

# The Arwen Trading Protocols (Full Version) <sup>\*</sup>

Ethan Heilman, Sebastien Lipmann, Sharon Goldberg

Arwen (arwen.io)

**Abstract.** The Arwen Trading Protocols are layer-two blockchain protocols for traders to securely trade cryptocurrencies at a centralized exchange, without ceding custody of their coins to the exchange. Before trading begins, traders deposit their coins in an on-blockchain escrow where the agent of escrow is the blockchain itself. Each trade is backed by the coins locked in escrow. Each trade is fast, because it happens off-blockchain, and secure, because atomic swaps prevent even a hacked exchange from taking custody of a trader’s coins. Arwen is designed to work even with the “lowest common denominator” of blockchains—namely Bitcoin-derived coins without SegWit support. As a result, Arwen supports essentially all “Bitcoin-derived” coins *e.g.*, BTC, LTC, BCH, ZEC, as well as Ethereum. Our protocols support Limit and RFQ order types, we implemented our RFQ protocol and are available for use at arwen.io.

## 1 Introduction

The promise of blockchain-backed cryptocurrencies is the ability to transact without relying on a single trusted party. Blockchains therefore present a breakthrough that circumvents a long-standing result in cryptography: namely, that *atomic swaps* are impossible without the help of a trusted third party [34]. In an atomic swap, two parties that do not trust each other swap items, such that either (1) the swap occurs, OR (2) each party reclaims their item. Atomic swaps of digital assets are possible when the blockchain acts as the trusted third party [8].

The Arwen Trading Protocols seek to deliver on this promise by bringing atomic swaps to the mainstream use case of cryptocurrency trading. With Arwen, traders benefit from the liquidity at centralized cryptocurrency exchanges without trusting the exchange with custody of their coins. Arwen traders maintain custody of their cryptographic keys and their coins. Each coin’s native blockchain acts as the agent of escrow. Arwen trades are fast because they happen off blockchain, and secure, because they are atomic swaps. We have implemented and deployed the Arwen trading RFQ protocol. It is currently enabling atomic swaps between Bitcoin (BTC), Bitcoin-cash (BCH), Litecoin (LTC) and Ethereum (ETH) on one of the largest global cryptocurrency exchanges, Kucoin[2].

---

<sup>\*</sup> Major contributions to the design of these protocols were made by James Dalessandro, Ezequiel Gomes Perez, Haydn Kennedy, Yuval Marcus, Chet Powers, Omar Sagga, Aleksander Skjolsvik and Scott Sigel.

Our protocols are specifically designed for the trading use case and supports trading instruments from traditional finance such as RFQs (Request For Quote) and limit orders. RFQs are a valuable trading instrument for atomic trades as they allow traders to swap coins immediately at current market prices. We use RFQs instead of market orders because in an RFQ, the trader learns the price the order will execute at before agreeing to execute the order, whereas in a market order the trader has no recourse if the exchange sets an absurdly low price. Limit orders are a basic and critical tool since they let a trader set their own price on an exchange’s order book.

In Section 2 we discuss issues hampering mainstream atomic swap adoption and how Arwen overcomes them. Section 3 provides an overview of Arwen followed by our protocol for RFQs (Section 4) and limit orders (Section 5). Finally we compare Arwen to related work (Section 6).

## 2 Whither Atomic Swaps?

*Cross-blockchain atomic swaps* seek to supplant today’s dominant form of cryptocurrency trading: *custodial trading* at centralized exchanges. With custodial trading, when users wish to trade they must first deposit their coins at the exchange; this is done using an on-blockchain transfer of coins from the user to the exchange. Trading occurs within the databases of the centralized exchange, and is not recorded on the blockchain. Finally, users can take custody of their coins by withdrawing from the exchange; that is, the exchange uses an on-blockchain transaction to send coins from the exchange back to the user. Custodial trading at a centralized exchange exposes users to serious counterparty risk—the exchange may be unable to transfer coin back to the user’s wallet. This risk has been realized, starting with the hack of MtGox [46] and continuing to the present [11,39,21,9,30,14,23,22,40,7,48].

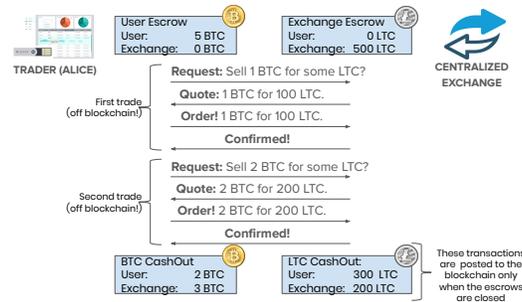


Fig. 1. Arwen Trading Protocol for two RFQ trades between the user and exchange.

**The Bitcoin TierNolan Protocol.** The TierNolan protocol [43] is the original Bitcoin-compatible atomic swap; it can also be used for cross-blockchain atomic

swaps for “Bitcoin-like” blockchains (*e.g.*, BCH, LTC, ZEC, *etc.*). TierNolan uses *Hashed Time-Locked Contract (HTLC)* smart contracts as follows.

Bob chooses a random *solution*  $x$  and computes a *puzzle*  $y$ , where  $y = H(x)$  and  $H$  is a cryptographic hash function. Bob reveals the puzzle  $y$  to Alice and keeps  $x$  secret. Next, Bob locks up 100 LTC in an HTLC smart contract on the Litecoin blockchain which stipulates: “before time  $tw_B$ , 100 LTC can be claimed by a transaction signed by Alice containing the solution to puzzle  $y$ ”. Alice similarly locks up 1 BTC on the Bitcoin blockchain in an HTLC, which stipulates: “before time  $tw_A$ , the 1 BTC can be claimed by a transaction signed by Bob containing the solution to the puzzle  $y$ ”. The atomic swap executes when Bob claims 1 BTC by posting a transaction to the Bitcoin blockchain containing  $x$ . Thus, Alice learns  $x$  and can post her solve transaction to the Litecoin blockchain and claim 100 LTC. Security follows from the fact that Bob must reveal  $x$  in order to claim his coins.

