CONTENTS

VIII. Appendix

A. Proof

Lemma 2 (Uniformity): If parties have only local information, 3PP is uniform.

Proof. Uniformity is the conjunction of three conditions. We prove them in turn.

(1) If all parties follow 3PP, then the transaction is committed. We prove the following sequence of facts.

(a) All the contracts are eventually created. We prove that every party eventually creates its outgoing contracts. This fact is proved by induction on the maximum of the distances of the party from the leaders. In the base case, the leaders themselves create their outgoing contracts with no condition (lines P_2-P_3). In the inductive case, every follower eventually receives her last incoming contract from her predecessor that is on the path from the leader with the longest distance. In response, she creates her outgoing contracts (lines P_8-P_{11}). We note that as the parties are assumed to follow the protocol, the contracts that they create are valid. Thus, parties do not stop because of invalid contracts.

(b) Leaders do not wait indefinitely. By [a], leaders eventually observe their incoming contracts and in response, stop waiting for incoming contracts (line P_4). Further, from [a], the representative sinks receive their incoming contracts and send messages to representative sources (lines P_{12}-P_{13}). Therefore, the representative sources receive messages from all the representative sinks and stop waiting (lines P_6-P_7).

(c) Every secret eventually reaches a representative source. By [b], the feedback vertex set eventually start the second phase and release their secrets on their incoming edges (lines P_{15}-P_{16}). Parties propagate secrets backwards (lines P_{17}-P_{18}). Consider a leader v_l in the feedback vertex set. Consider its super-vertex w_l in the condensation graph. The vertices in w_l are an SCC; thus, they are reachable from each other. Therefore, the secret of v_l propagates to every party in w_l. Further, every super-vertex is reachable from at least one super-source. Therefore, the secret of v_l eventually propagates back to at least one super-source w_s. The vertices in w_s are an SCC and the clearing service has chosen one of them as the representative source. Therefore, the secret of the leader v_l eventually propagates to that representative source. In addition to the feedback vertex set, the representative sources are leaders themselves. Their secrets are trivially at the representative sources.

(d) All vertices eventually receive all secrets. The representative sources relay their secrets to the pseudo-sinks (lines P_{20}-P_{21}). Thus, by [c], every pseudo-sink eventually receives all secrets. In the condensation graph, every super-vertex is reachable to at least one super-sink. Further, vertices of a super-vertex are an SCC and reachable from each other. Parties other than the representative sources propagate secrets backwards (lines P_{23}-P_{24}). Therefore, the secrets are eventually propagated from the pseudo-sinks back to all vertices.

(e) The transaction is eventually committed. First, from [a], all contracts are eventually created. Second, from [d], all vertices eventually receive all secrets. Parties apply the secrets that they receive to their incoming contracts (lines P_{23}-P_{24}). Therefore, every contract is eventually triggered i.e. the transaction is committed.

(2) If any set of parties deviate from 3PP, no conforming party finishes with a $UnderWater$ state. A party ends up in a $UnderWater$ state when some of her outgoing contracts are triggered, but all her incoming contracts are not triggered. It is trivial that a source party cannot end up in a $UnderWater$ state as it does not have any incoming contracts. If the party is a conforming leader, she releases her secret only after she receives her incoming contracts (lines P_4-P_5). Therefore, her outgoing contracts can be triggered only after her incoming contracts are already created because all of her outgoing contracts are locked by her own secret. Thus, if any of her outgoing contracts are triggered, she can learn the secrets and subsequently trigger her incoming contracts. Conforming followers create outgoing contracts only after they receive their incoming contracts (lines P_8-P_{11}). Therefore, if an outgoing contract is triggered, they can learn the secrets and trigger their incoming contracts. Thus, if an outgoing contract of a conforming party is triggered, she can trigger all her incoming contracts. Thus, she cannot end up in a $UnderWater$ state.

(3) $\mathcal{P}$ is end-to-end. Assuming that the transaction is source-paid, we show that it is sink-paying. Since the transaction is source-paid, a contract is triggered in the transaction. Every contract is hashlocked with the secret of all leaders. The representative sources are in the set of leaders. Therefore, the secrets of the representative sources should be known. Therefore, the third phase must have started. Similar to 1.(c), we can show that by the end of the second phase, all the secrets reach the representative sources. At the beginning of the third phase, the representative sources send secrets to pseudo-sinks. These secrets include representative sources' own secrets and the secrets of the feedback vertex set that they have received in the second phase. Every complying vertex applies the secrets that it receives to its incoming contracts. Therefore, the complying pseudo-sinks apply the secrets to the incoming contracts and trigger them. Thus, the transaction is sink-paying. □

