Rift

Trustless Swaps Between Bitcoin and Ethereum

Clifford Syner
cliff@rift.trade

Tristan Barrett
tristan@rift.trade

Samee Siddiqui
samee@rift.trade

Dan Robinson
dan@paradigm.xyz

ABSTRACT

Satoshi's original vision was to build a peer-to-peer protocol for transferring value between any two people in the world. However, over 90% of trading volume today occurs on centralized exchanges due to the fragmentation of assets across different blockchains. Trustless cross-chain swaps would realize the original vision of crypto as a replacement for these centralized financial institutions. We propose Rift, a protocol that enables trustless swaps between Bitcoin and Ethereum by unlocking ERC20s to a market maker upon verifying a zero-knowledge proof that they paid a user a specified amount of BTC. In addition to the core escrow-unlock mechanism, we outline an open auction system that matches users with the market maker providing the best exchange rate for their swap.

1. INTRODUCTION

Bitcoin [Nak08] and Ethereum [But14] are the two largest blockchains by market cap, yet there is no way to trade between them without relying on trusted third parties with lower security guarantees than the source and destination chains. Trustless cross-chain swaps have been achieved between EVM chains [Xie+22], but swaps between Bitcoin and other VMs are much more complex due to Bitcoin's lack of smart contracts and the prohibitively high gas cost of verifying the state of Bitcoin on another chain using a standard light client. Users who want to trade native BTC for ERC20s [Vog+15] today have five options, all of which suffer from the same core problem of being far less secure and decentralized than the underlying networks that users hold assets on:

1. Centralized Exchanges (CEXs) - (Binance, Coinbase, ChangeNow, etc.) are controlled by single entities and have historically struggled to manage user funds, resulting in multiple bankruptcies and financial losses (e.g. Bybit [IC325], FTX [Inv23], Celsius [Cel22]). CEXs are also highly regulated, force KYC, rate-limit swapping, and can freeze customer funds at will.

2. Over-The-Counter Trading Desks - (Galaxy, Flow, Wintermute, FalconX, GSR, etc.) are able to offer better rates than CEXs for large trades, but they are fully centralized and inherit the same security risks.

3. Alternative Layer 1 Blockchains - (THORchain [Tho20], Chainflip [Har+24], NEAR Chain Abstraction [Nea24], etc.) are blockchains used for message passing and/or trade execution between source and destination chain. They are far less decentralized than Bitcoin and Ethereum [Eth25, MPS25, CW25], and have far less capital staked [Tho25] in the network, leading to lower capital efficiency, economic security, and swapping limits for users.

4. Hashed Timelock Contracts - (Garden Finance [Gar24]) in their purest form remove the need for a custodian, but require a two way handshake that depends on both parties being online to complete a swap. In practice, this necessity for simultaneous interaction significantly limits usability, leading production implementations to rely on centralized signing services to manage these interactions. Despite having higher security guarantees than CEXs and OTC desks, users are still exposed to similar censorship and regulatory risks.

5. **Bridged Bitcoins** - (wBTC [Wbt19], cbBTC [CB24], tBTC [Thr24], renBTC, etc.) are a useful primitive, but custody Bitcoin in a high-risk N-of-M multisig wallet where if N keys are compromised, all collateral can be stolen [Coi22]. Additionally, since the multisig ownership is often centralized, they inherit the same censorship and regulatory risks as CEXs. This has already resulted in the collapse of renBTC, which once held >$1b in collateral [Ren23].

In addition to the systems above, there were a number of light-client-based designs that retained higher security guarantees at the cost of increased transaction fees. While innovative, this tradeoff ultimately made them infeasible to run sustainably in production.

1. **Stateful Light Clients** - (BTCRelay [Cho16], zkRelay [Wes+20]) eliminate the need for a custodian by implementing an on-chain light client with a Bitcoin SPV proof. Solutions such as BTCRelay proved to be prohibitively expensive, as they required continuously submitting Bitcoin headers to an Ethereum smart contract. Zero-knowledge (ZK) based solutions, such as zkRelay, solved this with batched block proofs that could be verified in a single on-chain transaction, but could never enter production due to the complexity of managing trusted setups of circuits for each block batch size.

2. **Stateless Light Clients** - (Summa [Pre18]) present a sliding window of Bitcoin block headers, rather than storing every header in contract. While eliminating the need for relayers to keep the light client constantly up to date, transactions are actually more expensive if usage is high, due to re-verification of the same set of blocks. Summa also outlined a novel cross-chain auction mechanism, but it was limited to receiving BTC, and also required users to have >0 BTC in order to start an auction.

With security via self-custody and decentralization as the core ethos of blockchain technology, and the increasing demand for swaps between Bitcoin and other ecosystems, the need for a trustless cross-chain swapping protocol is more apparent than ever.

2. SOLUTION

Rift implements an on-chain Dutch auction with a novel Bitcoin ZK light client and payment verification scheme, enabling users to swap ERC20s for native BTC without custodians. A user creates an `Order` by depositing an ERC20 into the auction contract and specifying a list of market makers (MMs) who can bid for their asset. This auction can theoretically be for any ERC20, but to simplify quoting, we'll assume a bridged Bitcoin variant, such as cbBTC, for the remainder of this paper. Once the price reaches a profitable threshold for a MM, they call `claimAuction` to lock the user's ERC20 and gain an exclusive right to complete the swap until the lockup expires. A "free option" is prevented here by using a bridged Bitcoin as the escrowed asset, ensuring the option value is negligible. The MM then pays the user BTC to fill the `Order` and start the settlement process.