There are number of subtle issues that prevent current non-custodial trading solutions from seeing widespread adoption for cryptocurrency trading. Below we highlight several of these issues, and explain how Arwen overcomes them.

**The challenge of providing liquidity.** Most decentralized exchange (DEX) protocols, including EtherDelta [1], 0x [45], and SparkSwap [5], are peer-to-peer trading systems; Each trade involves a transfer of funds directly from trader Alice’s wallet to trader Bob’s wallet. The peer-to-peer approach limits liquidity, because Alice can only trade with traders that use that same peer-to-peer trading system. If a system has too few users, it will not be able to provide good liquidity.

Arwen eschews the peer-to-peer approach because, today, the best liquidity for cryptocurrency trading is found at centralized exchanges. With Arwen, Alice can benefit from the liquidity at a centralized exchange even if she is the only Arwen user at the exchange.

**The pitfalls of on-blockchain protocols.** On-blockchain protocols such as TierNolan, EtherDelta [1] and 0x [45] suffer from slow trade execution because they are bound by the speed at which blockchains confirms blocks. Many confirmations are often required to ensure a transaction can not be reversed [17] *e.g.*, the cryptocurrency exchange Kraken waits 6 confirmations (60 minutes) for BTC and 30 confirmations for ETH (6 minutes) [25]. When trading, even a few seconds of latency is problematic, especially given the famously volatile cryptocurrency prices. Even worse, if every single trade must be confirmed on-blockchain, and a healthy trading ecosystem leads to many trades, then the blockchains involved will be clogged with transactions resulting from each trade.

Ethereum DEX protocols *e.g.*, EtherDelta and 0x, use the Ethereum blockchain to trade one ERC-20 token for another ERC-20 token. In EtherDelta and 0x Alice first broadcasts an order to the network without identifying a counterparty. A counterparty Bob then sees Alice’s broadcast, decides to trade with Alice, and adds his information to the order. Bob then posts the order to the blockchain. Anyone can learn the details of Alice’s trade with Bob, and attempt to profit from it by front-running Bob’s trade [45,16].

Arwen avoids these speed, scalability and frontrunning pitfalls, because trades execute off-blockchain.

**Dealing with lockup griefing.** Lockup griefing affects any protocol that requires users to lock coins in a smart contract. In TierNolan, Alice and Bob’s coins are locked in smart contracts until the trade executes or the timelock on the smart contracts expires. To ensure the security of the swap, timelocks are generally a few hours long. These long expiry times creates a “lock-up griefing” problem where one party (Alice or Bob) tricks the other into pointlessly locking coins in the smart contract.

In Arwen the exchange has no incentive to launch a lockup griefing attack; such an attack harms the exchange’s reputation, and prevents Alice from trading, which is the exchange’s main source of revenue. The exchange, however, must protect itself from Alice who might ask the exchange to lock up coins without the intention to trade. Arwen introduces a novel escrow fee mechanism (see Section 3.2) that compensates the exchange for locking up coins while rewarding Alice for unlocking the exchange’s coins in a timely manner.

**Atomic swaps as trading instruments.** To use atomic swaps to provide traditional trading instruments Arwen must avoid a misalignment of incentives. We’ve already discussed how Arwen aligns incentives of opening escrows; we now focus on trading incentives. Let’s revisit the TierNolan protocol.

The TierNolan Protocol is asymmetric as only Bob knows the secret solution  $x$ . This means that Bob has the unilateral ability to decide whether to execute the atomic swap by revealing  $x$  (or not). Because the timelocks  $tw_A, tw_B$  on the smart contracts must at least be as long as the time it takes to confirm transactions on the blockchain, Bob has minutes or hours to decide whether market conditions justify the execution of the swap (or not). This means that the TierNolan Protocol is actually an *American call option*: namely, Bob has the right, but not the obligation, to buy 1 BTC from Alice at a strike price of 100 LTC, any time before the expiry time  $tw_A$ . Typically, the asymmetry in an option is handled by requiring Bob to pay a premium to Alice before the option is set up. However, in TierNolan Bob gets the option for free, resulting in a misalignment of incentives.

Arwen is explicitly designed to support additional trading instruments beyond the American call option. For example in Arwen’s RFQ trade, the exchange commits to a price, called the *quote*, before Alice decides whether or not to place an order for the trade. (Quote: “You can buy 40 BCH, quote open for 1 second”). Importantly, RFQs are inherently asymmetric, because Alice gets to decide whether the trade executes. Therefore, to align incentives, the exchange’s quote includes a spread around the current price compensating the exchange for price movements after the quote is given. If the exchange is unable to execute a trade against a quote it provided, the exchange can abort the trade. While no coins are lost, this is sufficiently harmful to the exchange’s reputation that we would expect an exchange to avoid aborting if possible.

### 3 Arwen overview

The Arwen Trading Protocol is a blockchain-backed two-party cryptographic protocol between a user Alice and a centralized exchange. Alice first locks her coins in an on-blockchain *user escrow*. Next, Alice asks the exchange to lock its coins in an on-blockchain *exchange escrow*. To compensate the exchange for locking up its coins, Alice pays an *escrow fee* to the exchange from Alice’s user escrow. Each trade is an off-blockchain atomic swap. From these we build non-custodial unidirectional trading instruments for RFQs (Section 4) and limit orders (Section 5). In Appendix A we show how to modify our unidirectional protocols to be bidirectional.

#### 3.1 On-blockchain escrows.

Escrows are opened and closed by confirming a transaction on the coin’s native blockchain. Opening and closing escrows takes the same amount of time it would take to deposit or withdraw coins from a custodial centralized exchange.