IX. Use-cases

A. Use-case 1

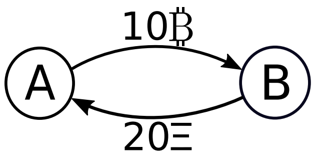
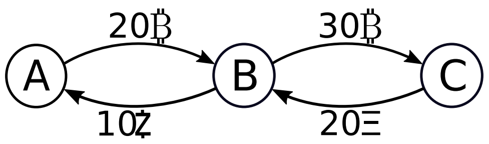


Fig. 9. First Use-Case

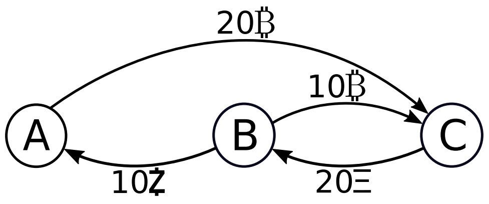
Leaders	Synthesis Time	Gas Consumption				
	(ms)	Deploy	Initiate	Redeem	Claim	Total
A	294	1220926	77772	47366	54074	1400138
		1221334	77794	47322	54074	1400524

TABLE I
Execution of First Use-case

B. Use-case 2



(a) Transaction with 3 nodes



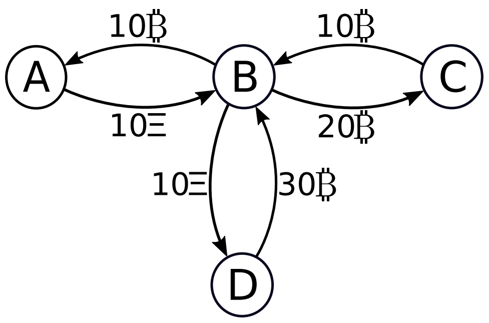
(b) Equivalent transformation of the transaction (a)

Fig. 10. Second Use-case

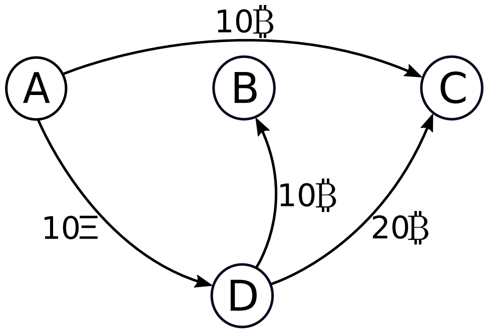
Leaders	Synthesis Time (ms)	Gas Consumption				
		Deploy	Initiate	Redeem	Claim	Total
B	375	1263228	78350	47300	55644	1444522
		1367090	78328	47366	55622	1548406
		1263160	78350	47300	55644	1444454
		1263368	78350	47300	55644	1444662

TABLE II
EXECUTION OF SECOND USE-CASE

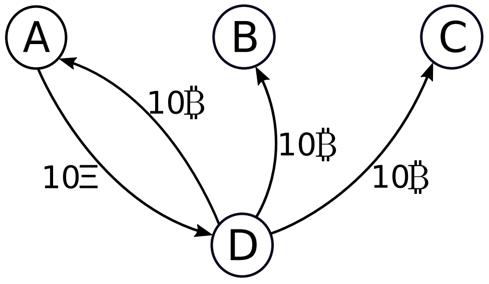
C. Use-case 3



(a) Transaction with 4 nodes



(b) Equivalent transformation of the transaction (a)



(c) Equivalent transformation of the transaction (b)

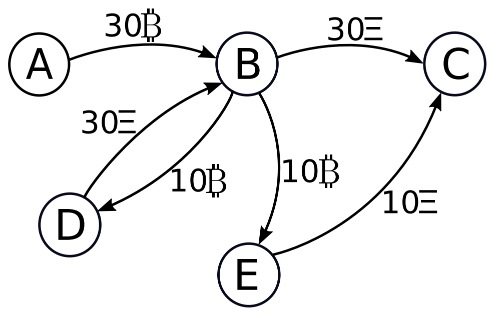
Fig. 11. Equivalent transformation of the transaction (b)

