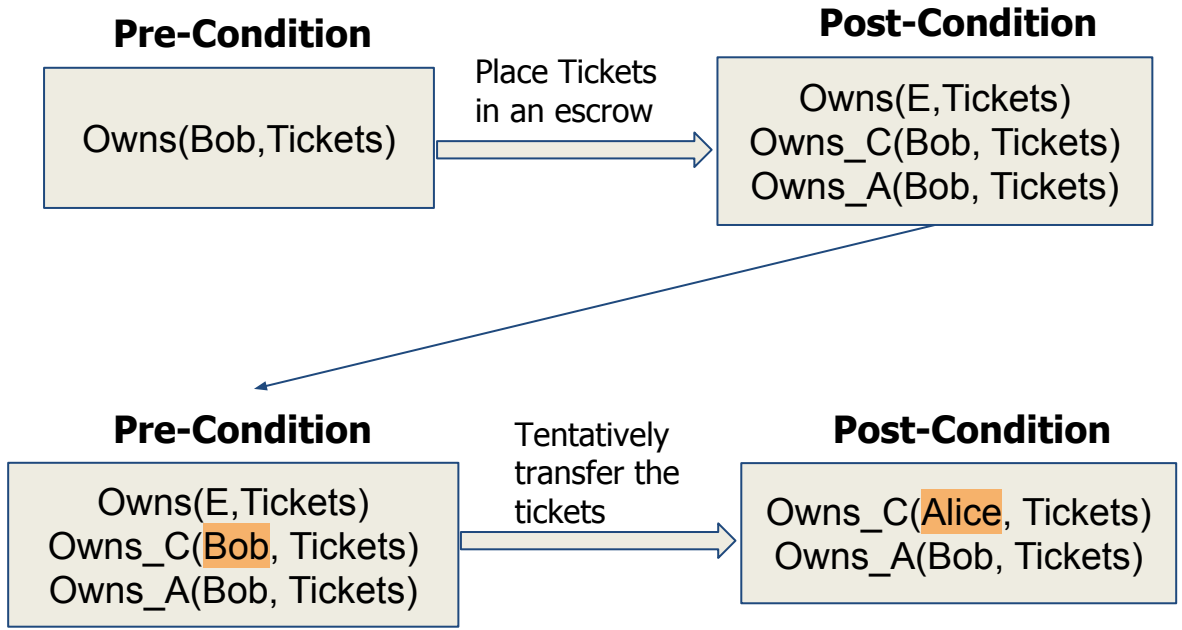
Protocol	Escrow	Transfer and Validation	Commit	Abort
Timelock	Δ	$k\Delta$ or Δ	$O(n)\Delta$	$O(n)\Delta$
CBC	Δ	$k\Delta$ or Δ	$O(1)\Delta$	per-party timeout

Discussion

- Provide incentives for good behavior by escrowing small deposits from parties.
- Denial-of-service attack could happen in both Timelock and CBC
- CBC: censorship
- n parties vs. f + 1 validators

Questions?

- E = Escrow
- Asset = Tickets
- Owns_C = Who owns Tickets if a deal is committed
- Owns_A = Who owns Tickets if deal is aborted



What does safety entail in Timelock and CBC

1. What does safety entail?
    a. What happens if someone backs out of the deal last minute in terms of safety?
    b. How is the paper guaranteeing the safety?

    Safety entails the following:
    - Every compliant party ends with an acceptable payoff.

    If someone backs out of the deal last minute then the escrowed assets are refunded back to their original owners, thus, every complaint party is “no worse off”.

    For timelock protocol, the paper guarantees safety by ensuring that every commit vote for an asset has the signature of the party to whom the vote is forwarded.

    For CBC, the paper guarantees safety because complaint parties agree on whether a deal commits or aborts.

Proof-of-work (Nakamoto) Consensus

- Producing commit or abort proofs from a proof-of-work CBC
- Lack finality (Any proof might be contradicted by the later proof)
- A proof includes some confirmation blocks
- The number of confirmation blocks depends on the value of the deal

ByShard and CBC Protocol

Similarities

- Multi-shard transaction processing protocols
- Processes of shards broken down into three steps: vote, commit & abort
- Uses mechanism of locking

Difference

- 1 client per ByShard versus CBC can handle multiple clients
- BySharding is closer to cross-chain swaps rather than cross-chain deals