Cross-L2 Bridge

Hashed Timelock Contract vs Delayed-Proved Timelock Contract

Qi Zhou

Jan. 6, 2022

Problem

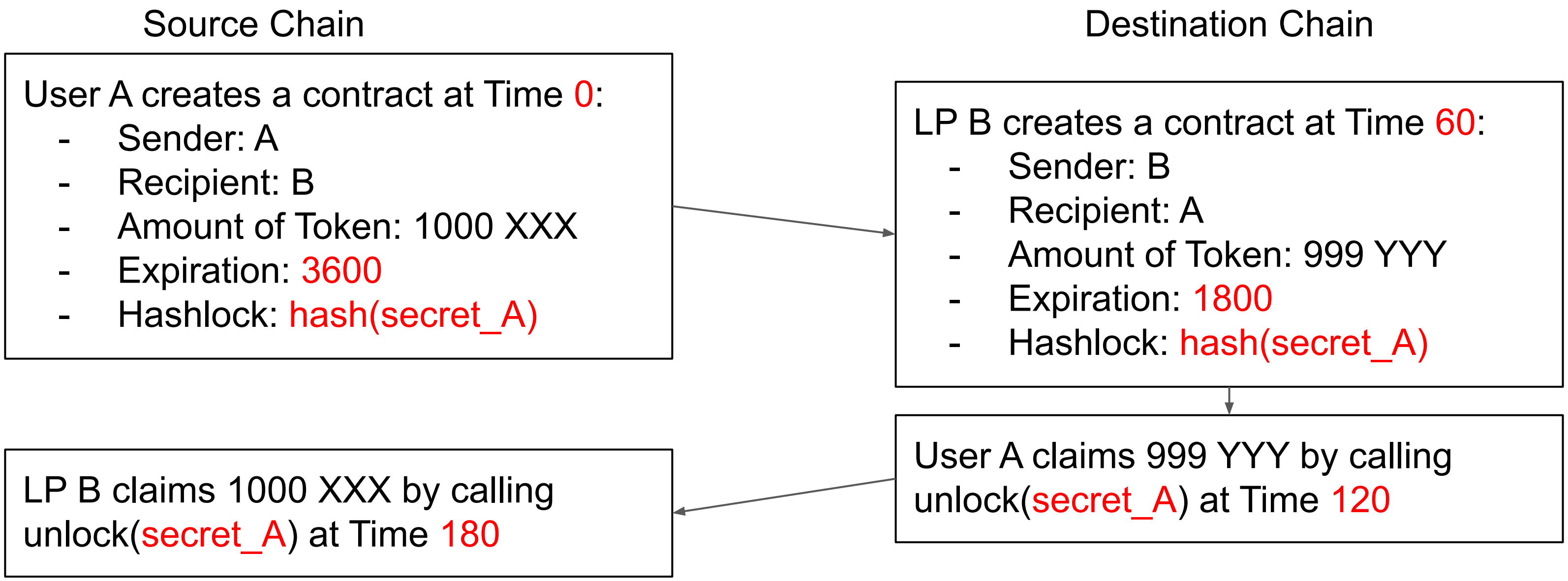
- Transfer a token from an L2 to another L2 can be costly and time consuming
    - E.g., Arbitrum to Optimism
    - Route the token with Arbitrum -> ETH -> Optimism
    - Cost: Arbitrum withdraw + ETH withdraw + ETH deposit (generally pays Optimism withdraw)
        - ETH withdraw and deposit may be combined
    - Time: Arbitrum -> ETH takes 7 days
- Goal:
    - Speed up transfer
        - Almost instantly
    - Less cost
        - User: one-time payment at source
    - Only rely on security between L1<->L2

Hashed Timelock Contract

- A contract with the following information
    - sender
    - recipient
    - amount of token
    - expiration
    - hashlock=hash(secret_A)
- Anyone can create the contract by generating a secret and supplying the token
    - The token will be locked in the contract
- Token can be unlocked if
        - hash(preimage) == hashlock by *recipient* with *preimage*; or
        - now > expiration by *sender*
- https://github.com/Dapp-Learning-DAO/Dapp-Learning/tree/main/basic/63-htlc-crosschain

