Renegade Whitepaper

Protocol Specification, v0.4

Christopher Bender
chris@renegade.fi

Joseph Kraut
joey@renegade.fi

Abstract

Renegade[1] is an on-chain dark pool. In contrast to other non-custodial decentralized exchanges, Renegade maintains complete anonymity during the entire lifecycle of a trade.

Peer-to-peer order matching is inferred via a secure multi-party computation protocol, and atomic settlement of matched orders is performed via zero-knowledge proofs of valid matches. Because of this hybrid MPC-ZKP match-settle architecture, all third parties (including the block proposer) learn nothing about any user's token balances, pending orders, or trade history, ultimately leading to minimal miner extractable value and higher-quality trade execution at midpoint prices.

1 Introduction

Currently, non-custodial trading suffers from four big problems:

- **Miner Extractable Value**. Block producers (in a L1 context) and/or sequencers (in a L2 context) can see full transaction information, allowing for arbitrary reordering, frontrunning, backrunning, and trade censorship.
- **Pre-trade transparency**. Non-marketable trades that rest on a limit order book are visible to all third-parties, leading to quote fading.
- **Post-trade transparency**. All third-parties can query the entire trade history, leading to tracking and tracing of trading activities.
- **Address discrimination**. Traders can see the origination address (pseudonymous identity) of all outstanding orders, leading to worse fills against toxic counterparties.

All of these design flaws lead to worse trade execution, particularly for whale traders with large market impact.

Background on Dark Pools.
A *dark pool* is a well-understood feature of traditional finance market structure. Legally classified as "alternative trading systems", dark pools are off-exchange trading venues with better privacy protections for traders.

Dark pools have the same functionality as the more familiar "lit" exchanges like the NYSE or NASDAQ, with one important difference: The order book is not publicly visible, meaning that traders cannot see the outstanding quotes of others. Participants in a dark pool are only informed about matches on their own trades. This allows for traders to privately search for a counterparty, all without broadcasting their trading intentions to the wider market.

Dark pools are typically used for better price execution of trades on large blocks of equities. If a large trade were to be executed at once on a lit exchange, insufficient liquidity would lead to significant price impact and cross-exchange arbitrage. TWAP-style orders are often employed to help massage the order into the lit market over time, but statistical arbitrageurs can often detect such patterns, once again leading to quote fading and worse execution.

In decentralized systems, the block trade problem is even worse: Not only do current DEXes leak the current state of the order book, but blockchains inherently have fully-auditable state history. In addition to seeing the order book, all participants can analyze past activity of any trader.

At its core, these problems arise because of *too much information* that is leaked to arbitrary third-parties who are not a part of the trade.

Our Solution.
In this paper, we introduce Renegade, a non-custodial dark pool. We solve all four problems outlined previously by hiding all information about the state of the exchange with zero-knowledge proofs.

Renegade is functionally equivalent to a CLOB-style DEX, but with an encrypted and distributed order book. Matches between users' orders are inferred via a cryptographically secure multi-party computation. Once a match has been found, settlements of swapped tokens are done via zero-knowledge proofs to hide all trade information while maintaining consistency of the system.

In order to implement this MPC-ZKP architecture, we use the **collaborative SNARK** framework from Ozdemir et

[1]Renegade is hiring! Check out our jobs page and get in contact on Twitter.

al.[2] Essentially wrapping zero-knowledge proof generation inside of a MPC, collaborative SNARKs allow for traders to transact while leaking zero information to third-parties.

In the remainder of this paper, we give a formal protocol specification for Renegade.

We start by giving a general overview of the protocol in Section 2, defining the idea of a *wallet* that maintains private state. We describe how wallets are created and destroyed, showing how the *commitment tree* can allow full user state privacy. From this, we give a precise specification of all state that is maintained by the system, both on the client side and on the smart contract side.

Next, in Section 3, we illustrate the full lifecycle of a trade from wallet creation to trade settlement, and explain the core collaborative SNARK protocol that allows for completely anonymous trading.

Then, in Section 4, we describe our peer-to-peer protocol for counterparty discovery. We illustrate the network topology of the system, introducing the concept of a *relayer*. In addition, we define the key hierarchy that allows for various levels of access controls into a trader's wallet.

Next, in Section 5, we describe the concept of *indications of interest*, showing how Renegade allows for individual traders to trade off privacy for liquidity.

Finally, in the Appendix A, we give formal specifications for the seven different NP statements that are used to ensure state consistency within Renegade.

2 Protocol Overview

In this section, we describe the stateful elements of the protocol. Later, in Section 3, we define the rules by which this state may be updated.

2.1 Wallets

Balance and order information cannot be posted on-chain (or else privacy would be compromised), so traders instead maintain a private **wallet** containing all local state. A wallet has two primary functions:

- Keep track of what tokens a trader owns: Renegade maintains global pools of all token deposits, so that balances are completely obfuscated.
- Keep track of the trader's set of outstanding orders.

Formally, let $M_O, M_B, M_F \in \mathbb{N}$ be public constants defining the maximum number of orders, balances, and fee approvals, respectively, that a user may have at once. Now, for some prime field $\mathbb{F}$, a wallet W is defined as a tuple

$$W := (B, O, F, \mathrm{pk}^{\mathrm{root}}, \mathrm{pk}^{\mathrm{match}}, \mathrm{pk}^{\mathrm{settle}}, r) \in \mathbb{F}^{2M_B+7M_O+2M_F+4}$$

with the following definitions:

- $B = (m_i, v_i)_{i\in[M_B]}$ is a list of size M_B of elements of $\mathbb{F}^2$:
  - ▶ $m_i \in \mathbb{F}$ is the mint (i.e. contract) address of a token that is held in this wallet.
  - ▶ $b_i \in \mathbb{F}$ is the amount of this token that is held in this wallet.
- $O = (k_i, m_{1_i}, m_{2_i}, s_i, p_i, a_i, \tau_i)_{i\in[M_O]}$ is a list of size M_O of elements of $\mathbb{F}^7$:
  - ▶ $k_i \in \mathbb{F}$ is a flag that denotes the type of the order (0 is midpoint-pegged, 1 is limit, etc.).
  - ▶ $m_{1_i} \in \mathbb{F}$ is the mint address of the quote token.
  - ▶ $m_{2_i} \in \mathbb{F}$ is the mint address of the base token.
  - ▶ $s_i \in \mathbb{F}$ is the side of the order (0 is buy, 1 is sell).
  - ▶ $p_i \in \mathbb{F}$ is the limit price (in units of quote per base), encoded as a fixed-point integer. Ignored if the order is not a limit type.
  - ▶ $a_i \in \mathbb{F}$ is the amount of base currency that the user wants to buy or sell.
  - ▶ $\tau_i \in \mathbb{F}$ is the timestamp of when this order was last updated, used for limit order price improvement.
- $F = \left(\mathrm{pk}^{\mathrm{settle}}_{\mathrm{relayer}_i}, \phi_i\right)_{i\in[M_F]}$ is a list of size M_F of elements of $\mathbb{F}^2$:
  - ▶ $\mathrm{pk}^{\mathrm{settle}}_{\mathrm{relayer}_i} \in \mathbb{F}$ is the public settle key of some relayer that is allowed to take a fee for matching this wallet.
  - ▶ $\phi_i \in \mathbb{F}$ is the fixed-point fee that is allowed to be taken by this relayer.
- $\mathrm{pk}^{\mathrm{root}} \in \mathbb{F}$ is the public key that corresponds to a secret key $\mathrm{sk}^{\mathrm{root}} \in \mathbb{F}$ that must be known in order to deposit/withdraw from the balances B, or to update the order book O. This is typically a secret key controlled by the end user / trader.
- $\mathrm{pk}^{\mathrm{match}} \in \mathbb{F}$ is the public key that corresponds to a secret key $\mathrm{sk}^{\mathrm{match}} \in \mathbb{F}$ that must be known in order to match outstanding orders in the order book O. This is typically a secret key controlled by a relayer.
- $\mathrm{pk}^{\mathrm{settle}} \in \mathbb{F}$ is the public key that corresponds to a secret key $\mathrm{sk}^{\mathrm{settle}} \in \mathbb{F}$ that must be known in order to settle the outputs of matches or balance transfers. Similarly to $\mathrm{sk}^{\mathrm{match}}$, this secret key is typically controlled by a relayer.
- $r \overset{\$}{\sim} \mathbb{F}$ is a random secret that is used to cryptographically hide the commitments and nullifiers.

As mentioned previously, the local lists B and O allow traders to keep all relevant balance and order information private from third-parties. In Section 2.2, we explain how the randomness r prevents leakage of any information about a user's wallet.

Additionally, in Section 4.1, we show how the three public keys $\mathrm{pk}^{\mathrm{root}}$, $\mathrm{pk}^{\mathrm{match}}$, and $\mathrm{pk}^{\mathrm{settle}}$ allow for various levels of

[2]Ozdemir and Boneh, *Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets*, https://eprint.iacr.org/2021/1530

access control over the wallet W, and in Section 4.2, we explain the role of the fee list F.