After BTC is sent, anyone can generate and submit a ZK proof of this payment by calling `submitPaymentProofs`, providing the proposed transaction and proof of a valid Bitcoin state change to the light client contract. The protocol verifies four invariants before an `Order` is considered valid:

1. **The proposed block is part of the longest Bitcoin chain** - verified by ensuring each new block since the last light-client update follows Bitcoin's Proof-of-Work consensus mechanism, and that the proposed block has at least N confirmation blocks built on top of it.

2. **The proposed block contains the MM's transaction to the user's wallet address** - verified with a Bitcoin SPV proof.

3. **The user was paid the correct amount of BTC** - verified by ensuring that there exists a UTXO in the MM's transaction equal to the satoshis specified by the user.

4. **The transaction has the order data inscribed** - verified by confirming the order hash is a member of the expected orders set, and the hash of this set is inscribed using `OP_RETURN` [Bit25].

Once a payment proof is verified, there is a short challenge period where heavier work chains can be presented before funds are released from escrow. This is meant to prevent a malicious user from exploiting a significantly out-of-date or "stale" Bitcoin light client. Assuming no challenge, `settleOrders` is called, releasing the user's ERC20 to the MM. The new Bitcoin state, represented by a Merkle Mountain Range (MMR) root [Tod16], is stored in the light client smart contract to be used as a circuit input for future payment verifications. If a longer chain is presented during the challenge period, a heavier work chain containing the MM's transaction must be presented before the lockup, otherwise the user will be refunded.

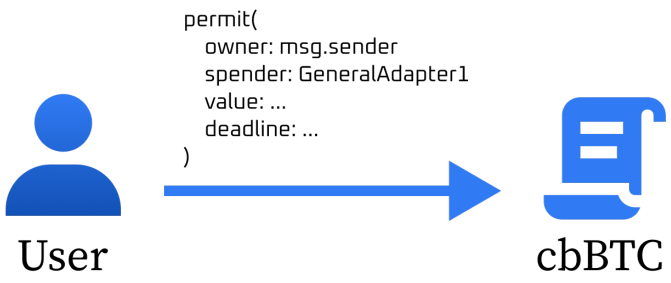
Assuming the light client state remains relatively up to date, we can now perform a trustless swap between the chains, since faking a payment would require having enough hash power to out-mine the Bitcoin network.

3. SWAP FLOW

The core protocol deals with auctioning off a bridged Bitcoin ERC20 for BTC. While focusing on a single asset is most capital efficient from an MM's perspective, many users want to trade arbitrary ERC20s for BTC. By leveraging on-chain liquidity (with DEX aggregators, such as Paraswap [Vel25]) and multicall contracts (like `Bundler3` [Mor25]), it's possible to support this flow with no additional signatures by the user. The possible swap flows are outlined below.

3.1 USER STARTS WITH BRIDGED BITCOIN

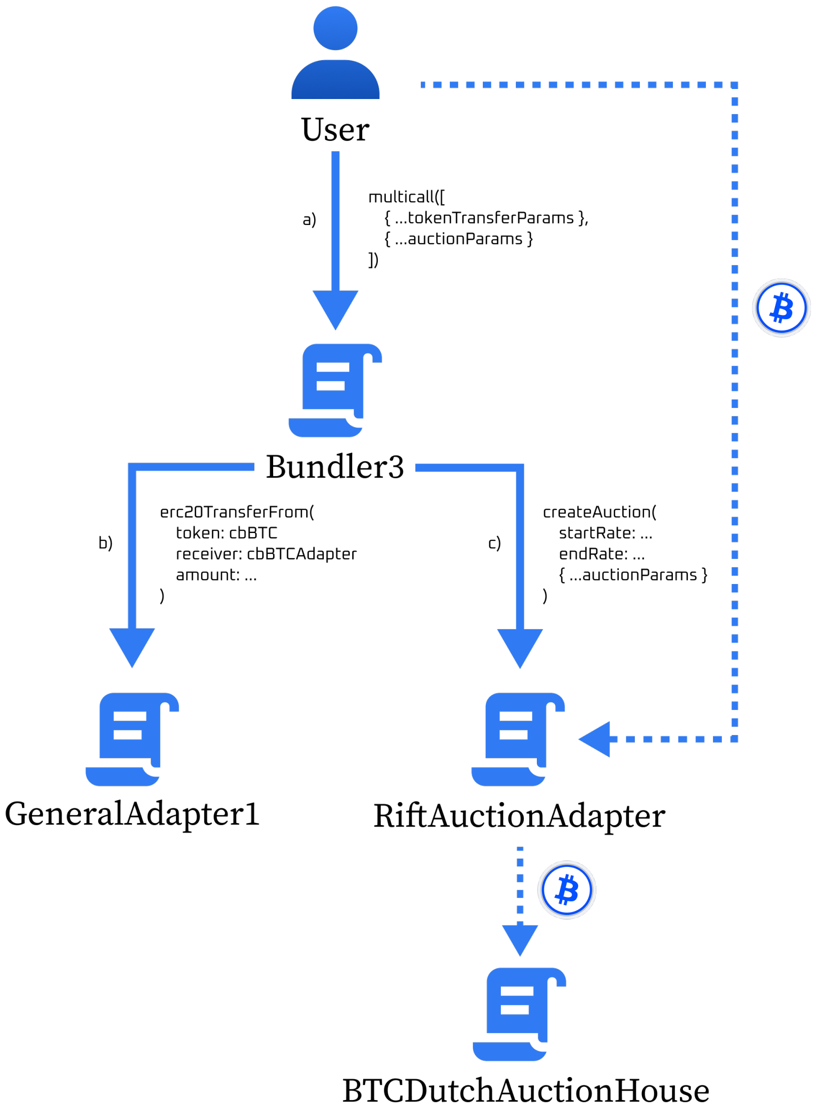
1. A user signs a cbBTC permit to `GeneralAdapter1`, allowing it to transfer cbBTC on the user's behalf.



