Contents

1 Dark automated match engine.

Non-custodial order book over darkfi blokchcain.

1.1 order type

the match order only accept limit orders.

1.2 exchange transaction

Assume a liquidity provider owns a coin c of Token t_1 in order to add an order in the order book with public key $pk^{withdraw}$, the LP burns c, mint new $c' = (pk_1^{withdraw}, v_1, t_1, spendhook_1, userdata_1, coinblind_1, tokenblind_1)$, after validating transfer call, mint an order with quote token id t_2, $tokenblind_2$, quote rate p (quote price per base price), timeout duration, τ, constrain order commit for order o = (t_2, p, τ), the proof need to commit to withdraw public key $pk^{withdraw}$ that can be used in case of withdraw or settlement.

1.2.1 Order ciruit

```
circuit "Order" {
    # Poseidon hash of the order
    O = poseidon_hash(
        withdraw_public_x,
        withdraw_public_y,
        base_value,
        quote_value,
        base_token_id,
        quote_token_id,
        spend_hook,
        user_data,
        order_blind,
        timeout_duration_blind,
    );
    constrain_instance(O);

    # Pedersen commitment for order's base order_value
    base_vcv = ec_mul_short(base_value, VALUE_COMMIT_VALUE);
    base_vcr = ec_mul(base_value_blind, VALUE_COMMIT_RANDOM);
    base_order_value_commit = ec_add(base_vcv, base_vcr);
    # Since the base_value commit is a curve point, we fetch its coordinates
    # and constrain them:
    constrain_instance(ec_get_x(base_order_value_commit));
    constrain_instance(ec_get_y(base_order_value_commit));

    # Pedersen commitment for order's quote_value
    quote_vcv = ec_mul_short(quote_value, VALUE_COMMIT_VALUE);
    quote_vcr = ec_mul(quote_value_blind, VALUE_COMMIT_RANDOM);
```

```
    quote_order_value_commit = ec_add(quote_vcv, quote_vcr);
    # Since the quote_value commit is a curve point, we fetch its coordinates
    # and constrain them:
    constrain_instance(ec_get_x(quote_order_value_commit));
    constrain_instance(ec_get_y(quote_order_value_commit));

    # Commitment for order's base_token_id ID. We do a poseidon hash since it's
    # cheaper than EC operations and doesn't need the homomorphic prop.
    base_order_token_id_commit = poseidon_hash(base_token_id, base_token_id_blind);
    constrain_instance(base_order_token_id_commit);

    # Commitment for order's quote_token_id ID. We do a poseidon hash since it's
    # cheaper than EC operations and doesn't need the homomorphic prop.
    quote_order_token_id_commit = poseidon_hash(quote_token_id, quote_token_id_blind);
    constrain_instance(quote_order_token_id_commit);

    # Commitment for order's timeout_duration.  We do a poseidon hash since it's
    # cheaper than EC operations and doesn't need the homomorphic prop.
    timeout_duration_commit = poseidon_hash(timeout_duration, timeout_duration_blind);
    constrain_instance(timeout_duration_commit):

    # At this point we've enforced all of our public inputs.
}
```

1.2.2 send the gas fee in the order details (TODO)

exchange proof need to commit to gas fee value, gas fee token.

1.3 match proof

It's a proof of knowledge that the match engines have access two coins with non-negative spread (this is two steps, the coin values are sufficient, and the rate matches), two coins are of the opposite direction, the counter party token is the desired token. for a match between two parties left (l), and right (r) prove konwledge to two coins l, r, such that:

base token of the left token is the same as quote of the right token, quote of the left token is the same as base of the right token, the exchange rate of both parties has non-negative spread, any positive spread is added to the order book.

$$t_1^l == t_2^r$$
$$t_2^l == t_1^r$$

the following make sure that the two parties has non-negative spread.

$$p_1 p_2 <= 1$$

constrain, and reveal the two coins used.

1.3.1 Order match proof circuit

```
circuit "OrderMatch" {
    # Match base, and quote pairs tokens for first, and second orders.
    constrain_equal_base(base_token_id_1, quote_token_id_2);
    constrain_equal_base(quote_token_id_2, base_token_id_1);

    # Match base, and quote values for first and second orders.
    Base quote_1_mul_quote_2 = base_mul(quote_value_1, quote_value_2);
    Base base_1_mul_base_2 = base_mul(base_value_1, base_value_2);
    Base q1q2_sub_b1b2 = base_sub(quote_1_mul_quote_2, base_1_mul_base_2);
    ZERO = witness_base(0);
    less_than_strict(q1q1_sub_b1b2, ZERO);
```

```
}
```

1.3.2 spread difference (TODO)

the spread should be added to the book, commit to the spread difference value, it will be spent along with the fee transaction upon swap success.

1.4 order settlement transaction

settlement node, would combine match call with transfer call in a single transaction.

1.5 withdraw transaction

LP who issued a exchange order can withdraw any moment as long as the settlement tx isn't broadcasted yet.

1.5.1 exchange transaction spend hook

if the duration τ specified in the order is timeout, the spend hook's withdraw is called.

1.6 cancel order

at any moment as long as the order isn't settled, the LP can cancel the order by minting a new coin, or if the order duration is timedout, can mint new Coin of the same Token and value back to the LP with $pk^{withdraw}$