Lets look at an example. Alice wishes to trade bitcoins for litecoins as shown in Figure 1. Alice funds the on-blockchain *user escrow*. The user escrow locks *e.g.*, 5 BTC from the user’s wallet on the Bitcoin blockchain until the pre-agreed-upon expiry time  $tw_A$ . The initial balance in this escrow is 5 BTC owned by the user, and 0 BTC owned by the exchange. The exchange funds the *exchange escrow*. To open the exchange escrow, Alice pays the exchange an escrow fee, as described in Section 3.2. The exchange escrow locks 500 LTC from the exchange’s wallet on Litecoin’s blockchain until some pre-agreed-upon expiry time  $tw_B$ . The initial balance in this escrow is 0 LTC owned by the user, and 500 LTC owned by the exchange.

**Escrow smart contracts.** The Arwen escrow is a timelocked two-of-two multisig smart contract that stipulates the following:

“spending requires joint signatures of the user and the exchange, OR after time  $tw$  only the signature of the party that funded this escrow.”

Escrows come with an expiry time that protect each party against a malicious counterparty. Escrow expiry times can vary, but must be longer than the time needed to reliably confirm a transaction on blockchain.

If the exchange and user are cooperative then escrows can be closed at any time, even before they expire. Each escrow is closed via a jointly signed *cashout transaction*, posted to the blockchain, that reflects the balance of the escrow. If either counterparty is malicious the other party can unilaterally recover their funds. These unilateral recovery procedures are specific to each of Arwen’s trading instruments.

Arwen escrow smart contracts (see Appendix D) are written in *Bitcoin-script* allowing support for BTC, BCH, LTC, ZEC, *etc.*. The Ethereum implementation of Arwen leverages the greater functionality of Ethereum smart contracts to

replicate the properties of the Bitcoin-script smart contracts (See Appendix B). Script and contract source available on github<sup>1</sup>.

### 3.2 Arwen's escrow fee mechanism.

When an exchange funds an exchange escrow for a specific user Alice (*e.g.*, the 500 LTC exchange escrow in Figure 1), the exchange is locking coins in an escrow that can only be used by Alice. These coins can come out of the exchange's own inventory. Alternatively, they could be coins deposited by custodial users that the exchange uses to fund escrows, in exchange for earning interest on those deposits.

For this reason, when Alice requests an exchange escrow, she first pays an *escrow fee* to compensate the exchange for locking up its funds. Arwen's escrow fees are an in-band mechanism that avoids the introduction of out-of-band payments or of a superfluous fee token.

**The escrow fee mechanism.** The escrow fee is proportional to the amount of coin locked in the exchange escrow, and to the expiry time of the exchange escrow. Alice pays the escrow fee upfront, before she opens the exchange escrow. Alice receives a rebate of a portion of the escrow fee if she closes the exchange escrow early, before it expires.

Alice pays the upfront escrow fee via a fast off-blockchain transfer out of the coins locked in one of her user escrows. Alice receives the rebate out of the exchange escrow, once that exchange escrow is closed.

**Paying escrow fees.** This is best illustrated with an example. Consider the situation in Figure 1, and suppose that Alice has an open user escrow with a balance of 5 BTC owned by Alice. Alice then asks the exchange to open a 500 LTC exchange escrow for her that expires two days later, and indicates that she can pay the escrow fee out of her BTC user escrow.

Suppose the escrow fee for the requested exchange escrow is 1 LTC/day and Alice decides to pay the upfront escrow fee using her BTC user escrow. First, the exchange performs a currency conversion of the escrow fee, converting it from 2 LTC into 0.02 BTC. Next, the exchange quotes an escrow fee of 0.02 BTC to Alice. If Alice accepts this fee, Alice sends the exchange a 0.02 BTC off-blockchain payment from her user escrow that alters the balance in the user escrow so that the exchange owns 0.02 additional BTC and Alice owns 4.98 BTC. Once the exchange receives this payment, the exchange funds an exchange escrow for Alice for 500 LTC.

**Escrow fee rebate.** Now suppose that Alice has made trades that alter the balance in the exchange escrow so that 300 LTC is owned by Alice and 200 LTC is owned by the exchange. Alice then decides to close her exchange escrow one day early, so she is entitled to a escrow-fee rebate of 1 LTC. Alice is paid the rebate out of the closed exchange escrow. Thus, the exchange escrow is closed with a

---

<sup>1</sup> <https://github.com/cwcrypto/arwen-eth-contracts>  
<https://github.com/cwcrypto/arwen-btc-scripts>

balance of 301 LTC sent to Alice’s wallet and 199 LTC sent to the exchange’s wallet.

### 3.3 Security model.

Arwen assumes the exchange is almost always online, while the user is usually not online. Atomic swap security for users of Arwen assumes (1). The traded coins’ native blockchain is secure *i.e.*, when selling or buying bitcoins we assume Bitcoin’s blockchain is secure. (2). The user comes online in order to recover coins from frozen escrows during their coin-recovery time period, and to close escrows in a “timely manner”. Each Arwen protocol has a specific definition of what it means to close escrows in “timely manner”.

## 4 Unidirectional RFQs

The following protocol is *unidirectional* [41] because it only allows Alice to sell coins from her user escrow, and buy coins to her exchange escrow. Arwen’s more complex *bidirectional* RFQ protocol is described in Appendix A. We show how to port this same logic to Ethereum in Appendix B.

Each off-blockchain RFQ trade is backed by a user escrow (with expiry time  $tw_{\mathcal{A}}$ ) and an exchange escrow (with expiry time  $tw_{\mathcal{B}}$ ). The protocol for opening these escrows is in Section 3.1. Each trade generates a pair of *puzzle transactions* for puzzle  $y = H(x)$  and solution chosen by the exchange  $x$ . One puzzle transaction spends the user escrow and has timelock  $\tau_{\mathcal{A}}$ , and the other spends the exchange escrow and has timelock  $\tau_{\mathcal{B}}$ . Each pair of puzzle transactions reflects the new balance of coins in the escrows after the trade, and “overwrites” the transactions from previous trades. This protocol enables each party to *unilaterally* close escrows with the correct balance even if the other party is malicious.