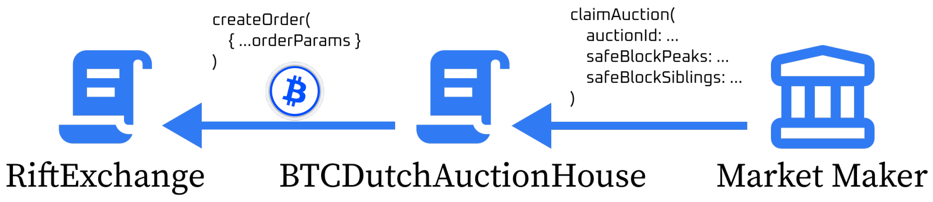
2a. The user then initiates an atomic multicall transaction using `Bundler3`, passing auction params.

2b. `GeneralAdapter1` transfers cbBTC from the user to `RiftAuctionAdapter`.

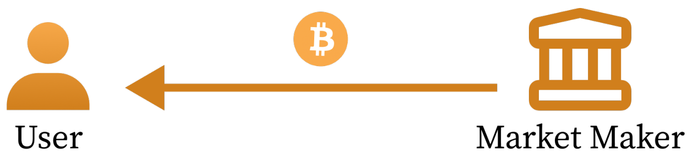
2c. `RiftAuctionAdapter` deposits the user's cbBTC after calling `startAuction` on `BTCDutchAuctionHouse`.



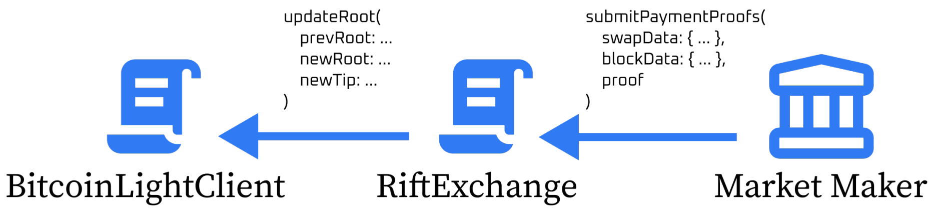
3. Once the auction reaches the target price for a MM, and they call `claimAuction` to lock the user's tokens in `RiftExchange`.



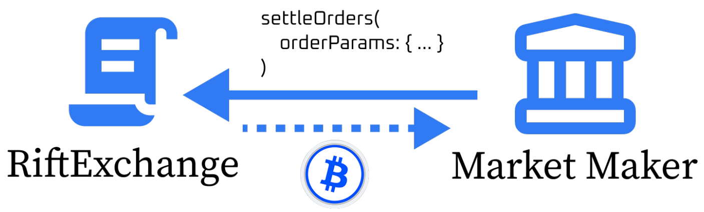
4. The MM sends BTC to the user.



5. After b Bitcoin confirmation blocks, the MM submits the proof to `RiftExchange`, updating the light client MMR root and initiating the challenge period.

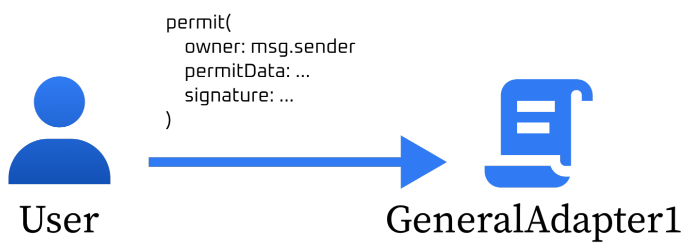


6. After the challenge period ends, the MM calls `settleOrders` to unlock the escrowed user funds, minus a fee to the protocol.