Note that the wallet W may be privately kept off-chain, or the wallet may be encrypted with a view keypair and written on-chain[3], so that a user only needs to keep track of their native Ethereum private key in order to access their funds.

2.2 The Commitment Tree

In order to keep track of which off-chain wallets are valid, the smart contract maintains a Merkle tree of **commitments**.

Let $H : \mathbb{F} \to \mathbb{F}$ be a SNARK-friendly hash function, and let $\mathcal{H} : \mathbb{F}^n \to \mathbb{F}$ be the Merkle hash function generated by iteratively applying H.

The commitment $C(W)$ to a wallet W is defined as

$$C(W) := H\left(\begin{array}{c} \mathcal{H}(B) \,||\, \mathcal{H}(O) \,||\, \mathcal{H}(F) \,|| \\ \mathrm{pk}^{\mathrm{root}} \,||\, \mathrm{pk}^{\mathrm{match}} \,||\, \mathrm{pk}^{\mathrm{settle}} \,||\, r \end{array} \right),$$

where $||$ denotes concatenation.

To perform operations on their wallet (depositing and withdrawing, submitting and cancelling orders, etc.), the user must **reveal** some information about their old wallet and **commit** to a new wallet. To ensure that the update is valid, the user must also supply a zero-knowlege proof that the update is benign (e.g. does not add free tokens to B).

Note that the randomness $r \in \mathbb{F}$ must be included in the definition of a wallet W and in the computation of the wallet commitment $C(W)$ in order to **hide** the contents of the wallet: If no such randomness were used, then adversaries could generate a rainbow table of common wallets (e.g., the wallet with zero balances and zero orders could be easily identified).

Zero-knowledge proofs are stateless (i.e., if a ZK proof is valid once, it will always be valid), so the contract needs to maintain some state in order to ensure that the user cannot double-reveal a wallet that was only committed to once. To do this, when revealing an old wallet, the user computes two **nullifiers** of a wallet in addition to computing the commitment to the new wallet.

The **nullifier set** is the set of all nullifiers that have been "seen", meaning that they have been used to reveal a wallet in the past. The contract will reject all reveal-commit transactions if any of the nullifiers have been seen before.

In order to reveal their wallet W, the user first constructs a new wallet W' with the appropriate changes (a new set of orders, a change in balances to reflect an order settlement, etc.). The user then computes the two nullifiers of their old wallet W, a **wallet-spend nullifier** and a **wallet-match nullifier**.

[3]For strongest guarantees against data loss, the user may choose to store the wallet as on-chain L1 calldata, paying high gas cost. However, if the user is particularly fee-sensitive, they may want to instead store the wallet in some off-chain data availability layer, avoiding the more expensive calldata fees.

The wallet-spend nullifier is

$$N^{\text{wallet-spend}}(W) := H(r)$$

and the wallet-match nullifier is

$$N^{\text{wallet-match}}(W) := H(r+1).$$

We will see in Section 3.4 how this dual-nullification allows for us to perform pairwise matches between two different wallets.

Now, the user submits a zero-knowledge proof that 1) there exists some valid Merkle path to $C(W)$, implying that a previous transaction committed to the wallet W, 2) the nullifiers are properly computed for the wallet W, 3) the transition from W to W' is valid (e.g., the user has not increased balances without depositing additional tokens), and 4) the user knows $\mathrm{sk}^{\mathrm{root}}$. The contract checks that the ZK proof is valid and that the two nullifiers have not already been seen. If this check passes, the contract marks the nullifiers as seen and inserts the new commitment $C(W')$ into the commitment Merkle tree.

This basic reveal-commit scheme is used for all possible operations on a user's wallet.

In addition to the wallet commitments, the global Merkle tree also accepts insertions of **notes**. A note is essentially an unspent transaction output (i.e. a claim on some funds that needs to be settled). Correspondingly, when a note is settled, a **note-settle** nullifier is revealed. We describe these further in Section 3.6.

2.3 Entire State

We have seen how *wallets* kept secret by individual traders, when combined with the idea of the *commitment tree*, allows for privately-held state with global consensus about validity of that distributed state.

In Table 1, we summarize the previous two sections into a precise description of all state (with types) held by both individual clients and the global contract.

3 Trade Lifecycle

In this section, we will go through an entire lifecycle of a trade, including creating a wallet, depositing into the system, peer discovery and handshakes in our p2p protocol, the multiparty computation itself, and settlement of matched trades.

3.1 Creating a New Wallet

When a user joins Renegade for the first time, they have no wallet that has been committed in the global Merkle tree. So, the smart contract has a special functionality that allows for inclusion of a new wallet W without revealing any nullifier,

	Notation	Type
Client State		
Balances List	$B = (m_i, v_i)_{i \in [M_B]}$	$\mathbb{F}^{2M_B}$
Orders List	$O = (k_i, m_{1_i}, m_{2_i}, s_i, p_i, a_i, \tau_i)_{i \in [M_O]}$	$\mathbb{F}^{7M_O}$
Fees List	$F = (\mathrm{pk}_i^{\mathrm{relayer}}, f_i)_{i \in [M_F]}$	$\mathbb{F}^{2M_F}$
Root Public Key	$\mathrm{pk}^{\mathrm{root}}$	$\mathbb{F}$
Match Public Key	$\mathrm{pk}^{\mathrm{match}}$	$\mathbb{F}$
Settle Public Key	$\mathrm{pk}^{\mathrm{settle}}$	$\mathbb{F}$
Randomness	r	$\mathbb{F}$
Contract State		
Merkle Path	`current_merkle_path`	$\mathbb{F}^L \times \mathbb{B}^L$
Nullifier Set	`is_nullifier_used`	$\mathbb{B}^{\mathbb{F}}$
Wallet Store	`wallet_store`	$\left(\mathbb{F}^{2M_B+7M_O+2M_F+4}\right)^{\mathbb{F}}$

Table 1. Full Client and Contract State

so long as the user proves that the wallet W is indeed a new wallet.

Specifically, the user generates a new wallet

$$W = (B, O, F, \mathrm{pk}^{\mathrm{root}}, \mathrm{pk}^{\mathrm{match}}, \mathrm{pk}^{\mathrm{settle}}, r)$$

by setting B, O, F to be the lists of all zeros, setting $\mathrm{pk}^{\mathrm{root}}$, $\mathrm{pk}^{\mathrm{match}}$, and $\mathrm{pk}^{\mathrm{settle}}$ to be appropriate access control keys as discussed in Section 4.1, and choosing a random r.

Then, this user submits $C(W)$ to the contract, along with a proof that $C(W)$ was indeed computed by committing to some wallet that had zero balance. Once the contract verifies this proof, it inserts $C(W)$ into the global Merkle tree, now creating a usable wallet for the new trader.

We instantiate this argument of knowledge as a formal NP statement and corresponding R1CS constraint system in the statement VALID WALLET CREATE, defined in Section A.1.

3.2 Updating a Wallet

Now that a user has committed to a new wallet W, they may *update* this wallet. When updating a wallet, a trader may do any subset of the following:

- Depositing or withdrawing external ERC-20 balances from outside the dark pool.
- Send some tokens to a different user inside of the dark pool.
- Add new orders or cancel old orders in O.
- Add new fee approvals or cancel old approvals in F.

Formally, the user generates a new wallet

$$W' = (B', O', F', \mathrm{pk}^{\mathrm{root}'}, \mathrm{pk}^{\mathrm{match}'}, \mathrm{pk}^{\mathrm{settle}'}, r')$$

with arbitrary O', F', r. The user also creates two tuple of *transfer tokens*, the **internal transfer tuple** and the **external transfer tuple**.

The internal transfer is a tuple

$$T_I = (\tilde{m}_I, \tilde{v}_I) \in \mathbb{F}^2$$

and the external transfer is a tuple

$$T_E = (\tilde{m}_E, \tilde{v}_E, \tilde{d}_E) \in \mathbb{F}^3.$$

These two tuples determine what token (if any) to be either transferred to another user inside of the dark pool or deposited/withdrawn from the protocol entirely. $\tilde{m}_I$ and $\tilde{m}_E$ denote the token type (i.e., mint), and $\tilde{v}_I$ and $\tilde{v}_E$ determine the amount of token to be transferred. $\tilde{d}_E$ denotes the direction (0 is deposit, 1 is withdraw) of the external transfer.

Note that either tuple may consist entirely of zeros, indicating that the user does not desire to transfer any tokens.

Then, the user submits $C(W')$, T_E, $N^{\text{wallet-spend}}(W)$, and $N^{\text{wallet-match}}(W)$ to the contract, alongside a proof that W' was indeed formed by correctly applying the transfer tuples T_I and T_E to the old wallet W and all commitments and nullifiers are correctly computed. As before, we provide a formal NP statement of VALID WALLET UPDATE in Section A.3.