Example: Normal Path

- User A wishes to swap 1,000 XXX on source chain with LP B for 999 YYY on destination chain with a secret only known by A at the beginning



Source Chain

User A creates a contract at Time 0:

- Sender: A
- Recipient: B
- Amount of Token: 1000 XXX
- Expiration: 3600
- Hashlock: hash(secret_A)

Destination Chain

LP B creates a contract at Time 60:

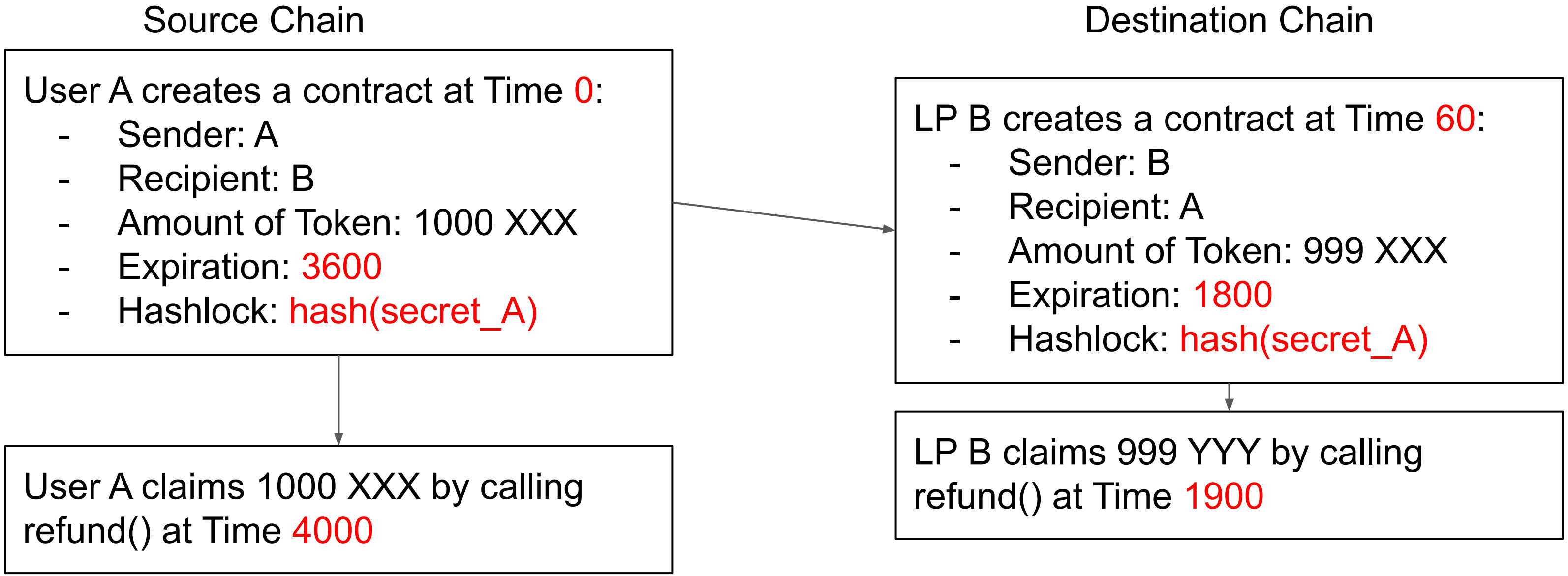
- Sender: B
- Recipient: A
- Amount of Token: 999 YYY
- Expiration: 1800
- Hashlock: hash(secret_A)

User A claims 999 YYY by calling unlock(secret_A) at Time 120

LP B claims 1000 XXX by calling unlock(secret_A) at Time 180

Example: Bad Path

- User A wishes to swap 1,000 XXX on source chain with LP B for 999 YYY on destination chain with a secret only known by A at the beginning



Source Chain

User A creates a contract at Time 0:

- Sender: A
- Recipient: B
- Amount of Token: 1000 XXX
- Expiration: 3600
- Hashlock: hash(secret_A)