3.2 USER STARTS WITH AN ARBITRARY ERC20

1. A user signs an ERC20 permit2 [Uni22] to `GeneralAdapter1`.

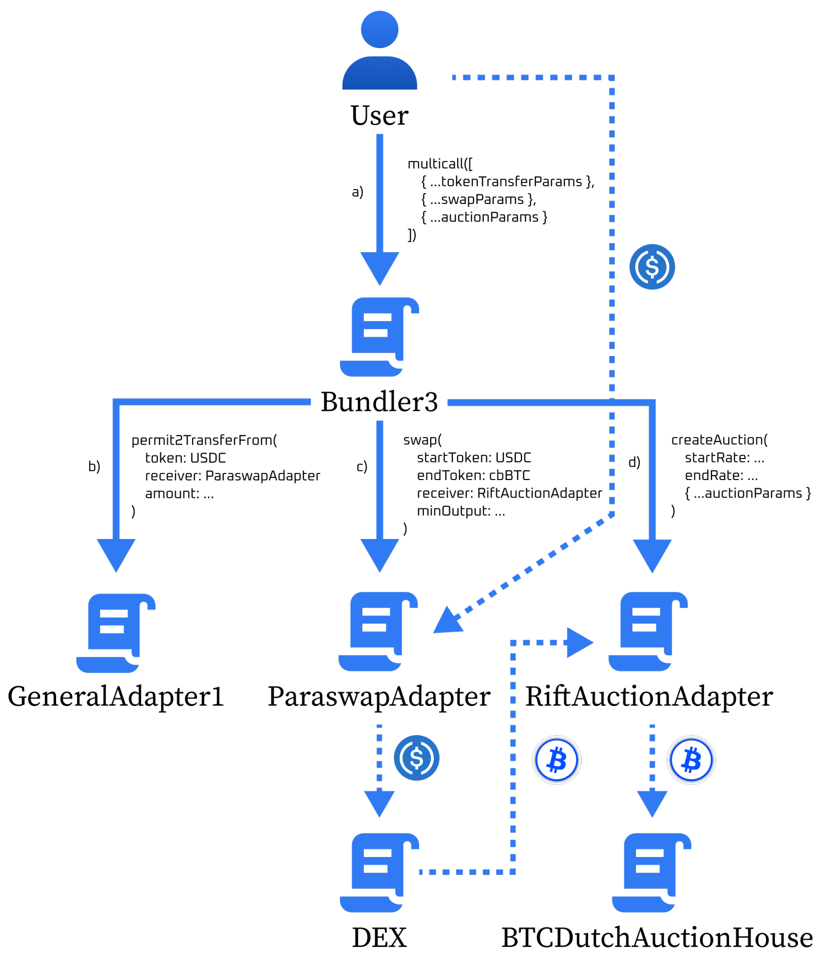


2a. The user then initiates an atomic multicall transaction using `Bundler3`, passing a swap route and auction params.

2b. `GeneralAdapter1` transfers the ERC20s from the user to `ParaswapAdapter`.

2c. `ParaswapAdapter` swaps the ERC20s for cbBTC and sends the tokens to `RiftAuctionAdapter`.

2d. `RiftAuctionAdapter` deposits the user's cbBTC after calling `startAuction` on `BTCDutchAuctionHouse`.

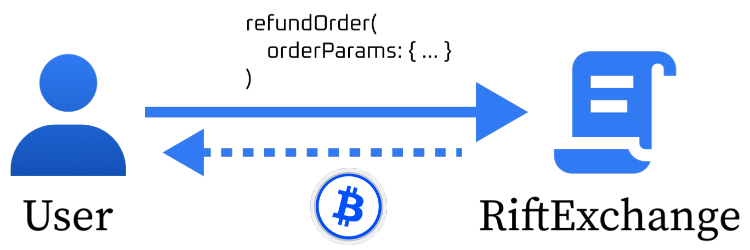


3. Continue from step 3 of bridged Bitcoin flow.

3.3 MARKET MAKER NEVER FILLS ORDER

1-3. Start from bridged Bitcoin flow.

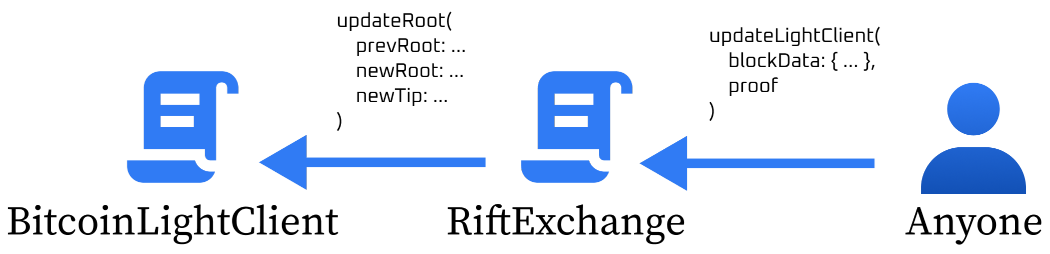
4. After the lockup expires, `refundOrder` is called, returning cbBTC to the user.



3.4 LONGER CHAIN IS PRESENTED DURING CHALLENGE PERIOD

1-5. Start from bridged Bitcoin flow.

6. A heavier work chain is presented to the light client, causing the MMR root to be updated. If the new chain includes the MM's payment block, no action is needed. If the new chain does not include the payment block, the MM has the remainder of the lockup period to submit a transaction and light client update of a heavier work chain.



4. PROTOCOL

The protocol consists of three solidity smart contracts, and two ZK circuits that are aggregated into a single proof using SP1 [Suc24]. The components and their functionality are outlined below.

4.1 BITCOIN AUCTION HOUSE CONTRACT

`BTCDutchAuctionHouse` is where the user's cbBTC is auctioned off to a MM. `startAuction` is called as the last step of the `Bundler3` multicall, passing in the Dutch auction starting and ending range, the duration in blocks, and the MMs that can participate in the auction.

```
struct DutchAuctionParams {
  uint256 startBtcOut; // The starting amount of BTC the auction will sell
  uint256 endBtcOut; // The ending amount of BTC the auction will sell
  uint256 decayBlocks; // The number of blocks price will decay over
  uint256 deadline; // The deadline of the auction (as a timestamp)
  address fillerWhitelistContract; // The whitelist contract to use for
    validating the filler
}
```

Once the rate reaches a profitable threshold for an online MM, and they can create an `Order` by calling `claimAuction`, passing their payout address and safe block data specified by the user's leaf and roots in the original auction parameters. This will create an `Order` and escrow the funds in `RiftExchange`. If the auction reaches the deadline without being filled, it becomes expired and the user can withdraw their funds using `refundAuction`.