Fig. 12. Third Use-case

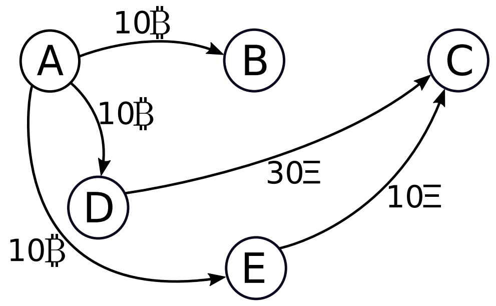
Leaders	Synthesis Time	Gas Consumption				
	(ms)	Deploy	Initiate	Redeem	Claim	Total
A	330	1721271	78902	47342	57188	1904703
		1617422	78902	47342	57188	1800854
		1513707	78946	47254	57232	1697139
		1304858	78906	47258	57170	1488192

TABLE III
EXECUTION OF THIRD USE-CASE

D. *Use-case 4*



(a) Transaction with 5 nodes



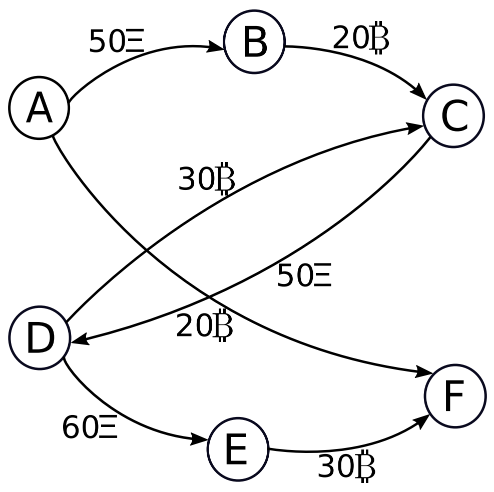
(b) Equivalent transformation of the transaction (a)

Fig. 13. Fourth Use-case

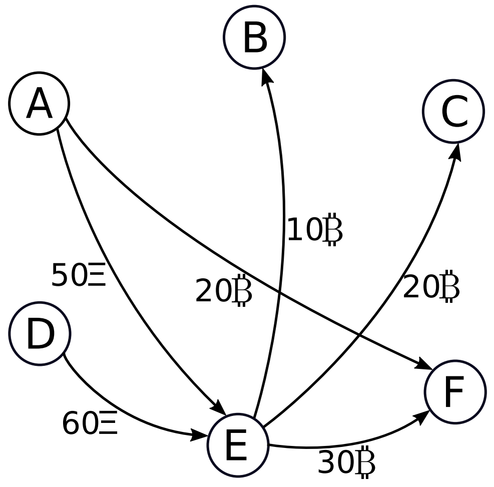
Leaders	Synthesis Time (ms)	Gas Consumption				
		Deploy	Initiate	Redeem	Claim	Total
A	310	1347499	79462	47306	58740	1533007
		1347499	79462	47306	58740	1533007
		1347499	79462	47306	58740	1533007
		1347435	79462	23874	21693	1472464
		1347631	79462	47306	58740	1533139

TABLE IV
EXECUTION OF FORTH USE-CASE

E. Use-case 5



(a) Transaction with 6 nodes



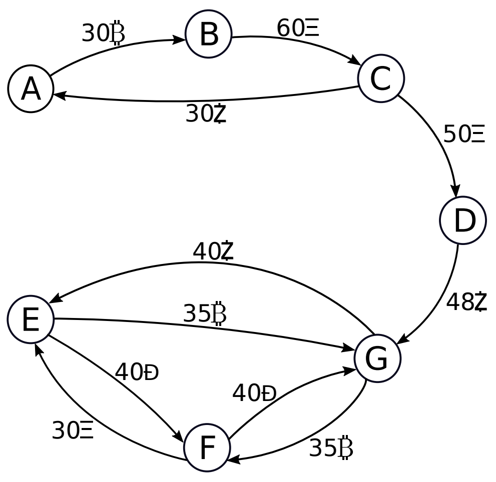
(b) Equivalent transformation of the transaction (a)

Fig. 14. Fifth Use-case

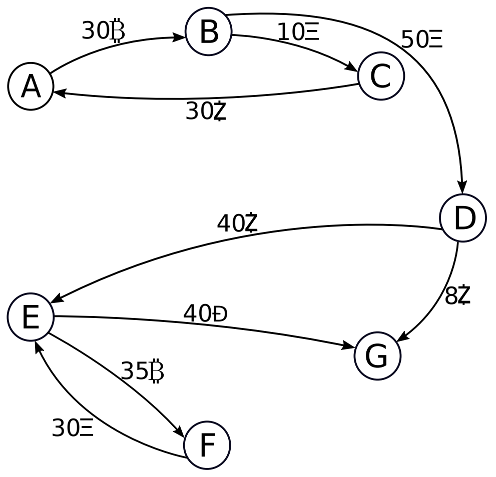
Leaders	Synthesis Time	Gas Consumption				
	(ms)	Deploy	Initiate	Redeem	Claim	Total
A, D	318	1494188	80018	79636	59664	1713506
		1494252	80018	79636	59664	1713570
		1494056	80018	79636	59664	1713374
		1494252	80018	79636	59664	1713570
		1494252	80018	79636	59664	1713570
		1494252	80018	79636	59664	1713570