User A claims 1000 XXX by calling refund() at Time 4000

Destination Chain

LP B creates a contract at Time 60:

- Sender: B
- Recipient: A
- Amount of Token: 999 XXX
- Expiration: 1800
- Hashlock: hash(secret_A)

LP B claims 999 YYY by calling refund() at Time 1900

Pros and Cons of HTLC

- Pros:
  - Security: P2P, no pooled custodian
  - Normal path: almost instantly
  - Work on two independent blockchains

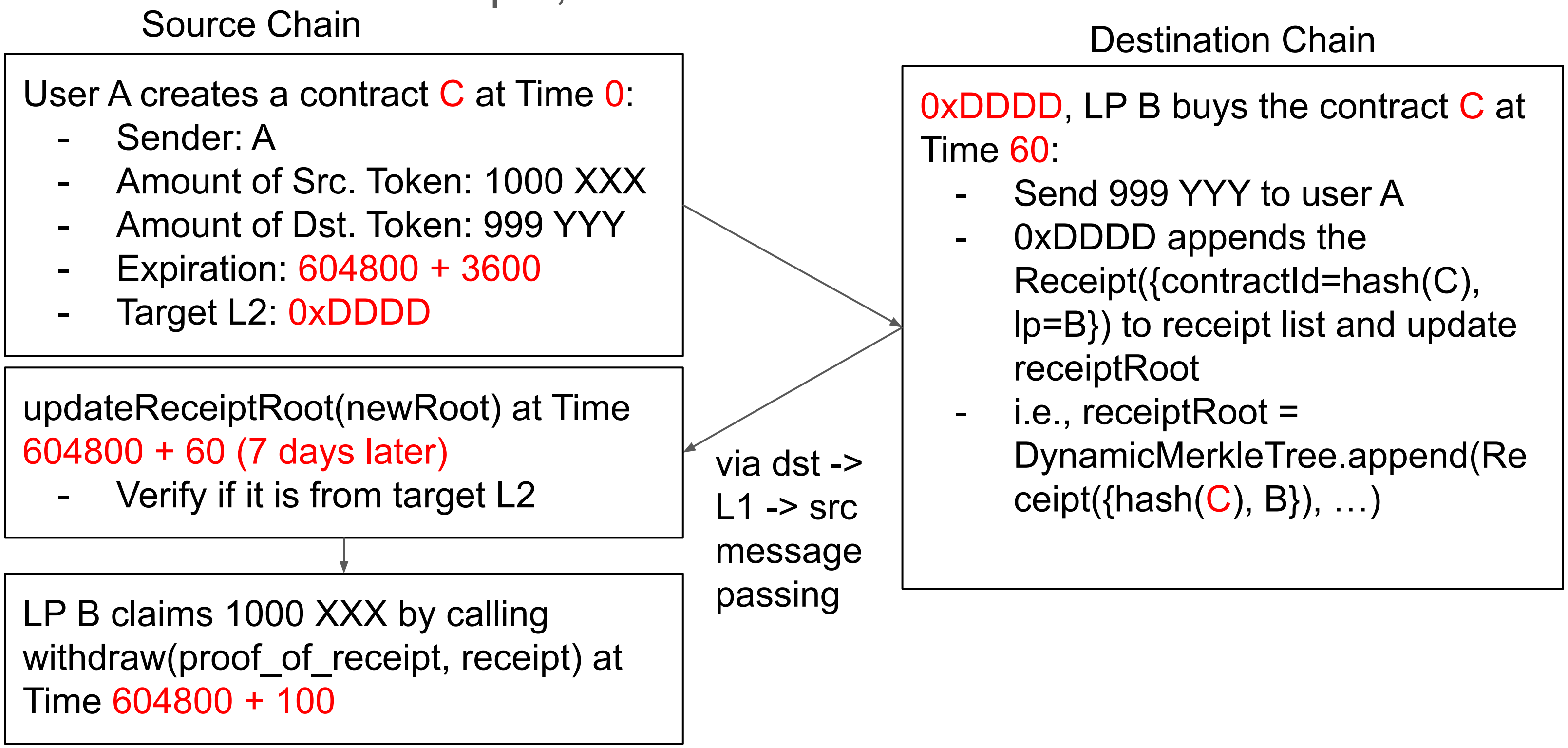
- Cons:
  - User needs to find an LP off-chain
    - An off-chain match engine is required
  - User needs to submit two transactions per transfer
  - Cannot support smart-contract-initiated transfer (as a contract has no secret)
  - Require LP to sufficient security assurance on L2
    - E.g., tx on L2 may be reverted or challenged