### 4.1 Security assumptions.

**Timelocks.** Security of this protocol follows from setting the timelocks to be

$$\tau_{\mathcal{A}} = tw_{\mathcal{A}} \quad \tau_{\mathcal{B}} = \max(tw_{\mathcal{B}}, \tau_{\mathcal{A}} + 2\varrho) \quad (1)$$

where  $\varrho$  is the time required for a transaction be reliably confirmed on the blockchain. There is no relationship between the escrow expiry times ( $tw_{\mathcal{A}}, tw_{\mathcal{B}}$ ). We can pair any user escrow and exchange escrow regardless of expiry time.

**Closing escrows in a timely manner.** To withstand attacks by a malicious exchange the user must close her exchange escrow before it expires at time  $tw_{\mathcal{B}}$ . If the user forgets to do this, an honest exchange will close the escrow on the user’s behalf, but a malicious exchange may be able to steal coins from the escrow. This requirement is for exchange escrows only; there is no requirement that the user close her user escrows in a timely manner. Similarly, to withstand attacks by a compromised or malicious user the exchange must close its user escrow before it expires at time  $tw_{\mathcal{A}}$ . Finally, the time period in which the user can unilaterally recover coins from frozen escrows is  $(tw_{\mathcal{A}}, \tau_{\mathcal{B}})$ .

## 4.2 Off-blockchain RFQ trades.

As shown in Figure 1 we suppose that Alice wants to do a trade, selling 2 bitcoins for 200 litecoins. We also assume that, in all previous successfully-completed trades, Alice has sold at total 1 BTC from the user escrow and 100 LTC from the exchange escrow that are backing the current trade. Each RFQ is an off-blockchain four-message protocol comprising the following four messages.

**Request.** Alice requests a quote to sell 2 BTC in order to buy LTC.

**Quote.** The exchange responds with the quote—“2 BTC can be sold for 200 LTC, open for time  $\delta$ ”. The exchange has now committed to executing the trade should Alice choose to place an order before the quote expires at time  $\delta$ .

To commit to the quote, the exchange chooses a secret  $x$  and computes a puzzle  $y = H(x)$ . The exchange sends Alice a Litecoin *puzzle transaction* signed by the exchange’s key, spending the output of the exchange escrow, and reflecting the current balance in the LTC exchange escrow, except that 200 LTC is locked in an HTLC smart contract stipulating

”spending requires the user’s signature and the solution to puzzle  $y$ , OR  
after time  $\tau_B$  only exchange’s signature”

**Order.** If the user decides not to place the order, then the escrows remain open and can be used for other trades.

To place an order, Alice signs and sends the exchange a new Bitcoin *puzzle transaction* using the same puzzle  $y$  chosen by the exchange. The puzzle transaction spends the output of the user escrow and reflects the current balance in the user escrow, except that 2 BTC is locked in an HTLC smart contract stipulating

”spending requires exchange’s signature and the solution to puzzle  $y$  OR  
after time  $\tau_A$  only user’s signature”

At this point the exchange can now unilaterally decide whether or not the trade executes. (This follows because the exchange can use this puzzle transaction, and the solution  $x$ , to unilaterally close the user escrow).

**Execute.** If the user placed the order before time  $\delta$ , then the exchange is expected to execute the trade by releasing  $x$ . After which both Alice and the exchange hold transactions that allow them to unilaterally close their escrows, reflecting the new balance after the trade. (the user can unilaterally close the exchange escrow; the exchange can unilaterally close the user escrow.) In most situations the user will prefer to keep trading against her open escrows. In this case, no transactions are posted to the blockchain and both escrows remain open.

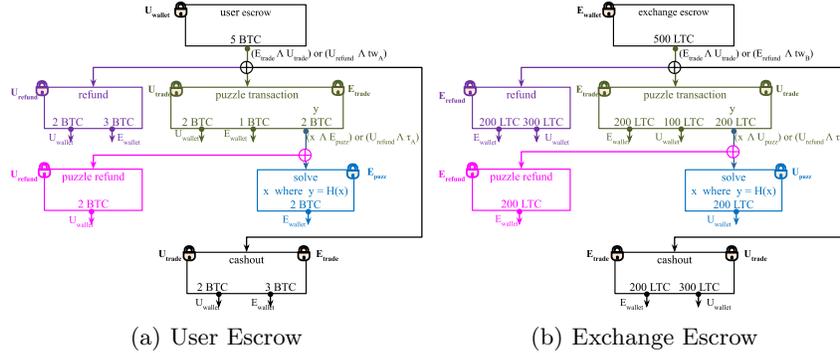
If the exchange does not properly execute the trade by releasing  $x$  Alice will *freeze* the user escrow and exchange escrow that backed the aborted trade and launch a procedure for recovering her coins, as described in Section 4.5.

## 4.3 The magic of unidirectionality.

The security of our protocol follows, in part, from an observation made by Spilman [41]. This is a unidirectional protocol, which means that the user can

only use the exchange escrow to buy coins from the exchange. Thus, each subsequent trade changes the balance of coins in the exchange escrow such that the user holds more litecoins and the exchange holds less litecoins. For this reason, the user will always prefer to post the transactions resulting from the most recent trade to the Litecoin blockchain. This is why the Litecoin transactions resulting from a new trade will “overwrite” the Litecoin transactions of the previous trade. Both parties are incentivized to close the escrow they funded before it expires using transactions from the most recent trade. If a party goes rogue and closes the escrow they funded using transactions from a prior trade they only hurt themselves (they get fewer coins, their counterparty gets more coins)!

**Paying escrow fees.** Unidirectionality makes it easy for the Alice to pay escrow fees out of her user escrow. Suppose that, after the second trade in Figure 1, Alice wishes to pay an 0.02 BTC escrow fee to open a new exchange escrow. To do this, Alice signs and sends the exchange a *cashout transaction* that reflects the current balance of the user escrow, with an additional 0.02 BTC allocated to the exchange. The same unidirectional argument means that the exchange is incentivized to have this cashout transaction “overwrite” the puzzle transaction received from the previous trade.