3.3 Handshakes

Now that a user has a wallet W with non-zero lists of balances B and orders O, they may begin searching for potential counterparties to trade with.

In order to find peers, Renegade implements an off-chain **peer-to-peer messaging protocol**. Implemented over QUIC transport for low handshake latency, the p2p network allows for both 1) gossip for peer discovery, and 2) Kademlia DHT-based peer lookup and information exchange.

To find peers, the trader connects to the network and selects an order $o = (k, m_1, m_2, s, p, a, \tau) \in O$ that they would like to match. The trader then finds the balance $b = (m, v) \in B$ that **covers** this order (e.g., if o is a buy order, then $m = m_2$).

In addition, the trader selects a **relayer fee** $f = \left(\text{pk}^{\text{settle}}_{\text{relayer}}, \phi\right)$ to be taken by the party that is performing this computation (relayers are explained in Section 4).

Now, the trader generates three values $H_o = H(o \mid\mid r)$, $H_b = H(b \mid\mid r)$, and $H_f = H(f \mid\mid r)$. These are hiding and binding commitments to the chosen order, associated covering balance, and fee tuple; they used for cross-input consistency between the MPC and the zero-knowledge proof.

Finally, the trader generates a zero-knowledge proof π of the statement VALID COMMITMENTS, as defined in Section A.2. This statement essentially proves that the trader does indeed know some unspent wallet W containing an order o, balance b, and fee f with the given commitment hashes H_o, H_b, and H_f.

Note that the generation of π may be done completely asynchronously and be reused over multiple attempted MPCs: The proof is agnostic to the counterparty.

Now that the grader has a generated a proof π of VALID COMMITMENTS, they may begin handshaking with potential counterparties.[4] The trader sends the handshake tuple

$$H := (\pi, H_o, H_b, H_f, N^{\text{wallet-match}}(W), \text{pk}^{\text{settle}})$$

to a potential counterparty, and the counterparty checks that the proof is correct, that the nullifier has not yet been "seen" on-chain (meaning that the wallet would be already spent), and that the nullifier pair

$$(N^{\text{wallet-match}}(W_1), N^{\text{wallet-match}}(W_2))$$

has not already been cached as a non-match.[5]

If the proof is accepted by the counterparty and the wallet-match nullifier has not already been cached, then the counterparty responds with a similar handshake tuple H_2. The trader checks that the counterparty's handshake proof is valid, and if so, the traders proceed with the MPC.

3.4 MPC and Match Proofs

Let Party 1 hold order $o_1 \in \mathbb{F}^7$, balance $b_1 \in \mathbb{F}^2$, fee $f_1 \in \mathbb{F}^2$, and randomness $r_1 \in \mathbb{F}$. Let o_2, b_2, f_2, r_2 be defined similarly for Party 2. The parties Shamir secret-share all eight of these values with each other. In addition, the parties secret-share the publicly-known values $H_{o_1}, H_{b_1}, H_{f_1}, H_{o_2}, H_{b_2}, H_{f_2}$.

Now, given all the shares[6]

$$[o_1], [o_2], [b_1], [b_2], [f_1], [f_2], [r_1], [r_2],$$
$$[H_{o_1}], [H_{b_1}], [H_{f_1}], [H_{o_2}], [H_{b_2}], [H_{f_2}],$$

the parties run a SPDZ-style maliciously-secure MPC-with-abort to compute a secret-share of the **match tuple**

$$M := (\hat{m}_1, \hat{m}_2, \hat{v}_1, \hat{v}_2, \hat{d}, f_1, f_2, H(r_1), H(r_2)) \in \mathbb{F}^{11}$$

and a secret-share of a hiding commitment to the tuple $H_M = H(M)$.

This tuple gives the matched values between the two orders o_1 and o_2 when constrained by the balances b_1 and b_2: That is, assuming the orders are of the same quote/base pair, $\hat{v}_i$ is the amount of $\hat{m}_i$ that is swapped between the two parties for $i = 1, 2$. $\hat{d} \in \mathbb{B}$ is the direction of the transfer, where $\hat{d} = 0$ means that Party 1 can increase their balances by $\hat{v}_1$ units of m_1 and decrease their balances by $\hat{v}_2$ units of m_2, and vice-versa for Party 2.

We include the additional randomness $H(r_1)$ and $H(r_2)$ as an output in the matches tuple in order to prevent against similar rainbow table-style attacks against commitments to match tuples.

Note that the reason we needed to secret-share the public commitments $H_{o_1}, H_{b_1}, H_{f_1}, H_{o_2}, H_{b_2}, H_{f_2}$ is in order to **zero out** the output matches M in case either parties lies about their inputs to the maliciously-secure MPC. Since the hashes used in the commitment function is preimage-resistant, it is infeasible for either of the parties to manipulate their inputs without also changing their commitments.

Importantly, note that the parties *do not open* M immediately. If the parties were to open the match now, both parties would learn information about each others' orders while being able to hangup the connection.

Now that the parties have secret-shares of every single wire in the MPC functionality including the output M, they perform a **collaborative SNARK** proving step that produces a secret-share of a proof π_M of the statement VALID MATCH MPC. Defined formally in Section A.5, this statement essentially proves that given the parties' collective inputs o_1, b_1, o_2, b_2, the commitment H_M is indeed the commitment to the unique match output of the input orders and balances.

Now, the traders may finally open π_M and M, revealing the output of the matches and a proof that the matches lists were correctly computed from valid pair of orders.[7]

We illustrate this entire handshake and matching process in Figure 1.

[4]In order to bootstrap connection into the p2p network, Renegade maintains an ENS record of **authoritative relayers** that allows for new entrants to find counterparties.

[5]Note that caching of nullifier pairs only works if both orders are of "limit" or "midpoint" type: Midpoint-limit matches are not cacheable.

[6]In addition to the secret-shared inputs described here, the parties must also agree on a vector of **midpoint prices** from an oracle for a fixed number of assets. They secret share these points $[m_i]$, $[p_i]$ for use in midpoint order calculation.

[7]TODO: Describe guaranteed output delivery / fairness here.

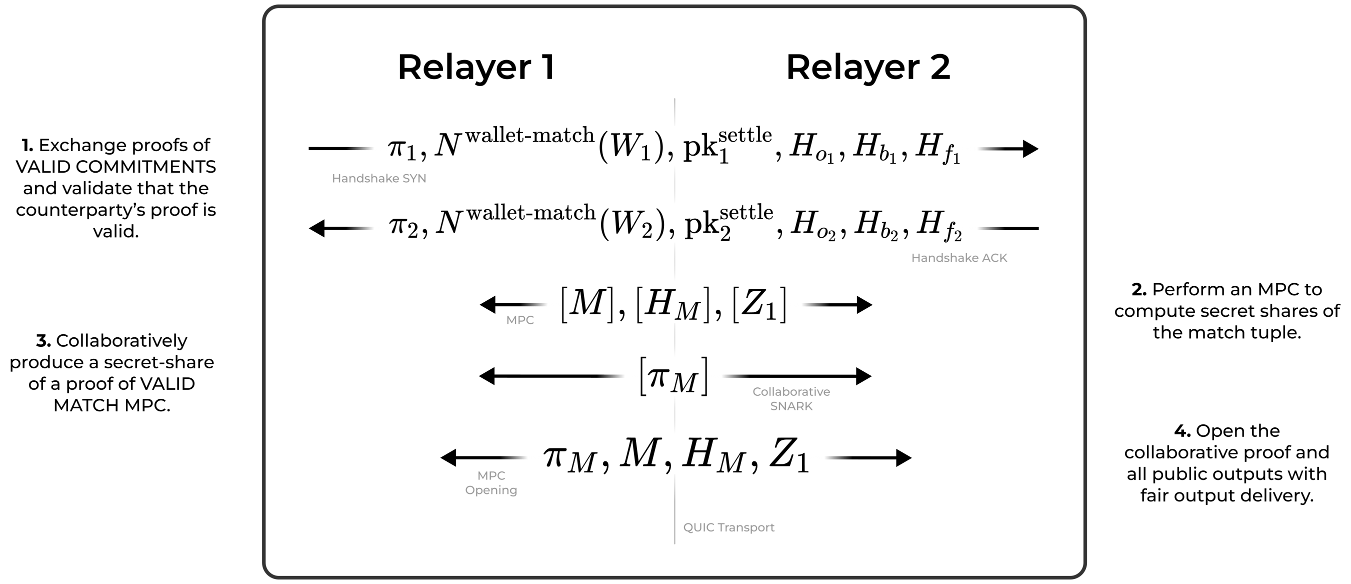
1. Exchange proofs of VALID COMMITMENTS and validate that the counterparty's proof is valid.
3. Collaboratively produce a secret-share of a proof of VALID MATCH MPC.
Relayer 1
Relayer 2
$\pi_1, N^{\text{wallet-match}}(W_1), \text{pk}_1^{\text{settle}}, H_{o_1}, H_{b_1}, H_{f_1}$
Handshake SYN
$\pi_2, N^{\text{wallet-match}}(W_2), \text{pk}_2^{\text{settle}}, H_{o_2}, H_{b_2}, H_{f_2}$
Handshake ACK
MPC
$[M], [H_M], [Z_1]$
$[\pi_M]$
Collaborative SNARK
π_M, M, H_M, Z_1
MPC Opening
QUIC Transport
2. Perform an MPC to compute secret shares of the match tuple.
4. Open the collaborative proof and all public outputs with fair output delivery.