TABLE V
EXECUTION OF FIFTH USE-CASE

F. Use-case 6



(a) Transaction with 7 nodes



(b) Equivalent transformation of the transaction (a)

Fig. 15. Sixth Use-case

Leaders	Synthesis Time	Gas Consumption				
	(ms)	Deploy	Initiate	Redeem	Claim	Total
A, F	343	1534206	80552	79754	61190	1755702
		1534214	80574	79688	61212	1755688
		1534818	80574	79694	61212	1756298
		1534282	80596	79644	61212	1755734
		1535886	80574	79700	61212	1757372
		1534882	80574	79694	61212	1756362
		1534882	80574	79694	61212	1756362
		1535954	80574	79700	61212	1757440
		1535086	80574	79694	61212	1756566

TABLE VI
EXECUTION OF SIXTH USE-CASE

G. Use-case 7

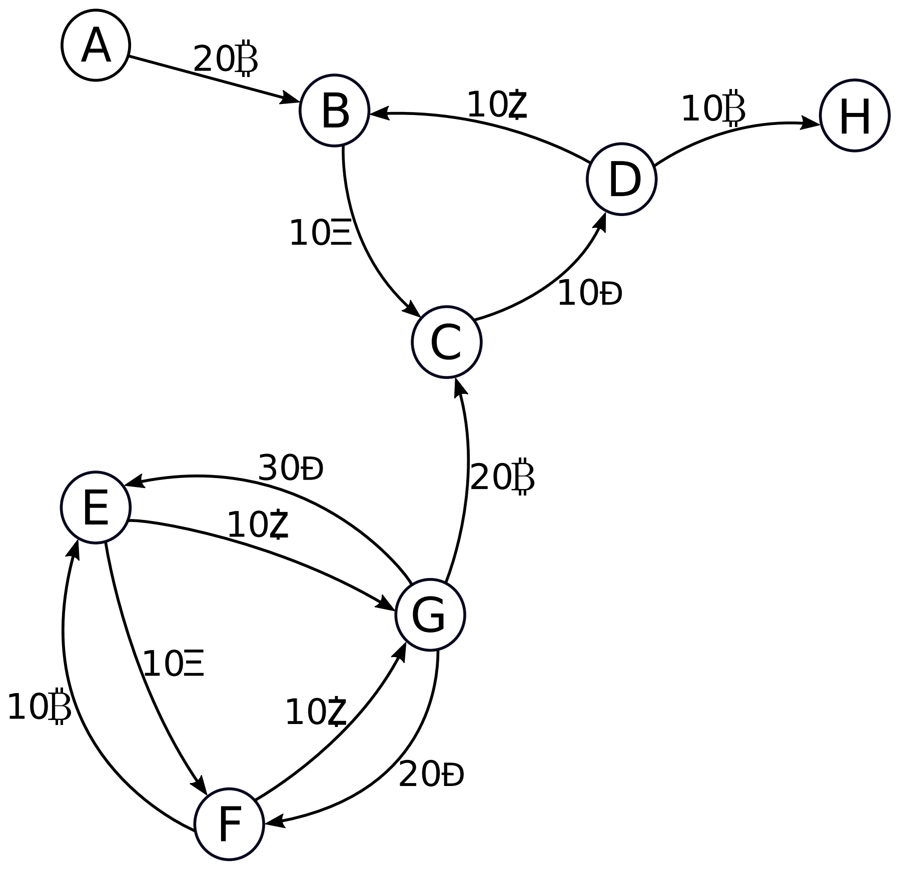


Fig. 16. Seventh Use-Case

Leaders	Synthesis Time (ms)	Gas Consumption				
		Deploy	Initiate	Redeem	Claim	Total
A, B, E, F	351	1788818	81192	144424	61574	2076008
		1994975	81192	144566	61530	2282263
		1995371	81214	144478	61596	2282659
		1994975	81192	144544	61530	2282241
		1995371	81236	144478	61618	2282703
		1788550	81192	144424	61574	2075740
		1995375	81236	144478	61618	2282707
		1995375	81214	144434	61596	2282619
		1788546	81192	144424	61574	2075736
		1788278	81192	144360	61574	2075404
		1788814	81192	144424	61574	2076004
		1789490	81170	144496	61552	2076708