**Fig. 2.** Unidirectional RFQ protocol transaction diagram. Balances are per Figure 1. Green and blue transactions unilaterally close the escrow if a counterparty is uncooperative. Purple transactions refund the escrow after it expires at time  $tw_A$  or  $tw_B$ . Magenta transactions refund the puzzle transactions after the expiry time  $\tau_A$  or  $\tau_B$ . The  $\oplus$  symbol is an XOR: only one of the transactions from the  $\oplus$  can be posted to the blockchain. The lock symbol represents a signature.

#### 4.4 Cooperative close

If neither the user or the exchange are unresponsive or malicious, escrows can be closed prior to their expiry using the cooperative close. Both parties jointly sign and post *cashout transactions* spending and reflecting the final balance of each

escrow. Cooperatively closing is in the interest of both parties. It reduces mining fees by closing an escrow with a single transaction rather than two (*i.e.*, the puzzle and solve transactions) and a cooperative close of the exchange escrow rebates the user some escrow fees.

#### 4.5 Unilaterally closing an open escrow

What happens if the user and exchange fail to cooperatively close an escrow?

First we consider the case where all trades against the escrow have properly completed. If the exchange refuses to close an exchange escrow before time  $tw_B$  Alice signs and posts the latest puzzle and solve transactions releasing the final balance to both parties. If Alice does not close the exchange escrow before time  $tw_B$  the exchange can unilaterally close the exchange escrow after it expires at time  $tw_B$  using a *refund transaction* (Figure 2(b)). If the user Alice forgets to close the user escrow before time  $tw_A$ , then the exchange signs and posts the latest puzzle and solve transactions unilaterally closing the user escrow. If the exchange refuses to close the user escrow, the user waits until the user escrow expires at  $tw_A$ , and unilaterally closes the user escrow via a *refund transaction*.

Next we consider the case where Alice places an order against a quote provided by the exchange, but the exchange does not release the preimage  $x$ . Alice asks the exchange to cooperatively close the user escrow backing this trade. If the exchange refuses Alice unilaterally closes the exchange escrow by posting the puzzle transaction from the aborted trade. The coins from the aborted trade are now locked in the puzzle transaction's smart contract until time  $\tau_B$ . We call these coins the *outstanding coins*. If the exchange executes the aborted trade the outstanding coins belong to Alice; otherwise, the outstanding coins belong to the exchange. To claim the outstanding coins whenever they are rightfully hers, Alice comes online during time window  $(tw_A, \tau_B)$  and performs the correct action for each case:

**User escrow closed using a successful trade.** The exchange closed the user escrow on the Bitcoin blockchain via a puzzle transaction for any trade *prior* to the aborted trade. No further action is needed from Alice. The outstanding coins rightfully belong to the exchange. The exchange uses a puzzle-refund transaction to unilaterally claim the coins once the timelock  $\tau_B$  expires.

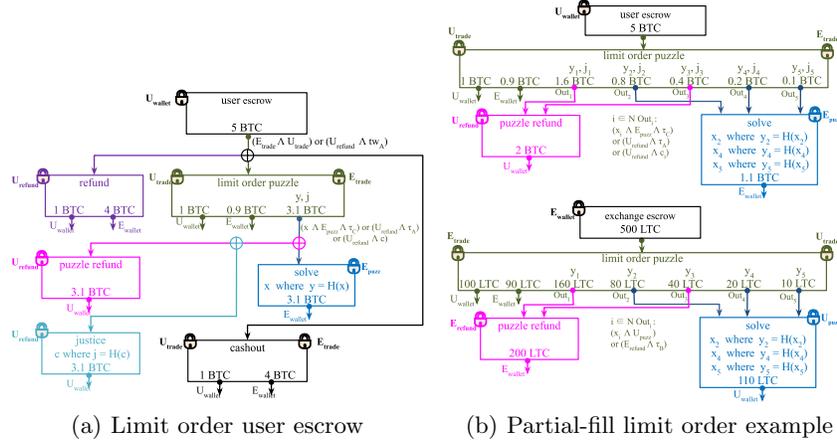
**User escrow closed using the aborted trade.** The exchange closed the user escrow on the Bitcoin blockchain via a puzzle transaction for the aborted trade, as well as its corresponding solve transaction. Alice learns the solution  $x$  from the Bitcoin solve transaction and uses  $x$  to claim her coins on the Litecoin blockchain via a solve transaction. She must complete this action before  $\tau_B$  as the outstanding coins can be unilaterally claimed by the exchange after  $\tau_B$ .

**User escrow partially closed.** The exchange posted the puzzle transaction for the aborted trade, but the coins locked in this puzzle transaction on the *Bitcoin blockchain* are unspent. Alice recovers the coins locked in the puzzle output from the *user escrow* by unilaterally posting a puzzle-refund transaction to the Bitcoin blockchain after the timelock expires at time  $\tau_A$ .

**User escrow not closed.** The exchange did not execute the aborted trade. To recover her coins in the user escrow Alice posts the refund transaction. This must be done after the user escrow expires at  $tw_A$  and before  $\tau_B$ .

#### 4.6 Deployment status

We implemented the unidirectional RFQ protocol described in this section. A release of our trading software is currently available for download enabling users to atomically trade on the orderbook of the centralized exchange kucoin. We support BTC, BCH, LTC and ETH on their respect mainnets. Our client is composed of a daemon written in C# which acts as the user’s agent in the protocol and a graphical interface written in typescript. The other protocols described in this paper *e.g.*, limit orders, have not yet been implemented.