Figure 1. Inter-Relayer Communication Flow

3.5 Encumbering

Once the proof π_M and matches tuple M have been opened, both parties now have enough information to complete the match and the communication link between the parties may be closed.

As a final step before this proof may be submitted to the contract, the trader must prepare a few slightly altered matches lists, termed **notes**. These notes are essentially per-trader records of what tokens were swapped and are directly generated from M. One note is generated for each of the traders' wallets:

$$N_1 := \left(\hat{m}_1, \hat{v}_1, \hat{d}, \hat{m}_2, \hat{v}_2, 1 - \hat{d}, \mu, H(r_1)\right) \in \mathbb{F}^8$$

and

$$N_2 = \left(\hat{m}_1, \hat{v}_1, 1 - \hat{d}, \hat{m}_2, \hat{v}_2, \hat{d}, \mu, H(r_2)\right) \in \mathbb{F}^8.$$

In addition to these two notes, we generate two extra notes for the relayer fees (relayers are discussed in Section 4), plus one final note for the global protocol fee.

We cannot send notes in-the-clear to the contract (or else third-parties would learn what tokens are being swapped), so we encrypt these two matches lists under the corresponding settle key of each party:

$$E_{\text{pk}_1^{\text{settle}}}(N_1)$$

and

$$E_{\text{pk}_2^{\text{settle}}}(N_2).$$

Now, the trader generates a proof π_E of the statement VALID MATCH ENCRYPTION as defined in Section A.7, which essentially shows that the notes N_1, N_2 we indeed formed from the original match tuple M, that M is indeed consistent with the publicly-known commitment H_M, and that the notes are properly encrypted.

Finally, after generating π_E, the trader is now ready to interact with the smart contract. The trader sends four different proofs π_1, π_2, π_M, π_E to the contract (i.e. two proofs of VALID COMMITMENTS, one proof of VALID MATCH MPC, and one proof of VALID MATCH ENCRYPTION), alongside all public variables:

$$H_{o_1}, H_{b_1}, H_{o_2}, H_{b_2}, N^{\text{wallet-match}}(W_1), N^{\text{wallet-match}}(W_2),$$
$$\text{pk}_1^{\text{match}}, \text{pk}_2^{\text{match}}, H_M, R_{\text{global}}, E_{\text{pk}_1^{\text{match}}}(N_1), E_{\text{pk}_2^{\text{match}}}(N_2)$$

The contract checks that all four zero-knowledge proofs are valid under the given public inputs. If all checks pass, the contract then marks the two nullifiers $N^{\text{wallet-match}}(W_1)$ and $N^{\text{wallet-match}}(W_2)$ as being used, an inserts both of the encrypted notes into the commitment tree.

Note that this nullification is different from how reveal-commit schemes work in VALID WALLET UPDATE as in Section 3.2. Here, we only mark the *wallet-match* nullifiers as "seen", not the wallet-spend nullifiers. In addition, we do not insert a commitment to a wallet into the global Merkle tree; rather, we are inserting an encrypted note.

Note that since we have revealed the wallet-match nullifiers, calling VALID WALLET UPDATE is now impossible, as neither party can prove that their wallet has an unseen wallet-match nullifier. Both wallets are now considered "encumbered", and the only operation that either party may perform is to settle their matched order.

3.6 Settlement

Now that the trader's wallet W has been matched and encumbered, they need to **settle** this match in order to update their balances and un-encumber the wallet.

To do this, the trader first obtains their note

$$N = (\hat{m}_1, \hat{v}_1, \hat{d}, \hat{m}_2, \hat{v}_2, 1 - \hat{d}, H(r)).$$

The trader can find their note by either remembering the match they just performed, or if the counterparty was the one to submit the match proofs, by scanning through the commitment history to find the most recent encrypted note and decrypt it under their secret key $\mathrm{sk}^{\mathrm{settle}}$.

Now, the trader constructs a new wallet

$$W' = (B', O', F', \mathrm{pk}^{\mathrm{root}'}, \mathrm{pk}^{\mathrm{match}'}, \mathrm{pk}^{\mathrm{settle}'}, r')$$

such that F', $\mathrm{pk}^{\mathrm{root}'}$, $\mathrm{pk}^{\mathrm{match}'}$, and $\mathrm{pk}^{\mathrm{settle}'}$ are unchanged from the original wallet W. The randomness r' is chosen randomly from $\mathbb{F}$ as always.

To construct B', the trader simply adds or subtracts values v_i from the balances list according to the note N. There will always be exactly one balance that is increased, and exactly one balance that is decreased.

Finally, to construct O', the trader finds the order

$$o = (k_i, m_{1_i}, m_{2_i}, s_i, p_i, a_i, \tau_i) \in O$$

that was matched by N and decreases the size a_i by the corresponding matched value $\hat{v}_1$ or $\hat{v}_2$ depending on the direction of the match.

Now, given this new wallet W' that was formed by directly settling the match note N against the old wallet W, the trader constructs a proof π_S of the statement VALID SETTLE as defined in Section A.8.

In addition to revealing the commitment to the new wallet $C(W')$ as normal, the trader also reveals 1) the wallet-spend nullifier of their old wallet $N^{\text{wallet-spend}}(W)$ and 2) the **note-redeem** nullifier defined as

$$N^{\text{note-redeem}}(N) = H(H(r)).$$

Match-use nullifier exist in order to prevent replay-style attacks by double-settling a matched order.

The trader then sends this proof π_S to the contract, and assuming it is correctly verified, the contract will mark both $N^{\text{wallet-spend}}(W)$ and $N^{\text{match-use}}(N)$ as being seen, and insert the new commitment $C(W')$ into the Merkle tree.

Now, the trader has settled their matched orders, and all three nullifiers (wallet-match, wallet-spend, match-use) have been seen, making further use of ether the old wallet W or the used note N impossible.

In summary, this basic update-match-settle lifecycle is used for every single order that a trader would like to perform, all while avoiding any information leakage to third-parties.

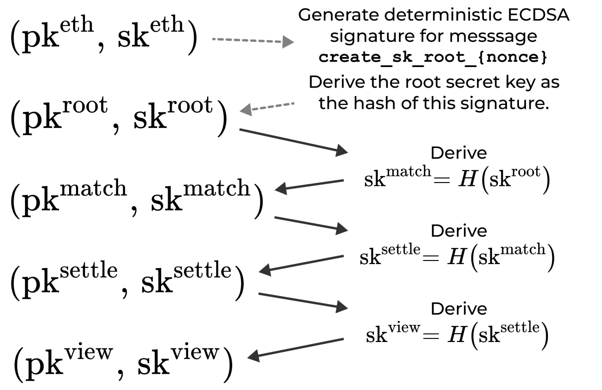
$(\mathrm{pk}^{\mathrm{eth}}, \mathrm{sk}^{\mathrm{eth}})$
Generate deterministic ECDSA signature for messsage create_sk_root_{nonce}
Derive the root secret key as the hash of this signature.
$(\mathrm{pk}^{\mathrm{root}}, \mathrm{sk}^{\mathrm{root}})$
Derive $\mathrm{sk}^{\mathrm{match}} = H(\mathrm{sk}^{\mathrm{root}})$
$(\mathrm{pk}^{\mathrm{match}}, \mathrm{sk}^{\mathrm{match}})$
Derive $\mathrm{sk}^{\mathrm{settle}} = H(\mathrm{sk}^{\mathrm{match}})$
$(\mathrm{pk}^{\mathrm{settle}}, \mathrm{sk}^{\mathrm{settle}})$
Derive $\mathrm{sk}^{\mathrm{view}} = H(\mathrm{sk}^{\mathrm{settle}})$
$(\mathrm{pk}^{\mathrm{view}}, \mathrm{sk}^{\mathrm{view}})$

Figure 2. Key Derivation Hierarchy

4 Relayer Delegation

In the previous Section 3 outlining the entire lifecycle of a trade, we assumed that the end-trader would be online at all times to handshake and perform MPC calculations with arbitrary counterparties.

However, this is unreasonably restrictive: Traders may want to "fire and forget" their orders, much like how current centralized exchanges operate. In addition, the trader may want to handshake with many counterparties at once (every single other node in the network may have a valid counter-order).