Delayed-Proved Timelock Contract

- User creates a contract with
    - sender
    - ~~recipient~~
    - amount of source token
    - amount of destination token
    - expiration
    - ~~hashlock~~
    - target L2 contract address
- Anyone/smartcontract can create a contract by supplying the source token
- Token can be unlocked only if
    - now > expiration by *sender*; or
    - proof from target L2 contract by an *LP* that the transfer contract is bought at destination chain
        - i.e., an LP offers destination token to sender
- Credits: @vbuterin, @amritkumarjain, @qizhou, etc
- https://gitcoin.co/issue/gitcoinco/skunkworks/253/100027342

Example: Normal Path

- User A wishes to swap 1,000 XXX on source chain for 999 YYY on dst chain



Source Chain

User A creates a contract C at Time 0:

- Sender: A
- Amount of Src. Token: 1000 XXX
- Amount of Dst. Token: 999 YYY
- Expiration: 604800 + 3600
- Target L2: 0xDDDD

updateReceiptRoot(newRoot) at Time 604800 + 60 (7 days later)

- Verify if it is from target L2

LP B claims 1000 XXX by calling withdraw(proof_of_receipt, receipt) at Time 604800 + 100

via dst -> L1 -> src message passing

Destination Chain

0xDDDD, LP B buys the contract C at Time 60:

- Send 999 YYY to user A
- 0xDDDD appends the Receipt({contractId=hash(C), lp=B}) to receipt list and update receiptRoot
- i.e., receiptRoot = DynamicMerkleTree.append(Receipt({hash(C), B}), …)

Demo on Testnet

- See https://github.com/QuarkChain/DynamicMerkleTree/
-

Pros and Cons of DPTLC

- Pros:
  - Instant transfer with single user transaction
  - No off-chain matching (instead, using reverse auction)
- Cons:
  - LP takes long time to complete

- Improvement:
  - Instead of pushing L2 -> L1 and waiting 7 days, L1 directly reads unconfirmed block/tx result (with some finality criterion) and sends the receipt root hash to source chain directly
    - E.g., Optimism will send the world state root of per post-tx on L1
  - Issue: L2-on-L1 data is not cross-chain friendly
    - Proof via world state root can be very large
    - Receipt root is preferred but currently optimism has no receipt root on L1