**Fig. 3.** (a). User Escrow modified for limit orders by adding a cancel condition on the puzzle output (b). Unilaterally closed partial-fill limit order with  $N = 5$  puzzle outputs

## 5 Limit Orders

In this section we introduce off-blockchain atomic trading protocols for *All-or-None (AoN) limit orders* and *partial-fill limit orders*. Our limit order protocols allow the user Alice to place a order for a specified amount and limit price against a (user escrow, exchange escrow) pair. For example, Alice might say “I will sell 3.1 BTC at the price of 1 BTC for 100 LTC”. In our *All-or-None limit order*, this order would remain open until the limit price is met for the entire amount, then the exchange would execute the entire order (*e.g.*, Alice sells 3.1 BTC and buys 310 LTC). In our partial-fill limit order the exchange can execute or fill

the order in increments *e.g.*, the exchange could execute the trade 0.3 BTC for 30 LTC. Then later when the price is met again, the exchange could fill (*aka*, execute) an additional trade of 0.8 BTC for 80 LTC. Unlike RFQs, limit orders can remain open for long periods of time. The user can cancel her limit order at any time. When the user cancels a partial-fill limit order, she only cancels the unfilled part of the order (*e.g.*, if Alice’s order has already filled for 110 LTC, the remaining 200 LTC of the order is canceled, with result being that Alice sold 1.1 BTC to buy 110 LTC).

Technically speaking, our limit orders protocols and transactions are very similar to our RFQ protocol (Section 4) with one exception. We add the ability for the exchange to “cancel” a limit order after the user places it. To ensure the exchange can not steal the user’s funds by posting canceled orders, our cancel functionality must be cryptographically enforceable by the user. This change is necessary because limit orders, unlike RFQs, are not designed to execute immediately and can stay open indefinitely. Users often cancel and reissue limit orders depending on market conditions.

**Canceling user escrow puzzles.**

We modify the user escrow puzzle transaction so the puzzle output stimulates:

“spending requires the user’s signature and the cancel value  $c$   
OR after time  $\tau_C$  the exchanges signature and the solution to puzzle  $y$   
OR after time  $\tau_A$  the user’s signature .”

Figure 3(a) shows our modified user escrow. For each user escrow puzzle transaction the exchange randomly chooses a secret cancel value  $c$ , hashes it to generate  $j = H(c)$ , and uses  $j$  as the cancel condition in the puzzle transaction puzzle output. When the exchange wishes to cancel the puzzle output, it sends  $c$  to the user. We say the output is “canceled” because, if the exchange misbehaves by posting the transaction that contains that output, the then user can retaliate and claim all coins in the canceled output at anytime before time  $\tau_C$ .

**5.1 Security assumptions**

**Timelocks.** Security of this protocol follows from setting the timelocks to be

$$tw_A + 2\varrho < \tau_C \quad \tau_C + 2\varrho < \tau_A \quad \tau_B = \max(tw_B, \tau_A + 2\varrho) \quad (2)$$

where  $\varrho$  is the time required for a transaction be reliably confirmed on the blockchain. There is no enforced relationship between the escrow expiry times ( $tw_A, tw_B$ ). Escrows can be paired regardless of expiry time.

**Closing escrows in a timely manner.** As in Section 4.1 the user must close her exchange escrow before it expires at time  $tw_B$ . Similarly, to withstand attacks by a malicious user, the exchange must close its user escrow before it expires at time  $tw_A$ . However in limit orders, the user must now come online between  $tw_A$  and  $\tau_C$  to either post her user escrow refund or if a malicious exchange has posted a canceled puzzle output the user must then use the cancel value  $c$  to

claim the coins from that output. Finally, the time period in which the user can unilaterally recover coins from frozen escrows is  $(tw_{\mathcal{A}}, \tau_{\mathcal{B}})$ .

Prior to opening a limit order on an escrow pair, the user and the exchange must cancel any currently open limit orders on that escrow pair using the *Cancel Limit Order* procedure.

## 5.2 All-or-None (AoN) Limit Orders

This protocol allows the user to place a limit order for a specified amount and price against a (user escrow, exchange escrow) pair. The order remains open until the limit price is met for the entire amount. Once the limit price is met, the exchange executes the order.

**Limit Order.** To place the limit order, the user specifies the amount and the limit price *e.g.*, “I will sell 3.1 BTC at the price of 1 BTC per 100 LTC”. To place the limit order, the user and exchange perform the “Request”, “Quote” and “Order” steps of the RFQ protocol in Section 4.2 for the price that the user requested. The exchange now has the ability to execute or fill the limit order by posting the user escrow solve and puzzle transactions thereby releasing  $x$ .

**Execute Limit Order.** To execute the order, the exchange performs the “Execute” step of the unidirectional RFQ protocol in Section 4.2. This fills the order at the limit price for the specified amount.

**Cancel Limit Order.** The user can cancel her order at any time after placing it and prior to it being filled. She can’t force the exchange to participate in the cancel protocol, but if the exchange does complete the protocol, even a malicious exchange can’t execute the order. To do this the user requests the order be canceled. In reply the exchange releases the cancel value  $c$  for the user escrow puzzle transaction used to place the limit order. This cancels the limit order since if the exchange misbehaves and posts the canceled puzzle transaction the user can reclaim the coins the exchange would be buying in the trade.

## 5.3 Partial-fill Limit Orders

We now show how to use our All-or-None Limit Order Protocol from Section 5.2 to construct a partial-fill limit order *i.e.*, an order that can be incrementally filled/executed at the limit price. Partial fill limit orders are important for trading as they are the default order type supported by all centralized exchanges. In fact, partial-fill limit orders are so basic that the term *limit orders* typically refers to partial-fill limit orders. Our partial-fill limit order is composed of  $N$  All-or-None limit orders (Section 5.2), which we call *sub-orders*. By selectively executing some of these sub-orders and not-executing others, the exchange is able to control how much of the limit order fills.

Our partial fill limit order will use puzzle transactions with  $N$  puzzle outputs rather than a single puzzle output as done in our other protocols. These  $N$  puzzle outputs  $\text{Out}_1, \dots, \text{Out}_N$  place  $N$  different All-or-None limit *sub-orders*. We denote the amount of coin the  $i$ -th sub-order locks in  $\text{out}_i$  as  $a_i$ . The amounts  $a_1, \dots, a_N$

locked in the  $N$  outputs are chosen such that each amount decreases by one half from the previous amount,  $a_i = \frac{1}{2} \times a_{i+1}$  and that they sum to the total amount  $A = \sum_{i=1}^N a_i$  which the user is selling in partial-fill limit order. Thus, for any  $N$  and  $A$  we determine the amount  $a_i$  to lock in a puzzle output  $out_i$  as

$$a_i = \frac{A(2^{N-i})}{(2^N - 1)} \quad (3)$$

Using this sub-orders the exchange can execute as limit order trade for any amount between 0 to  $A$  in increments of  $a_N = A/(2^N - 1)$ .