TABLE VII
EXECUTION OF SEVENTH USE-CASE

H. Extensions

Sample of Smart Contract

```
pragma solidity ^0.4.15;

contract AtomicSwap {

   address private counterParty;
   bytes20[] private hashedSecret;
   uint private delta;
   uint graphDiam;
   bool[] unlocked;
   bool[] leaders;

   uint initTimestamp;
   bytes32 secret;
   address party;
   uint256 value;
   bool emptied;
   uint amount;

   event Initiated(
      uint _initTimestamp,
      uint _delta,
      bytes20[] _hashedSecret,
      address _counterParty,
      address _party,
      uint256 _funds
   );

   function AtomicSwap(){
      hashedSecret = new bytes20[](8);
      unlocked = new bool[](8);
      leaders = new bool[](8);
      counterParty = 0
           x14723a09acff6d2a60dcdf7aa4aff308fddc160c;
      party = 0xca35b7d915458ef540ade6068dfe2f44e8fa733c;
      amount = 30 ether;
      hashedSecret[1] = 0
           x1c301c2b29511c607b02d7be6391e168f460a44a;
      hashedSecret[2] = 0
           x55f47027d4a971ca4505ed0df51030f3d5e81a96;
      hashedSecret[3] = 0
           x508afa12d5bb90c39df2e2cb7d7b6219c1100edb;
      hashedSecret[4] = 0
           xdad9d738012b6669a58a51227fd7f1367b2d39a2;
      hashedSecret[5] = 0
           x219568abcb139fd0f226b7734c7abe86d2121a25;
      hashedSecret[6] = 0
           xf1716c9b82cfcf38dd3ab19d782161d37741587e;
      hashedSecret[7] = 0
           x932e1f2851e4f3696402de839c6e8f2de83b4b94;
      leaders[0] = false;
      leaders[1] = true;
      leaders[2] = false;
      leaders[3] = false;
      leaders[4] = false;
      leaders[5] = false;
      leaders[6] = true;
      leaders[7] = false;
      delta = 7200;
      graphDiam = 4;
   }
```

Fig. 17. Smart Contract for an edge in Solidity (Part 1).

```
   modifier isRefundable() {
      require(emptied == false);
      _;
   }

   modifier isInitiator(uint _i) {
      require(msg.sender == party);
      _;
   }

   modifier isCorrectValue(){
      require(msg.value == amount);
      _;
   }

function initiate () isCorrectValue public payable {
   initTimestamp = block.timestamp;
   party = msg.sender;
   value = msg.value;
   emptied = false;
   Initiated(
      initTimestamp,
      delta,
      hashedSecret,
      counterParty,
      msg.sender,
      msg.value
      );
   }

function redeem2_3_1(bytes32 _secret, bytes _sig) public {
   if(msg.sender == counterParty){
      if(Verify("2-3-1", _sig) &&
         ripemd160(_secret) == hashedSecret[1] &&
         block.timestamp < initTimestamp + (graphDiam + 3) *
               delta)
      {
         unlocked[1] = true;
      }
   }
}
function redeem2_4_7_6(bytes32 _secret, bytes _sig) public {
   if(msg.sender == counterParty){
      if(Verify("2-4-7-6", _sig) &&
        ripemd160(_secret) == hashedSecret[6] &&
        block.timestamp < initTimestamp + (graphDiam + 4) *
              delta)
      {
        unlocked[6] = true;
      }
   }
}
```

Fig. 18. Smart Contract for an edge in Solidity (Part 2)..

```
    function claim() public{
       bool lock = false;
       if(msg.sender == counterParty){
          for(uint i=0; i<unlocked.length; i++){
             if(unlocked[i] == false && leaders[i] == true){
                lock = true;
                break;
             }
          }
          if(lock == false){
             counterParty.transfer(value);
             emptied = true;
          }
       }
    }

    function refund() public isRefundable() {
        bool lock = false;
        if(msg.sender == party){
           for(uint i=0; i<unlocked.length; i++){
             if(unlocked[i] == false && leaders[i]==true){
                lock = true;
                break;
             }
           }
           if(lock && block.timestamp > initTimestamp + (
               graphDiam + 1 + 3) * delta){
              party.transfer(value);
              emptied = true;
           }
        }
    }
}
```

Fig. 19. Smart Contract for an edge in Solidity (Part 3).