To allow for this, we introduce the concept of **relayers**. A relayer is essentially a stand-in for a trader that holds their wallet W and continually attempts MPC calculations followed by match-settle proofs in the event of a match.

In Section 4.1, we describe the key hierarchy that allows for trust minimization between the end-trader and their relayer, and in Section 4.2 we describe the high-level network topology of the system, including the idea of "relayer clusters".

4.1 Key Hierarchy

In designing the relayer system, the primary goal is *trust minimization* between the end-trader and the relayer. Specifically, a relayer should only ever be able to match outstanding orders and settle previous matches; the relayer should *not* be able to create or cancel orders, or deposit or withdraw funds.

To implement this level of access control, we introduce the **key hierarchy**, as outlined in Figure 2. Alongside the base Ethereum keypair, we have four different levels of Renegade-native keys, all with various levels of access controls.

The key hierarchy begins with a trader's Ethereum keypair $(\mathrm{pk}^{\mathrm{eth}}, \mathrm{sk}^{\mathrm{eth}})$ on the secp256k1 curve. Let

$$(r, s, v) \in \mathbb{B}^{256} \times \mathbb{B}^{256} \times \mathbb{B}^{8}$$

be the deterministic ECDSA signature of the message

```
create_sk_root_{nonce}
```

for some $\text{nonce} \in \mathbb{N}_{\geq 0}$. Construct

$$\text{sk}^{\text{root}} = H(r \mid\mid s \mid\mid v),$$

and recover the corresponding public key pk^{root}.

Using this process, we may always re-derive the root keypair so long as the trader does not lose their native Ethereum keypair. Note that we may skip this derivation entirely and simply generate a random sk^{root} if the trader does not want to maintain an Ethereum keypair, or if the trader wants to use a Renegade-native multisig solution.

This **root keypair** is the ultimate authority over a user's wallet: Any user who knows this secret key may perform arbitrary operations to the balances and orders inside the wallet, including withdrawing all funds.

Now that we have sk^{root}, we derive the other three keypairs simply as

$$\text{sk}^{\text{match}} = H(\text{sk}^{\text{root}})$$

and

$$\text{sk}^{\text{settle}} = H(\text{sk}^{\text{match}})$$

and

$$\text{sk}^{\text{view}} = H(\text{sk}^{\text{settle}})$$

and correspondingly recover the public keys pk^{match}, $\text{pk}^{\text{settle}}$, and pk^{view}. These three final keys are the **match keypair**, the **settle keypair**, and the **view keypair**.

Any party who knows the match keypair is allowed to match outstanding orders in the orders list O; importantly, this key is *not* able to arbitrarily modify O, deposit, or withdraw. The settle keypair has less authority, only being able to settle previous matches, or settle direct token transfers from another user inside the pool. Finally, the view keypairs has the least amount of authority, only being able to decrypt and view the wallet; the view keypair cannot modify the wallet in any way.

By introducing this five-level key hierarchy, the system allows for various levels of access control. In particular, the trader *only* sends sk^{match} to the relayer who will match orders on the trader's behalf. Even if the relayer is completely compromised, the worst outcome is that the relayer leaks the wallet W, compromising the privacy of the trader, but not stealing funds.

In addition, the arbitrary $\text{nonce} \in \mathbb{N}_{\geq 0}$ allows for key rotation, whereby a single Ethereum key can control many different wallets.

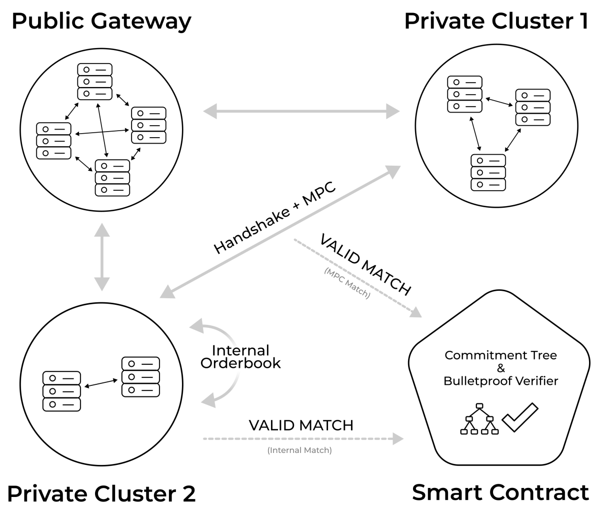

Figure 3. Network Topology, showing matches derived from both MPC and internal proofs.

4.2 Network Architecture

In Figure 3, we illustrate the high-level p2p network topology.

At the base layer, the network simply consists of various relayers that communicate with each other. The network is permissionless, meaning that at any time any new trader may enter the network as their own relayer and begin MPC handshakes.

However, since the network needs to support a large number of potential matches on many orders, the relayers are grouped into logical units called **relayer clusters**.

These clusters are fault-tolerant replicated groups of relayer nodes that all manage the same set of wallets. This replication allows for higher throughput of matches (since many relayers can try different matches at once), and allows for fault-tolerance under network interruption and partition.

In Figure 3, we illustrate three different clusters communicating with each other: Private Cluster 1 with three relayers, a smaller Private Cluster 2 with two relayers, and a special larger Public Gateway. The Public Gateway is a cluster of relayers like the rest (i.e., it has no special permissions), but is a Renegade-run set of relayers that allows for bootstrap connectivity into the network, and allows for traders who do not want to run their own infrastructure to participate in the network.

Note that high-volume traders who are maximially privacy-concerned should run their own relay clusters, as using the Gateway exposes trader's wallets to the centralized Gateway provider.

In addition to the standard handshake and MPC process, Figure 3 also illustrates the idea of an "internal match", to be described in Section 5. Also, Figure 3 illustrates the submission of a VALID MATCH MPC proof to the global Merkle commitment tree and zero-knowledge-proof verification contract.

4.3 Relayer Fees

In the definition of a wallet W, there is one final element that we have not yet described: The **fee list** F. Since running a relayer requires somewhat expensive hardware (e.g., zero-knowledge proving is quite memory-intensive), relayers naturally need compensation for performing matching and settlement.

To implement this functionality, the list

$$F = \left(\mathrm{pk}^{\mathrm{settle}}_{\mathrm{relayer}_i}, \phi_i\right)_{i \in [M_F]}$$

contains tuples of fee approvals. Each relayer that wants to match public wallets advertises a tuple

$$\left(\mathrm{pk}^{\mathrm{settle}}_{\mathrm{relayer}_i}, \phi_i\right).$$

The key $\mathrm{pk}^{\mathrm{settle}}_{\mathrm{relayer}_i}$ is the settle key of some relayer-controlled wallet, and the fee ϕ_i is the fixed-point-encoded fee that the relayer takes as a percentage of each matched transaction.

If this key-fee tuple is acceptable, the trader inserts this tuple into their fee list F. We include a formalization of this fee-taking process in Section A.8.

Importantly, note that fees may be avoided entirely for traders who run their own relay clusters: Indeed, we charge substantial fees on the Public Gateway to promote maximal decentralization of the network.

5 Indications of Interest

As mentioned in Section 2, the principal goal of the base-layer Renegade system is to ensure complete privacy of all relevant values (orders, balances, matches, etc.) from all third-parties to the trade.

However, in practice, having completely obfuscated "dark" orders may not be optimal for liquidity provision: Indeed, there is a core tradeoff between quality-of-execution and speed-of-execution (dark pools give best price, whereas lit pools let you transact immediately).

For traders who may want to tradeoff price for speed, Renegade allows for additional **indications of interest** flags.

An indication of interest is some predicate on a wallet W, proved in zero-knowledge. For example, a trader may reveal the fact "my wallet W is a buy order of WETH/USDC at the midpoint price", without disclosing the size of the trade. To ensure that the trader is not lying, this predicate must be proved in zero-knowledge as a part of the handshake process.

In Section A.4, we give formal NP statments for every IoI flag that Renegade supports, including the quote/base token pair, the side (buy or sell), if the order is a midpoint-pegged order, the limit price of the order, and the size of the order.

Note that the maximal indication of interest (i.e., turning on and proving all IoI flags) makes an order *equivalent to a lit order*. Indeed, we can embed an entire lit orderbook with dark-lit crossover matches inside of Renegade.

Note that if a counterparty turns on all IoI flags, there is actually no need to compute a MPC as normal: We may directly lift this lit order and match it with one of our own orders. In Section A.6, we formally define the NP statement VALID MATCH LIT that allows for this functionality.

Finally, one optimization to note is that if a relayer manages two different wallets W_1 and W_2 that contain an overlapping order, once again MPC is not necessary and the relayer may simply match these two orders directly. Here, the relayer computes the same VALID MATCH MPC as normal, but does not need to actually run a MPC with itself in order to generate the proof π_M.