Lets look at the example in Figure 3(b), Alice placed a limit order selling  $A = 3.1$  BTC for  $A = 310$  LTC. Thus if we set  $N = 5$  Alice's user escrow puzzle output amounts would be  $a_1 = 1.6, a_2 = 0.8, a_3 = 0.4, a_4 = 0.2, a_5 = 0.1$  (BTC) and using the price she set her exchange escrow puzzle output amounts are  $a_1 = 160, a_2 = 80, a_3 = 40, a_4 = 20, a_5 = 10$  (LTC). By selectively executing only the All-or-None sub-orders in  $Out_2, Out_4, Out_5$  the exchange fills the order so that Alice sells  $0.8 + 0.2 + 0.1 = 1.1$  BTC and buys  $80 + 20 + 10 = 110$  LTC.

Once a user opens a partial-fill limit order it stays open until (a). the user cancels it, (b). it fills completely, or (c). one of the parties unilaterally closes the user or exchange escrows. To determine how much of her limit order has filled the user runs the *Update Limit Order* protocol with the exchange. The exchange can unilaterally fill the limit order even if the user is offline.

**Limit Order.** To place the limit order, the user specifies the amount  $A$  and the limit price *e.g.*, "I will sell 3.1 BTC at price of 1 BTC per 100 LTC". the user and exchange then perform the Limit Order step of our all-or-nothing protocol  $N$ -times. Creating one puzzle transaction per escrow, with each puzzle transaction having  $N$  puzzle outputs. Since the exchange knows the solutions  $x_0 \dots x_N$  the exchange can release a subset of these puzzles to fill the order by amount it fills on the exchange's order book.

**Update Limit Order.** If the user is online, she can query the exchange to learn how much of the limit order she placed has been filled. To do this, the exchange signs and sends the user a new exchange escrow puzzle transaction reflecting the balance of the coins which have been bought and sold as part of the fill. This new exchange escrow puzzle transaction contains a new set of puzzle outputs holding the smaller yet to be filled remainder of the order. In reply, the user signs and sends the exchange a new user escrow puzzle transaction with puzzle outputs mirroring those in the new exchange escrow puzzle transaction. The exchange then releases all the cancel values  $c_1, \dots c_N$  for the previous user escrow puzzle transaction. If the order filled completely, then the order is moved to closed, and cashout transactions are used in place of puzzle transactions.

**Cancel Limit Order.** The user can ask the exchange to cancel her order at any time after she places the order. This is exactly like our Update Limit Order but both parties exchange cashouts rather than puzzle transactions.

## 5.4 Closing Limit Orders

We will describe the process for closing escrows whose last trade was a partial-fill limit order. All-or-None limit orders can be treated as a specific case of the partial-fill protocol where  $N = 1$ . The limit order cooperative close is the same as used by our RFQs protocol in Section 4.4.

Our unilateral close is very similar to the unilateral close and aborts given in unidirectional RFQ protocol given in Section 4.5. However the addition of a cancel on the user escrow puzzle transaction places new requirements on the user and the exchange. The magic of unidirectionality (Section 4.3) protects both the user and the exchange from the other party posting old cashout transactions.

To unilaterally close an exchange escrow the user Alice posts the latest exchange escrow cashout or puzzle transaction. She must come online after  $tw_{\mathcal{A}}$  and before  $\tau_{\mathcal{C}}$  to check if the exchange has unilaterally closed the associated user escrow. If the user escrow has not been spent she signs and posts the refund transaction and is done. If on the other hand it has been spent there are three cases. The user escrow was spent with: (1). the most recent cashout transaction in which case the user is done, (2). a canceled puzzle transaction in which case she claims the coins in the puzzle outputs, or (3). the latest puzzle transaction in which case she then waits until  $\tau_{\mathcal{A}}$  after which she claims the unspent puzzle outputs with a refund transaction and uses the solutions in the spent puzzle outputs to claim her coins from the exchange escrow.

The exchange must come online before  $tw_{\mathcal{A}}$  to post the latest user escrow cashout or puzzle transaction. If the exchange posted a puzzle transaction it must wait until  $\tau_{\mathcal{C}}$  to spend the puzzle outputs by posting a solve transaction containing some of the  $x_1, \dots, x_N$  solutions reflecting the how much of the limit order filled. After  $\tau_{\mathcal{B}}$  the exchange must come online and may post an escrow refund transaction or a puzzle refund transaction refunding the unsolved and unspent puzzle outputs closing the exchange escrow.

## 6 Related work

**Atomic swap protocols.** The first description of an atomic swap is commonly attributed to TierNolan’s 2013 forum post [43]. Many works have since explored atomic swaps [37,35,26,4], including cross-chain auctions [38], improved fungability [27,20], trading across blockchains [6,5] and forks [29] or between tokens on Ethereum’s blockchain [36]. An alternative approach to cross-chain atomic swaps is the trustless issuance of pegged tokens [47,33].