4.2 RIFT EXCHANGE CONTRACT

`RiftExchange` is responsible for managing Orders from creation to settlement. An `Order` is created when a MM expresses intent to swap at a specified exchange rate by calling `claimAuction` on `BTCDutchAuctionHouse`, resulting in the user's cbBTC being escrowed in `RiftExchange` until the MM actually fills the order.

```
struct Order {
  uint256 index; // Where in the order hash array this order is
  uint64 timestamp; // When the order was created
  uint64 unlockTimestamp; // When the order can be refunded (ie. No valid
    payment was submitted)
  uint256 amount; // Amount of ERC20 tokens to be transferred to `
    designatedReceiver` upon settlement
  uint256 takerFee; // The taker fee prepaid by the order creator, given
    to the protocol upon settlement
  uint64 expectedSats; // The expected amount of satoshis that MUST be
    sent to `bitcoinScriptPubKey` before the order can be settled
  bytes bitcoinScriptPubKey; // The scriptPubKey of the address that will
    receive the BTC output on the Bitcoin chain
  address designatedReceiver; // The address that will receive the ERC20
    tokens upon settlement
  address owner; // The address that created the order
```

```
    bytes32 salt; // A random number used to seed the order hash, to prevent
      replays of previous payments
    uint8 confirmationBlocks;  // The number of blocks that must be built on
      top of the block containing the payment before the order can be
     settled
    uint64 safeBitcoinBlockHeight; // Historical Bitcoin block height
     considered safe from reorganization by the order creator
    OrderState state; // The state of the order (`Created`, `Settled` or `
     Refunded`)
  }
```

Once the MM sends Bitcoin to the user, they submit a proof of the payment by calling `submitPaymentProofs`, passing the proof data and the block data needed for the light client update. Assuming the payment proof and light client update are valid, the challenge period kicks off, after which the MM can claim their funds by calling `settleOrders`. If the MM never fills, the user can call `refundOrder` to withdraw their funds after the lockup period ends. If there is a longer chain presented that no longer includes the payout transaction to the user, it can be submitted during the challenge period by calling `updateLightClient`, requiring the MM to submit a new chain to start a new challenge period and claim attempt. `settleOrders`, `refundOrder`, `submitPaymentProofs`, and `updateLightClient` can be called by anyone, although no one other than the affected party is incentivized to do so. For convenience, we call these on behalf of users when the relevant condition applies. Users who don't want to rely on Rift infrastructure being online to safely interact with the protocol can run their own `Hypernode` [Rif25], which is an open-source node software that anyone can use to index Bitcoin and generate proofs.

4.3 BITCOIN LIGHT CLIENT CONTRACT

`BitcoinLightClient` is a stateful light client that utilizes an MMR tree to store the current Bitcoin state. In the MMR tree, each Bitcoin block is a leaf that is combined through "peak bagging" to generate the MMR root. The contract stores the current `mmrRoot`, and a mapping of all previous "checkpoint" blocks, which are the chain tips each time the `mmrRoot` was updated. An example checkpoint block is below:

```
  struct BlockLeaf {
     bytes32 blockHash; // The hash of the Bitcoin block
     uint32 height; // The height of the Bitcoin block
     uint256 cumulativeChainwork; // The cumulative chainwork of the Bitcoin
       block
  }
```

`RiftExchange` calls `updateRoot` whenever there is a light client update, which verifies that the new `tipBlockLeaf` has greater chain work than the latest checkpoint block before updating the root. As part of this function call, a valid light client state transition proof must be presented, otherwise the entire `updateLightClient` call will revert. In addition to updating the light client, `RiftExchange` calls `verifyBlockInclusion` on `Order` creation that verifies that the safe block is part of the chain represented by the current `mmrRoot`. When the `Order` is being settled, `verifyBlockInclusionAndConfirmations` checks the same inclusion for the MM's payment block, in addition to confirming it has at least `N` confirmation blocks built on top of it.

4.4 LIGHT CLIENT STATE TRANSITION CIRCUIT

The Light Client State Transition Circuit works in tandem with `BitcoinLightClient` to ensure block updates submitted by a prover follow the consensus rules of Bitcoin. This is done through a series of invariant checks in the circuit:

1. The `sha256(sha256(header))` for `parent` and `retarget` blocks matches the hash stored in the corresponding `BlockLeaf` of the current MMR tree.
2. The chain of headers from the `parent` block to the `new_tip` block follows Bitcoin consensus rules (references previous blocks and satisfies proof of work).
3. The chain work of the `new_tip` is greater than the `current_tip`.
4. The `parent`, `retarget`, and `current_tip` blocks are part of the current MMR tree.

Under most circumstances, `parent` and `current_tip` will be the same block. In the case of a reorg, parent is the block that the proof is being built upon, and everything until current will be a disposed leaf of the MMR tree. In the case of a reorg, there are two additional invariants:

1. `disposed_leaves_count == current_tip.height - parent.height`
2. The MMR root at the `parent` block and corresponding `parent_peaks` produce the `current_root` after appending all disposed leaves.

Once these invariants are satisfied, each new leaf is appended to the parent to produce the `new_root`. All new leaves are then combined into a single hash that the prover passes in calldata when `verifyBlockInclusion` is called to verify the existence of the leaf data. The final state change is then returned to commit to the witness data.

```
public_input = {
  current_root: [u8; 32], // current MMR root
  new_root: [u8; 32], // new MMR root
  new_leaves_hash: [u8; 32], // hash of new leaves between current and new
     root
  new_tip: BlockLeaf // new tip block of Bitcoin
}
```

4.5 PAYMENT VERIFICATION CIRCUIT

The Payment Verification Circuit ensures the MM is correctly filling Orders by validating the structure of their Bitcoin transaction. The circuit verifies three invariants:

1. The MM's transaction is included in a specified block root (Bitcoin SPV)
2. The transaction output is sending the user the exact sats output specified in the `Order`
3. The `Order` hash is included in the set of expected Orders, and the aggregate hash is inscribed in the transaction using `OP_RETURN`.