6 Conclusion

Renegade aims to solve the four core problems outlined in Section 1: MEV, pre-trade transparency, post-trade transparency, and address discrimination. We claim the following two strong privacy properties as solutions to these problems:

- All third parties who are neither the end-user nor a managing relayer learn zero information about the activities inside of the pool, other than global token inflows and outflows. In particular, third parties cannot deduce the balances or orders of any wallet, and they cannot deduce any details of any match or settlement other than the fact that a match and settlement occurred between some pair of traders.
- Individual relayers learn nothing about the state of balances and orders of wallets that they do not manage. The Public Gateway is not a special relay cluster and has no in-protocol advantages over private clusters.

In all, we have seen how the idea of a *reveal-commit* scheme allows for private yet consistent user-managed state in the network. When local state is combined with the p2p network and pairwise MPC calculations, Renegade achieves maximal possible DEX privacy, allowing for a truly anonymous global exchange of value.

A Formal NP Statements

Here, we give precise specifications for each of the eight different NP statements that are used by the protocol. All of these statements are implemented as R1CS constraint systems and proved/verified via Bulletproofs.

Each statement consists of a list of "public variables" that are publicly known to the verifier. The "private variables" are the secret witnesses known only by the prover. Finally, we have a "such that" list of all properties that are encoded into the constraint systems.

Note: For notational clarity, we have omitted the range constraints. These are added by simply constraining all input variables (both public and private) to be less than 2^{128}.

A.1 VALID WALLET CREATE

With Public Variables

- $C(W') \in \mathbb{F}$, the commitment to the new wallet.
- $E_{\text{pk}^{\text{view}}}(W') \in \mathbb{F}^{2M_B+7M_O+2M_F+4}$, the encryption of W' under pk^{view}

I Know Private Variables

- $F \in \mathbb{F}^{2M_F}$
- $\text{pk}^{\text{root}}, \text{sk}^{\text{root}} \in \mathbb{F}$
- $\text{pk}^{\text{match}}, \text{sk}^{\text{match}} \in \mathbb{F}$
- $\text{pk}^{\text{settle}}, \text{sk}^{\text{settle}} \in \mathbb{F}$
- $\text{pk}^{\text{view}}, \text{sk}^{\text{view}} \in \mathbb{F}$
- $r \in \mathbb{F}$

Such That

- $C(W') = C(\mathbf{0}^{2M_B}, \mathbf{0}^{7M_O}, F, \text{pk}^{\text{root}}, \text{pk}^{\text{match}}, \text{pk}^{\text{settle}}, r)$
- $E_{\text{pk}^{\text{view}}}(W')$ is the proper ElGamal encryption of W' under the public key pk^{view}
- pk^{root} is the valid public key corresponding to sk^{root}
- pk^{match} is the valid public key corresponding to sk^{match}
- $\text{pk}^{\text{settle}}$ is the valid public key corresponding to $\text{sk}^{\text{settle}}$
- pk^{view} is the valid public key corresponding to sk^{view}
- $\text{sk}^{\text{match}} = H\left(\text{sk}^{\text{root}}\right)$
- $\text{sk}^{\text{settle}} = H\left(\text{sk}^{\text{match}}\right)$
- $\text{sk}^{\text{view}} = H\left(\text{sk}^{\text{settle}}\right)$

A.2 VALID COMMITMENTS

With Public Variables

- $N^{\text{wallet-match}}(W) \in \mathbb{F}$
- $R_{\text{global}} \in \mathbb{F}$, the current root of the commitment Merkle tree.
- $\text{pk}^{\text{settle}} \in \mathbb{F}$
- $H_o \in \mathbb{F}$, a commitment to an order.
- $H_b \in \mathbb{F}$, a commitment to a balance.
- $H_f \in \mathbb{F}$, a commitment to a fee tuple.

I Know Private Variables

- $W = (B, O, F, \text{pk}^{\text{root}}, \text{pk}^{\text{match}}, \text{pk}^{\text{settle}}, r)$
- $O(C(W), R_{\text{global}})$, a Merkle proof that $C(W)$ is inserted into the commitment tree.
- $\text{sk}^{\text{match}} \in \mathbb{F}$
- $o = (k, m_1, m_2, s, p, a, \tau) \in O$
- $b = (m, v) \in B$
- $f = \left(\text{pk}^{\text{settle}}_{\text{relayer}}, \phi\right) \in F$

Such That

- $N^{\text{wallet-match}}(W)$ is correctly computed.
- $O(C(W), R_{\text{global}})$ is a valid Merkle proof.
- pk^{match} is the valid public key corresponding to sk^{match}
- $H_o = H(o \mid\mid r)$
- $H_b = H(b \mid\mid r)$
- $H_f = H(f \mid\mid r)$
- If $s = 0$, then $m = m_2$
- If $s = 1$, then $m = m_1$

A.3 VALID WALLET UPDATE

With Public Variables

- $C(W') \in \mathbb{F}$, the commitment to the new wallet.
- $E_{\mathrm{pk}^{\mathrm{view}}}(W') \in \mathbb{F}^{2M_B+7M_O+2M_F+4}$, the encryption of W' under $\mathrm{pk}^{\mathrm{view}}$
- $N^{\text{wallet-spend}}(W) \in \mathbb{F}$
- $N^{\text{wallet-match}}(W) \in \mathbb{F}$
- $R_{\mathrm{global}} \in \mathbb{F}$, the current root of the commitment Merkle tree.
- $\mathrm{pk}^{\mathrm{settle}}_{\mathrm{receiver}}$, the settle public key of the user to receive the internal transfer.
- $E_{\mathrm{pk}^{\mathrm{settle}}_{\mathrm{receiver}}}(T_I)$, the encryption of the internal transfer tuple.
- $T_E = \left(\tilde{m}_E, \tilde{v}_E, \tilde{d}_E\right) \in \mathbb{F}^3$, the external transfer tuple.
- τ, the timestamp of this update.

I Know Private Variables

- $W = (B, O, F, \mathrm{pk}^{\mathrm{root}}, \mathrm{pk}^{\mathrm{match}}, \mathrm{pk}^{\mathrm{settle}}, r)$, the old wallet.
- $W' = (B', O', F', \mathrm{pk}^{\mathrm{root}'}, \mathrm{pk}^{\mathrm{match}'}, \mathrm{pk}^{\mathrm{settle}'}, r')$, the new wallet.
- $O(C(W), R_{\mathrm{global}})$, a Merkle proof that $C(W)$ is inserted into the commitment tree.
- $\mathrm{sk}^{\mathrm{root}}, \mathrm{sk}^{\mathrm{match}}, \mathrm{sk}^{\mathrm{settle}}, \mathrm{sk}^{\mathrm{view}} \in \mathbb{F}$
- $T_I = (\tilde{m}_I, \tilde{v}_I) \in \mathbb{F}^2$, the internal transfer tuple.

Such That

- $C(W)$ is correctly computed.
- $C(W')$ is correctly computed.
- $E_{\mathrm{pk}^{\mathrm{view}}}(W')$ is correctly computed.
- $E_{\mathrm{pk}^{\mathrm{settle}}_{\mathrm{receiver}}}(T_I)$ is correctly computed.
- $N^{\text{wallet-spend}}(W)$ is correctly computed.
- $N^{\text{wallet-match}}(W)$ is correctly computed.
- $O(C(W), R_{\mathrm{global}})$ is a valid Merkle proof.
- $\mathrm{pk}^{\mathrm{root}'} = \mathrm{pk}^{\mathrm{root}}$
- $\mathrm{pk}^{\mathrm{match}'} = \mathrm{pk}^{\mathrm{match}}$
- $\mathrm{pk}^{\mathrm{settle}'} = \mathrm{pk}^{\mathrm{settle}}$
- $\mathrm{pk}^{\mathrm{root}}$ is the valid public key corresponding to $\mathrm{sk}^{\mathrm{root}}$
- $\mathrm{pk}^{\mathrm{view}}$ is the valid public key corresponding to $\mathrm{sk}^{\mathrm{view}}$
- $\mathrm{sk}^{\mathrm{match}} = H\left(\mathrm{sk}^{\mathrm{root}}\right)$
- $\mathrm{sk}^{\mathrm{settle}} = H\left(\mathrm{sk}^{\mathrm{match}}\right)$
- $\mathrm{sk}^{\mathrm{view}} = H\left(\mathrm{sk}^{\mathrm{settle}}\right)$
- $\tilde{d}_E \in \{0, 1\}$
- For all balances $(m'_i, v'_i) \in B'$:
  - ► Either $m'_i = 0$, or m'_i is unique in the list of all mints of B'. (i.e., no duplicate mints are allowed).
  - ► v'_i is equal to
  $$\sum_{j \in [M_B] \text{ s.t. } m_j = m'_i} v_j$$
  plus
  $$\mathbf{1}_{m'_i = \tilde{m}_E \wedge \tilde{d}_E = 0} \tilde{v}_E - \mathbf{1}_{m'_i = \tilde{m}_E \wedge \tilde{d}_E = 1} \tilde{v}_E$$
  minus
  $$\mathbf{1}_{m'_i = \tilde{m}_I \wedge \tilde{d}_E = 1} \tilde{v}_I$$
  (i.e., balances are unchanged, except for a deposit or withdraw according to T_E and a transfer according to T_I)
- For all orders $(k'_i, m'_{1_i}, m'_{2_i}, s'_i, p'_i, a'_i, \tau'_i) \in O'$:
  - ► If $k'_i = k_i$ and $m'_{1_i} = m_{1_i}$ and $m'_{2_i} = m_{2_i}$ and $s'_i = s_i$ and $p'_i = p_i$ and $a'_i = a_i$, then $\tau'_i = \tau_i$. Otherwise, $\tau'_i = \tau$.