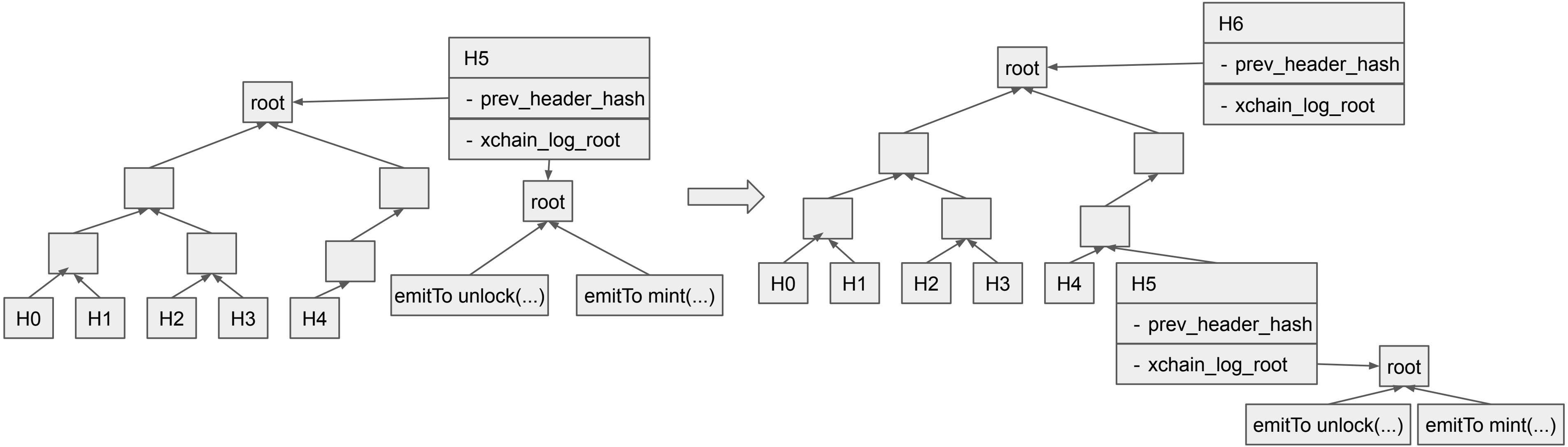
Further Thoughts

- Combined HTLC/DPTLC

- Bridge-friendly blockchain structure

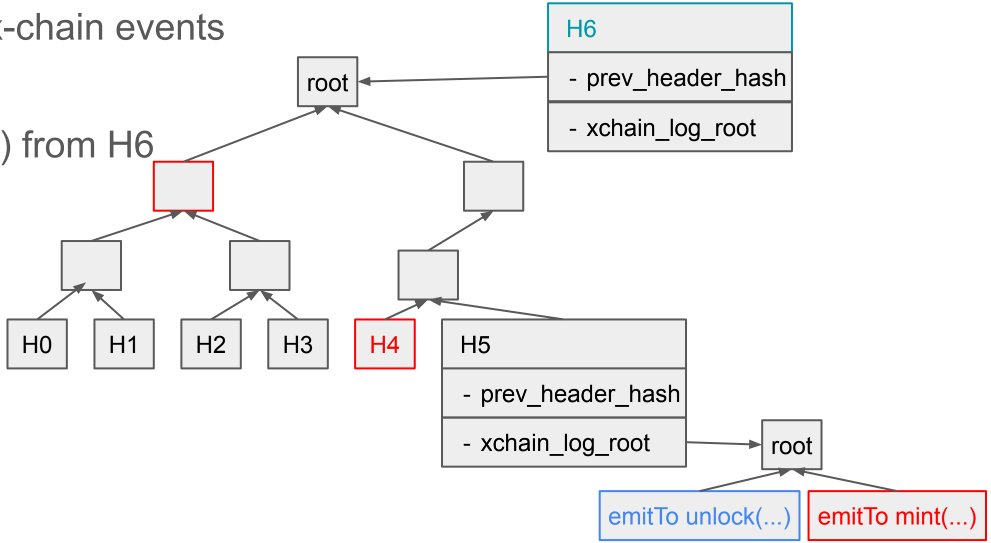
Further Thoughts (Bridge-Friendly Blockchain Structure)

- Bridge-friendly blockchain: Easy to prove any on-chain events with low cost
  - PoS + BFT with ECDSA multi-sign
  - header = [prev_header_hash, height, xchain_log_root, metadata_hash, ECDSA_multi_sign]
    - prev_header_hash = merkle_root([headers])
    - xchain_log_root = merkle_root([emitTo eventName(args…)])



Further Thoughts (Bridge-Friendly Blockchain Structure)

- Bridge-friendly blockchain: Easy to prove any on-chain events with low cost
  - Proof of any x-chain event will be:
    - proof_of_header=merkle_proof(prev_header_hash, header)
      - up to log2(N)*32 bytes for the header
    - proof_of_event=merkle_proof(header.xchain_log_root, xchain_log)
      - up to log2(N)*32 bytes for x-chain events
- Example:
  - Proof (red) of H5's unlock() event (blue) from H6


H6
- prev_header_hash
- xchain_log_root
root
H0
H1
H2
H3
H4
H5
- prev_header_hash
- xchain_log_root
root
emitTo unlock(...)
emitTo mint(...)