The final Order and transaction data are then returned to commit to the witness data.

```
public_input = {
  order_hash: [u8; 32], // hash of user's order
  bitcoin_block_hash: [u8; 32], // payment block hash
  bitcoin_tx_id: [u8; 32], // payment transaction id
}
```

5. SECURITY CONSIDERATIONS

Given the objective of creating a maximally secure exchange between Bitcoin and Ethereum, it is critical to discuss all possible attack vectors and how the protocol design prevents them.

1. Double claiming ERC20 for a single BTC payment - Prevented by the aggregate order hash inscribed in the `OP_RETURN` of the Bitcoin transaction, which binds each transaction to a specific set of Orders. Submitting a proof to claim user funds would fill the `Order`, thus preventing future use.

2. BTC payment is re-orged after MM claims ERC20 - Prevented by the user specifying the number of confirmation blocks required for a Bitcoin transaction to be considered valid. For the majority of transactions, this can be two blocks, as there hasn't been a longer re-org in 5+ years [Sta23, Bit17]. For high value transactions, this can be increased to meet confidence requirements of the user, with 4-6 typically considered "irreversible" in modern Bitcoin.

3. Submitting invalid Bitcoin blocks to the Light Client - Prevented by the state transition circuit ensuring the proposed blocks are valid and part of the longest chain. To successfully create a longer chain, an attacker would need more computational mining power than the Bitcoin network itself, or a stale light client far behind the actual Bitcoin network.

4. Submitting fraudulent Bitcoin blocks to a stale Light Client - In the case of a stale light client (no transactions for an extended period of time, resulting in an out of date tip block relative to the canonical Bitcoin chain), an attacker with sufficient computational power could mine fraudulent blocks that contain a payment to a user in an attempt to claim their escrowed asset. This attack becomes easier as the light client becomes more out of date. While a stale light client indicates no recent transactions, meaning no funds immediately available to steal, the next user that deposits could be vulnerable. To prevent this, funds are not released immediately after a proof is accepted. Instead, there is a challenge period where anyone can submit a heavier work chain, invalidating the fraudulent transaction. This relies on at least one honest online party who can submit a challenge to the light client with block data from the canonical Bitcoin chain. In order to do this, the challenge period must be long enough to generate a proof for the number of blocks between the light client tip and Bitcoin's tip, which we'll call Δb. To benchmark the minimum time to generate a proof, we chose the minimum hardware that supports CUDA proving with our SP1 circuits (8 vCPUs, 32 GiB RAM, and CUDA enabled GPU with 24 GiB of VRAM). This is an accessible minimum that can be achieved on cloud providers (such as AWS with a `g5.2xlarge` instance that has a NVIDIA A10G GPU), or with high end consumer cards such as RTX 3090 or newer. The results of the benchmark on 1000 runs on AWS, with the resulting time to prove T_p, are below:

Blocks	**Time**	**Standard Deviation**
1	58.28s	407ms
6	58.47s	380ms
24	58.67s	313ms
144	1.05m	266ms
288	1.15m	409ms
576	1.35m	420ms
1008	1.66m	473ms
2016	2.37m	720ms

$$T_p(x) = 0.0418x + 57.5842$$
$$R^2 = 0.9997$$

The final challenge period for Δb blocks can be defined below, with a 10% buffer applied for deviation in machine proof generation time, in addition to the finality time of the EVM in which the contract is deployed.

$$\tau_{\text{challenge}} = 1.1 \times T_p(\Delta b) + f_{\text{evm}}$$

5. Multiple MMs paying the same user and only one being able to claim the ERC20 - Prevented by the auction having only one winning MM, and the resulting `Order` can only pay out the auction winner.

6. Order lock-up expires after the MM has paid, but before they can claim the ERC20 - An early-expiring lock-up would result in the escrowed ERC20 being returned to the user after they had been paid. The protocol therefore sets the lock-up timer so that, with a one-in-a-billion failure probability ($\varepsilon = 10^{-9}$), the following three events complete before expiry:

i. **k Bitcoin confirmations**

We treat block confirmation times as a homogeneous Poisson process with rate $\lambda = 1/600s$, based on practical analysis of block times [Nak08, Bow+18, Vuo19]. Under this assumption, the waiting time for k blocks is Erlang-distributed. We pre-compute the $(1-\varepsilon)$ quantile of the Erlang(k, λ) distribution for every $1 \leq k \leq 255$. Empirically, this captures the entire distribution of inter-arrival times for up to 6 sequential blocks in the ASIC-mining era (post-block 337,340). To keep on-chain arithmetic cheap, we fit the computed quantiles with a square-root–linear function:

$$t(k) = \beta_0 + \beta_1\sqrt{k} + \beta_2 k$$

which results in:

$$t(k) = 142.59 + 53.92\sqrt{k} + 10.32k$$

With a RMS error is 0.01933% of the mean.

ii. **Proof generation**

Defined above as:

$$\tau_{\text{challenge}} = 1.1 \times T_p(\Delta b) + f_{\text{evm}}$$

iii. **Challenge window**

Since the lock-up is computed at the time an `Order` is created, we do not yet know the light client height where a potential challenge period would start. To remain safe in all cases, we use the worst-case value of $\Delta b = 2016$. Consequently, an `Order` should only be created if:

$$(h_{\text{BTC}} + k) - h_{\text{LC}} < 2016$$

where h_{BTC} is the observed Bitcoin height, h_{LC} is the on-chain light-client height, and k is the requested confirmation count. Otherwise, an auxiliary light-client update should be submitted first.