A.4 Indications of Interest Statements

In this section, we provide the formal specifications for a few different indications of interest. **Note**: We only provide partial statements here; each statement should be conjoined with a proof of VALID COMMITMENTS.

A.4.1 VALID IOI TYPE. Simply add "$k \in \mathbb{F}$" to the public variables.

A.4.2 VALID IOI PAIR. Simply add "$m_1, m_2 \in \mathbb{F}$" to the public variables.

A.4.3 VALID IOI SIDE. Simply add "$s \in \mathbb{F}$" to the public variables.

A.4.4 VALID IOI LIMIT PRICE. Simply add "$p \in \mathbb{F}$" to the public variables.

A.4.5 VALID IOI AMOUNT. Simply add "$a \in \mathbb{F}$" to the public variables.

A.4.6 VALID IOI BALANCE BOUND. Add "$\beta \in \mathbb{F}$" to the public variables and "$v \geq \beta$" to the constraint system.

A.4.7 VALID IOI FEE. Simply add "$\text{pk}^{\text{match}}_{\text{relayer}} \in \mathbb{F}$" and "$f \in \mathbb{F}$" to the public variables.

A.5 VALID MATCH MPC

With Public Variables

- $(m_i, p_i)_{p \in [M_P]} \in \mathbb{F}^{M_P}$, the vector of midpoint oracle prices.
- $H_{o_1}, H_{b_1}, H_{f_1} \in \mathbb{F}$, the hiding commitments to the order, balance, and fee from Relayer 1.
- $H_{o_2}, H_{b_2}, H_{f_2} \in \mathbb{F}$, the hiding commitments to the order, balance, and fee from Relayer 2.
- $H_M \in \mathbb{F}$, the hiding commitment to the matches tuple M.
- $Z_1 \in \mathbb{F}$, the bit that is 1 iff M is a non-trivial matches list.

I Know Private Variables

- $o_1 = (k_1, m_{1_1}, m_{2_1}, s_1, p_1, a_1, \tau_1) \in \mathbb{F}^7$
- $o_2 = (k_2, m_{1_2}, m_{2_2}, s_2, p_2, a_2, \tau_2) \in \mathbb{F}^7$
- $b_1 = (m_1, v_1) \in \mathbb{F}^2$
- $b_2 = (m_2, v_2) \in \mathbb{F}^2$
- $f_1 = \left(\text{pk}^{\text{settle}}_{\text{relayer}_1}, \phi_1\right) \in \mathbb{F}^2$
- $f_2 = \left(\text{pk}^{\text{settle}}_{\text{relayer}_2}, \phi_2\right) \in \mathbb{F}^2$
- $M = (\hat{m}_1, \hat{m}_2, \hat{v}_1, \hat{v}_2, \hat{d}, f_1, f_2, H(r_1), H(r_2)) \in \mathbb{F}^{11}$, the match tuple.
- $Z_2 \in \mathbb{F}$, the **zeroing bit** that equals 0 if any party lies about their secret input w.r.t. to the public commitments $H_{o_1}, H_{o_2}, H_{b_1}, H_{b_2}, H_{f_1}, H_{f_2}$, and equals 1 otherwise.

Such That

- $H_M = H(M)$
- $Z_1 \in \{0, 1\}$
- $Z_2 \in \{0, 1\}$
- $(\hat{v}_1, \hat{v}_2)$ is the output of the matching engine operation on the orders o_1, o_2 under the balance constraints b_1, b_2.
- $\hat{d} = s_1 \cdot (1 - s_2)$
- $$\begin{aligned} Z_2 = &\ \mathbf{1}_{H_{o_1}=H(o_1 \| r_1)} \cdot \mathbf{1}_{H_{o_2}=H(o_2 \| r_2)} \\ &\cdot \mathbf{1}_{H_{b_1}=H(b_1 \| r_1)} \cdot \mathbf{1}_{H_{b_2}=H(b_2 \| r_2)} \\ &\cdot \mathbf{1}_{H_{f_1}=H(f_1 \| r_1)} \cdot \mathbf{1}_{H_{f_2}=H(f_2 \| r_2)} \end{aligned}$$
- $Z_1 = Z_2 \cdot \mathbf{1}_{\hat{v}_1 \neq 0 \vee \hat{v}_2 \neq 0}$

A.6 VALID MATCH LIT

With Public Variables

- $(m_i, p_i)_{p \in [M_P]} \in \mathbb{F}^{M_P}$, the vector of midpoint oracle prices.
- $H_{o_1}, H_{b_1}, H_{f_1} \in \mathbb{F}$, the hiding commitments to the order, balance, and fee from Relayer 1.
- $H_{o_2}, H_{b_2}, H_{f_2} \in \mathbb{F}$, the hiding commitments to the order, balance, and fee from Relayer 2.
- $H_M \in \mathbb{F}$, the hiding commitment to the matches tuple M.
- $Z_1 \in \mathbb{F}$, the bit that is 1 iff M is a non-trivial matches list.
- $o_1 = (k_1, m_{1_1}, m_{2_1}, s_1, p_1, a_1, \tau_1) \in \mathbb{F}^7$
- $b_1 = (m_1, \beta_1) \in \mathbb{F}^2$
- $f_1 = \left(\text{pk}^{\text{settle}}_{\text{relayer}_1}, \phi_1\right) \in \mathbb{F}^2$

I Know Private Variables

- $o_2 = (k_2, m_{1_2}, m_{2_2}, s_2, p_2, a_2, \tau_2) \in \mathbb{F}^7$
- $b_2 = (m_2, v_2) \in \mathbb{F}^2$
- $f_2 = \left(\text{pk}^{\text{settle}}_{\text{relayer}_2}, \phi_2\right) \in \mathbb{F}^2$
- $r_2 \in \mathbb{F}$
- $M = (\hat{m}_1, \hat{m}_2, \hat{v}_1, \hat{v}_2, \hat{d}, f_1, f_2, H(r_1), H(r_2)) \in \mathbb{F}^{11}$, the match tuple.
- $Z_2 \in \mathbb{F}$, the **zeroing bit** that equals 0 if Party 2 (the non-lit party) lies about their secret input w.r.t. to the public commitments H_{o_2}, H_{b_2}, and equals 1 otherwise.[8]

Such That

- $H_M = H(M)$
- $Z_1 \in \{0, 1\}$
- $Z_2 \in \{0, 1\}$
- $(\hat{v}_1, \hat{v}_2)$ is the output of the matching engine operation on the orders o_1, o_2 under the balance constraints b_1, b_2.
- $\hat{d} = s_1 \cdot (1 - s_2)$
- $Z_2 = \mathbf{1}_{H_{o_2} = H(o_2 || r_2)} \cdot \mathbf{1}_{H_{b_2} = H(b_2 || r_2)} \cdot \mathbf{1}_{H_{f_2} = H(f_2 || r_2)}$
- $Z_1 = Z_2 \cdot \mathbf{1}_{\hat{v}_1 \neq 0 \vee \hat{v}_2 \neq 0}$

[8]Note that it is impossible for Party 1 (the lit party) to lie about their input, as they have proven every single IOI, and therefore we know their order o_1 and balance bound b_1 exactly.

A.7 VALID MATCH ENCRYPTION

With Public Variables

- $H_M \in \mathbb{F}$, the hiding commitment to the matches tuple M.
- $\mathrm{pk}_1^{\mathrm{settle}} \in \mathbb{F}$, the settle key of Party 1's wallet.
- $\mathrm{pk}_2^{\mathrm{settle}} \in \mathbb{F}$, the settle key of Party 2's wallet.
- $\mathrm{pk}_{\mathrm{protocol}}^{\mathrm{settle}} \in \mathbb{F}$, the settle key of the global protocol fee wallet.
- $\phi_{\mathrm{protocol}} \in \mathbb{F}$, the global protocol fee value.
- $E_{\mathrm{pk}_1^{\mathrm{settle}}}(N_1)$, the encrypted note for Party 1's settlees.
- $E_{\mathrm{pk}_2^{\mathrm{settle}}}(N_2)$, the encrypted note for Party 2's settlees.
- $E_{\mathrm{pk}_{\mathrm{relayer}_1}^{\mathrm{settle}}}(N_{R_1})$, the encrypted note for Relayer 1's fee.
- $E_{\mathrm{pk}_{\mathrm{relayer}_2}^{\mathrm{settle}}}(N_{R_2})$, the encrypted note for Relayer 2's fee.
- $E_{\mathrm{pk}_{\mathrm{protocol}}^{\mathrm{settle}}}(N_P)$, the encrypted note for the in-protocol fee.