Combining the proof generation and challenge window yields the fixed term:

$$2 \times \tau_{\text{challenge}}(2016)$$

Along with the confirmations, the lock-up timer is:

$$T_{\text{lock\_up}} = 2 \times \tau_{\text{challenge}}(2016) + t(k)$$

which ensures that the Bitcoin payment is under k confirmation blocks with 99.9999999% confidence, the MM has time to post a proof, and there exists a full dispute window remains before the escrow can be withdrawn.

7. MM has a free option on locked user funds - Prevented by using a bridged Bitcoin as the underlying escrowed asset, making the option value negligible.

8. MM locking user funds and never sending BTC (Denial of Service) - Prevented by the user passing a whitelist of MMs who can bid for their order. This ensures they can opt into trading with only reputable MMs who always fill orders they commit to, while retaining permissionless of the system. An alternative design would be to have all auctions be open, with a minimum stake requirement to bid as a MM. This introduces a major pricing challenge: a stake that is too low will result in cheap DOS attacks, especially in the case of notable individuals with doxxed wallets who could be infinitely denied from using the protocol. On the other hand, a stake that is too high decreases capital efficiency for MMs significantly. Since the majority of volume will come from a handful of MMs, we opted for the simpler design rather than over optimize for long tail MM discovery.

6. LIMITATIONS AND FUTURE WORK

There are a number of limitations with the current protocol that could be improved in subsequent iterations:

1. Bitcoin taker support - The protocol outlined is designed around EVM takers and Bitcoin makers. While it theoretically could work in the opposite direction, the current design is limited by the “walk away” problem of a Bitcoin taker. Specifically, after agreeing to a price, an EVM maker would need to lock their ERC20 in `RiftExchange` until they received payment. In the current protocol, there is no way to prevent a Bitcoin taker from walking away from this agreed-upon transaction, since the maker cannot use a whitelist (as there are many takers who would have no reputation). An alternative design could be to require the taker to deposit some collateral before the maker locks their capital, but this is impractical as many Bitcoin takers would not have ERC20s (in addition to being a bad user experience). One proposed solution around this is using a “proxy wallet”, which the taker sends BTC to, and forwards the BTC to the maker once they have deposited the ERC20. This forwarding logic could be run inside the taker’s wallet, or a TEE [Cos+16] to guarantee their funds could not be stolen. In a future where `OP_CAT` [Hei+24] is merged into Bitcoin core, there are several alternate designs that could verify another chain’s state on Bitcoin to spend the taker’s UTXO and achieve the same result.

2. Arbitrary ERC20 auctions - In order to simplify market making, the current design uses a bridged Bitcoin as the underlying asset for auctions. This is limiting for users who want to do large trades using other major assets (ETH, USDC, USDT), as they have to use on-chain routes to get to the bridged Bitcoin, which can result in significant slippage for large trade sizes. Additionally, using a bridged Bitcoin for the lockup period exposes the user to asset risk for at least a two block period, unlike a chain’s native token, such as ETH. An alternative is to allow for any ERC20 to be auctioned to MMs. This can be trivially supported at the contract level, but should be paired with a delta-neutral shorting strategy of the ERC20 to account for any price divergence between order filling and redemption. Additionally, a stake significantly greater than the option value of the ERC20/BTC pair would be required to ensure it was always unprofitable to delay payment (e.g. not immediately exercising the option). In the case of pairs that have non-zero option values due to price divergence (e.g. ETH/BTC, USDC/BTC), one possible solution would be to require MMs to stake x% of the swap size, which is slashed if they don’t fill the order.

3. Single MMR proof for Orders in the same block - MMs who fill multiple orders in a single block currently re-verify the same block N times in the `BitcoinLightClient`, increasing the gas cost of verification. This disproportionately affects low-value swaps, where a larger percentage of the swap is spent on gas. Gas can be saved here by optimizing the struct that MMs submit to identify what Orders they are filling with a specific UTXO.

4. Inscribe aggregate `Order` hash using Taproot witness - Paying requires the MM to inscribe the `Order` hash in a Bitcoin transaction, and taproot witness storage is cheaper than using `OP_RETURN`. Verifying this data is a more involved circuit implementation, but a clear optimization.

5. Combine approval and multicall into a single transaction using EIP-7702 - The permit call at the beginning of the swap flow can be combined with the `Bundler3` transaction using EIP-7702 [But+24], creating a one-click experience for future dapp and wallet users.

7. CONCLUSION

In summary, we’ve proposed a protocol that allows for peer-to-peer, trustless swaps between Bitcoin and Ethereum without relying on security-compromising intermediaries. We combined a novel Bitcoin ZK light client implementation with on-chain auction and escrow smart contracts to create a cross-chain swapping protocol that ensures the same level of security as the Bitcoin and Ethereum networks, as it does not rely on staked capital or third parties to submit proofs. Rift can be deployed to any Ethereum Virtual Machine (EVM) chain, allowing liquidity to flow seamlessly between the Bitcoin and Ethereum ecosystems for the first time.

REFERENCES

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2008. https://bitcoin.org/bitcoin.pdf.

[But14] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.* 2014. https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[Xie+22] Tiancheng Xie, Jiexi Zhang, Zihan Cheng, Fan Zhang, Yanzhi Zhang, Yinqian Jia, Dan Boneh, and Dawn Song. "zkBridge: Trustless Cross-chain Bridges Made Practical." 2022. https://doi.org/10.1145/3548606.3560652.

[Vog+15] Fabian Vogelsteller and Vitalik Buterin. *ERC-20 Token Standard.* 2015. https://eips.ethereum.org/EIPS/eip-20.

[IC325] IC3. *North Korea Responsible for $1.5 Billion Bybit Hack.* 2025. https://www.ic3.gov/psa/2025/psa250226.

[Inv23] Investopedia. *The Collapse of FTX: What Went Wrong With the Crypto Exchange?* 2023. https://www.investopedia.com/what-went-wrong-with-ftx-6828447.

[Cel22] Celsius Network LLC et al. *Bankruptcy Proceedings.* 2022. https://cases.stretto.com/celsius/.

[Tho20] THORChain. *A Decentralized Liquidity Network.* 2020. https://github.com/thorchain/Resources/blob/master/Whitepapers/THORChain-Whitepaper-May2020.pdf.

[Har+24] Simon Harman. *Chainflip Whitepaper.* 2024. https://assets.chainflip.io/whitepaper.pdf.

[Nea24] NEAR. *What is Chain Abstraction?* 2024. https://docs.near.org/build/chain-abstraction/what-is.

[Eth25] Etherscan. *Validators | Mainnet Beacon Chain (Phase 0).* 2025. https://beaconscan.com/validators (accessed May 11, 2025).

[MPS25] MiningPoolStats. *Bitcoin (BTC) SHA-256 Mining Pools.* 2025. https://miningpoolstats.stream/bitcoin (accessed May 11, 2025).

[CW25] CoinWarz. *Bitcoin Hashrate Chart—BTC Hashrate 621.48 EH/s.* 2025. https://www.coinwarz.com/mining/bitcoin/hashrate-chart (accessed May 11, 2025).

[Tho25] THORChain. *THORChain Network Explorer.* 2025. https://thorchain.net/nodes (accessed May 11, 2025).

[Gar24] Garden Finance. *Order Lifecycle Documentation.* 2024. https://docs.garden.finance/developers/core/order-lifecycle.

[Wbt19] wBTC. *Wrapped Tokens Whitepaper.* 2019. https://wbtc.network/assets/wrapped-tokens-whitepaper.pdf.

[CB24] Coinbase. *cbBTC.* 2024. https://www.coinbase.com/cbbtc.

[Thr24] Threshold Network. *tBTC Bitcoin Bridge Documentation.* 2024. https://docs.threshold.network/applications/tbtc-v2.

[Coi22] Cointelegraph. *Axie Infinity's Ronin Bridge Hacked for Over $600M.* 2022. https://cointelegraph.com/news/axie-infinity-s-ronin-bridge-hacked-for-over-600m.

[Ren23] Ren Project. *RenBTC Collapse - RenBridge.* 2023. https://bridge.renproject.io/.

[Cho16] Joseph Chow. *Linking the Chains with BTC Relay.* 2016. https://medium.com/consensys-media/linking-the-chains-with-btc-relay-5ffd2c8248.

[Wes+20] Martin Westerkamp and Jacob Eberhardt. *zkRelay: Facilitating Sidechains using zkSNARK-based Chain-Relays.* 2020. https://ieeexplore.ieee.org/document/9229702.

[Pre18] James Prestwich. *Cross-chain Auctions via Bitcoin Double Spends.* 2018. https://medium.com/summa-technology/summa-auction-bitcoin-technical-7344096498f2.

[Bit25] Bitcoin.org. *Null Data (OP_RETURN) Transactions.* 2025. https://developer.bitcoin.org/devguide/transactions.html#null-data (accessed May 11, 2025).

[Tod16] Peter Todd. *Merkle Mountain Ranges.* 2016. https://gnusha.org/pi/bitcoindev/20160517132311.GA21656@fedora-21-dvm/.

[Vel25] Velora. *Paraswap Aggregation Protocol.* 2025. https://docs.velora.xyz/intro-to-velora/velora-overview/aggregation-protocol (accessed May 11, 2025).

[Mor25] Morpho. *Bundler3.* 2025. https://github.com/morpho-org/bundler3.

[Uni22] Uniswap. *Permit2 and Universal Router.* 2022. https://blog.uniswap.org/permit2-and-universal-router.

[Suc24] Succinct Labs. *SP1 Documentation.* 2024. https://docs.succinct.xyz/docs/sp1/introduction.

[Rif25] Rift Research. *Hypernode.* 2025. https://github.com/riftresearch/protocol/tree/main/bin/hypernode.

[Sta23] Bitcoin Stack Exchange. *When Was The Most Recent Multiblock Reorg?* 2023. https://bitcoin.stackexchange.com/questions/120130/when-was-the-most-recent-multiblock-reorg.

[Bit17] BitMEX Research. *A Complete History of Bitcoin's Consensus Forks.* 2017. https://blog.bitmex.com/bitcoins-consensus-forks/.

[Bow+18] Rhys Bowden et al. *Blocktimes.* 2018. https://github.com/rhysbowden/blocktimes/tree/master.

[Vuo19] Aapeli Vuorinen. *The Blockchain Propagation Process: a Machine Learning and Matrix Analytic Approach.* 2019. https://bitcoin.aapelivuorinen.com/thesis.pdf.

[Cos+16] Victor Costan and Srinivas Devadas. *Intel SGX Explained.* 2016. https://eprint.iacr.org/2016/086.pdf.

[Hei+24] Ethan Heilman and Armin Sabouri. *BIP 420.* 2024. https://github.com/bip420/bip420.

[But+24] Vitalik Buterin, Sam Wilson, Ansgar Dietrichs, and lightclient. *EIP 7702.* 2024. https://eips.ethereum.org/EIPS/eip-7702.