I Know Private Variables

- $M = \begin{pmatrix} \hat{m}_1, \hat{m}_2, \hat{v}_1, \hat{v}_2, \hat{d}, \left(\mathrm{pk}_{\mathrm{relayer}_1}^{\mathrm{settle}}, \phi_1\right), \\ \left(\mathrm{pk}_{\mathrm{relayer}_2}^{\mathrm{settle}}, \phi_2\right), H(r_1), H(r_2) \end{pmatrix}$, the match tuple.
- $N_1 = \left(\hat{m}_1, \hat{v}_1^1, \hat{d}, \hat{m}_2, \hat{v}_2^1, 1 - \hat{d}, 1, H(r_1)\right)$
- $N_2 = \left(\hat{m}_1, \hat{v}_1^2, 1 - \hat{d}, \hat{m}_2, \hat{v}_2^2, \hat{d}, 1, H(r_2)\right)$
- $N_{R_1} = \left(\hat{m}_1, \hat{v}_1^{R_1}, 0, \hat{m}_2, \hat{v}_2^{R_1}, 0, 0, H(r_1)\right)$
- $N_{R_2} = \left(\hat{m}_1, \hat{v}_1^{R_2}, 0, \hat{m}_2, \hat{v}_2^{R_2}, 0, 0, H(r_2)\right)$
- $N_P = \left(\hat{m}_1, \hat{v}_1^P, 0, \hat{m}_2, \hat{v}_2^P, 0, 0, H(r_1) + H(r_2)\right)$

Such That

- $H_M = H(M)$
- N_1 is properly encrypted under $\mathrm{pk}_1^{\mathrm{settle}}$
- N_2 is properly encrypted under $\mathrm{pk}_2^{\mathrm{settle}}$
- N_{R_1} is properly encrypted under $\mathrm{pk}_{\mathrm{relayer}_1}^{\mathrm{settle}}$
- N_{R_2} is properly encrypted under $\mathrm{pk}_{\mathrm{relayer}_2}^{\mathrm{settle}}$
- N_P is properly encrypted under $\mathrm{pk}_{\mathrm{protocol}}^{\mathrm{settle}}$
- $v_1^1 = \hat{v}_1 \cdot \left(1 - \frac{\phi_1 + \phi_{\mathrm{protocol}}}{2}\right)$
- $v_2^1 = \hat{v}_2 \cdot \left(1 - \frac{\phi_1 + \phi_{\mathrm{protocol}}}{2}\right)$
- $v_1^2 = \hat{v}_1 \cdot \left(1 - \frac{\phi_2}{2}\right)$
- $v_2^2 = \hat{v}_2 \cdot \left(1 - \frac{\phi_2}{2}\right)$
- $v_1^{R_1} = \hat{v}_1 \cdot \frac{\phi_1}{2}$
- $v_2^{R_1} = \hat{v}_2 \cdot \frac{\phi_1}{2}$
- $v_1^{R_2} = \hat{v}_1 \cdot \frac{\phi_2}{2}$
- $v_2^{R_2} = \hat{v}_2 \cdot \frac{\phi_2}{2}$
- $v_1^P = \hat{v}_1 \cdot \frac{\phi_{\mathrm{protocol}}}{2}$
- $v_2^P = \hat{v}_2 \cdot \frac{\phi_{\mathrm{protocol}}}{2}$

A.8 VALID SETTLE

With Public Variables

- $C(W') \in \mathbb{F}$, the commitment to the new wallet.
- $E_{\mathrm{pk}^{\mathrm{view}}}(W') \in \mathbb{F}^{2M_B+7M_O+2M_F+4}$, the encryption of W' under $\mathrm{pk}^{\mathrm{view}}$
- $N^{\text{wallet-spend}}(W) \in \mathbb{F}$
- $N^{\text{wallet-match}}(W) \in \mathbb{F}$
- $N^{\text{note-settle}}(N) \in \mathbb{F}$
- $R_{\mathrm{global}} \in \mathbb{F}$, the current root of the commitment Merkle tree.
- $\mu \in \mathbb{F}$, the flag that designates if this settle arises from a match or from a transfer.

I Know Private Variables

- $W = (B, O, F, \mathrm{pk}^{\mathrm{root}}, \mathrm{pk}^{\mathrm{match}}, \mathrm{pk}^{\mathrm{settle}}, r)$, the old wallet.
- $W' = (B', O', F', \mathrm{pk}^{\mathrm{root}'}, \mathrm{pk}^{\mathrm{match}'}, \mathrm{pk}^{\mathrm{settle}'}, r')$, the new wallet.
- $O(C(W), R_{\mathrm{global}})$, a Merkle proof that $C(W)$ is inserted into the commitment tree.
- $\mathrm{sk}^{\mathrm{settle}}, \mathrm{sk}^{\mathrm{view}} \in \mathbb{F}$
- $N = \left(\hat{m}_1, \hat{v}_1, \hat{d}_1, \hat{m}_2, \hat{v}_2, \hat{d}_2, \mu, r\right)$, a note.
- $E_N = E_{\mathrm{pk}^{\mathrm{settle}}}(N)$, an encrypted note.
- $O(E_N, R_{\mathrm{global}})$, a Merkle proof that the encryption of N is inserted into the commitment tree.

Such That

- $C(W)$ is correctly computed.
- $C(W')$ is correctly computed.
- $E_{\mathrm{pk}^{\mathrm{view}}}(W')$ is correctly computed.
- $E_{\mathrm{pk}^{\mathrm{settle}}}(N)$ is correctly computed.
- $N^{\text{wallet-spend}}(W)$ is correctly computed.
- $N^{\text{wallet-match}}(W)$ is correctly computed.
- $N^{\text{note-settle}}(N)$ is correctly computed.
- $O(C(W), R_{\mathrm{global}})$ is a valid Merkle proof.
- $O(E_N, R_{\mathrm{global}})$ is a valid Merkle proof.
- $\mathrm{pk}^{\mathrm{root}'} = \mathrm{pk}^{\mathrm{root}}$
- $\mathrm{pk}^{\mathrm{match}'} = \mathrm{pk}^{\mathrm{match}}$
- $\mathrm{pk}^{\mathrm{settle}'} = \mathrm{pk}^{\mathrm{settle}}$
- $\mathrm{pk}^{\mathrm{settle}}$ is the valid public key corresponding to $\mathrm{sk}^{\mathrm{settle}}$
- $\mathrm{pk}^{\mathrm{view}}$ is the valid public key corresponding to $\mathrm{sk}^{\mathrm{view}}$
- For all balances $(m'_i, v'_i) \in B'$:
  - Either $m'_i = 0$, or m'_i is unique in the list of all mints of B'. (i.e., no duplicate mints are allowed).
  - v'_i is equal to
  $$\sum_{j \in [M_B] \text{ s.t. } m_j = m'_i} v_j$$
  plus
  $$\mathbf{1}_{m'_i = \tilde{m}_1 \wedge \tilde{d}_1 = 0} \tilde{v}_1 - \mathbf{1}_{m'_i = \tilde{m}_1 \wedge \tilde{d}_1 = 1} \tilde{v}_1$$
  plus
  $$\mathbf{1}_{m'_i = \tilde{m}_2 \wedge \tilde{d}_2 = 0} \tilde{v}_2 - \mathbf{1}_{m'_i = \tilde{m}_2 \wedge \tilde{d}_2 = 1} \tilde{v}_2$$
  (i.e., balances are unchanged, except for an increase or decrease according to N)
- For all orders $(k'_i, m'_{1_i}, m'_{2_i}, s'_i, p'_i, a'_i, \tau'_i) \in O'$:
  - $k'_i = k_i$, $m'_{1_i} = m_{1_i}$, $m'_{2_i} = m_{2_i}$, $s'_i = s_i$, $p'_i = p_i$, $\tau'_i = \tau_i$
  - a'_i is equal to
  $$\sum_{j \in [M_O] \text{ s.t. } \mu = 1 \wedge m_{1_j} = m'_{1_i} \wedge m_{2_j} = m'_{2_j}} a_j$$
  minus
  $$\mathbf{1}_{m'_{1_i} = \hat{m}_1 \wedge m'_{2_i} = \hat{m}_2} \cdot \hat{v}_1$$
  (i.e., the orders are unchanged, except for a decrease of the amount corresponding to the matched value)