**Layer-two or Off-blockchain protocols.** A layer-two blockchain protocol [32] binds off-blockchain transfers of funds to an on-blockchain smart contract. Typically they do not require the addition of a trusted third party, trusted oracle, or trusted gateway. There has been a variety of work on layer-two protocols for Bitcoin [41,37,13,20,35,26], where transfers of funds are accomplished via atomic swaps. In 2013, Spilman’s unidirectional payment channel was the first

to use the “magic of unidirectionality” that Arwen uses in Section 4.3. Meanwhile, bidirectional payment channels for Bitcoin payments were first proposed by [13,37], and significant progress has been made on the Lightning Network [3]. Today’s Lightning Network requires SegWit, and thus only supports Bitcoin and Litecoin, while Arwen does not require SegWit (See Appendix C) and thus supports more Bitcoin-derived coins, including BCH, ZEC. [35,26] build layer-two protocols “scriptlessly”, without smart contracts, by cleverly leveraging digital signatures. BOLT [18] is a layer two payments protocol with very strong privacy guarantees designed for Zcash. Sparkswap [5] is a peer-to-peer trading platform for BTC and LTC built on top of Lightning. Bitcoin covenants [31] proposes a change to Bitcoin allowing coins to carry scripts even after they are spent.

Smart contracts on Ethereum are Turing-complete, and thus support a dramatically richer set of operations than smart contracts written in Bitcoin Script. Thus, it is no surprise that Ethereum supports layer two protocols including “state channels” [4,28]. Plasma [36] is a proposal for a layer-two decentralized exchange protocol on Ethereum. Similar to Plasma is NOCUST [24] which uses zkSNARKs to ensure correctness of state updates and employs collateral-based protocols for faster transaction finality. Truebit is a fascinating approach, where computations (rather than payments) are moved off the Ethereum blockchain via a layer-two protocol [42]. Generally speaking [28,4,36,24,42] are for Ethereum and ERC-20s only, and so they leverage the richness of Ethereum smart contracts. Meanwhile, Arwen’s Ethereum leg is designed to be functionally equivalent to Arwen’s Bitcoin leg, and so it very strictly mimics the UTXO model used in Bitcoin scripts.

**Fees.** Payment focused protocols typically structure incentives around transaction fees, *i.e.*, fees earned when payments are made. This does not solve the problem of lockup griefing because no fees are earned if no payments are made. Arwen addresses this via escrow fees and reputation. Komodo [6] also aims to solve the lockup griefing problem for on-blockchain atomic swaps using fees. Also the peer-to-peer nature of Komodo means that Bob has a strong incentive to walk away after Alice pays her fee; by contrast Arwen escrow fees are sent from user to exchange, and the exchange’s reputation is at stake if it walks away with the fee without establishing an escrow.

## 7 Conclusion

Arwen is a layer-two blockchain trading protocol allowing traders to benefit from liquidity at centralized exchanges without trusting exchanges with custody of their coins. Instead, Arwen trades are backed by on-blockchain escrows, and executed via fast off-blockchain atomic swaps. Arwen’s RFQ protocol has been implemented and is currently deployed offering secure RFQ trades. Arwen solves many of the incentive issues that emerge when payment protocols are repurposed for cryptocurrency trading. Arwen supports a wide range of coins including Bitcoin, “Bitcoin fork” coins (BCH, LTC *etc.*), and Ethereum.

## 8 Acknowledgements

The authors would like to thank Underscore VC, Digital Garage, Notation, United Bitcoiners, Highland Capital Partners, and the Cybersecurity Factory for their support of Arwen. We also acknowledge Patrick McCorry, Ben Jones, David Vorick and our anonymous reviewers for their valuable feedback.

## References

1. Etherdelta. <https://etherdelta.com/>.
2. Kucoin: download arwen to trade on kucoin directly from your wallet. <https://www.kucoin.com/page/arwen>.
3. Lightning daemon (lnd). <https://github.com/lightningnetwork/lnd>.
4. Raiden network. <https://raiden.network/>.
5. Sparkswap. <https://sparkswap.com/>.
6. Komodo. *White paper*, June 3 2018.
7. Clare Baldwin. Bitcoin worth \$72 million stolen from bitfinex exchange in hong kong. *Reuters*, August 3 2016.
8. Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *International Conference on Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.
9. Bloomberg. Coincheck Hack: How to Steal \$500 Million in Cryptocurrency. *Fortune*, January 31 2018.
10. Mark Friedenbach BtcDrak and Eric Lombrozo. BIP 0112: CHECKSEQUENCEVERIFY.
11. Vitalik Buterin. Bitfloor Hacked, \$250,000 Missing. *Bitcoin Magazine*, September 5 2012.
12. Vitalik Buterin. Eip 1014: skinny create2. *Ethereum Improvement Proposals*, 2018.
13. Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.
14. Ian DeMartino. Mintpal hacked ‘considerable amount’ of vericoins stolen. *Cointelegraph*, Jul 13 2014.
15. Johnson Lau Eric Lombrozo and Pieter Wuille. BIP 141: Segregated Witness.
16. Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. 2019.
17. Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.
18. Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.
19. Ethan Heilman. Moving Arwen’s Ethereum smart contract to create2, 2019.
20. Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.
21. Stan Higgins. Details of 5 Million Bitstamp Hack Revealed. *CoinDesk*, July 1 2015.

22. Stan Higgins. Cryptsy threatens bankruptcy, claims millions lost in bitcoin heist. *Coindesk*, Jan 15 2016.
23. Stan Higgins. Bitcoin exchange youbit to declare bankruptcy after hack. *Coindesk*, Dec 19 2017.
24. Rami Khalil and Arthur Gervais. NOCUST-A Non-Custodial 2nd-Layer Financial Intermediary. *IACR Cryptology ePrint Archive*, 2018.
25. Kraken. How long do digital assets/cryptocurrency deposits take?
26. Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *Network and Distributed System Security Symposium (NDSS)*, 2019.
27. Gregory Maxwell. CoinSwap: Transaction graph disjoint trustless trading. *Bitcoin-talk*, 2013.
28. Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018.
29. Patrick McCorry, Ethan Heilman, and Andrew Miller. Atomically trading with roger: Gambling on the success of a hardfork. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 334–353. Springer, 2017.
30. David Z. Morris. BitGrail Cryptocurrency Exchange Claims \$195 Million Lost to Hackers. *Coindesk*, February 11 2018.
31. Malte Möser, Ittay Eyal, and Emin Gün Sirer. Bitcoin covenants. In *International Conference on Financial Cryptography and Data Security*. Springer, 2016.
32. Neha Narula. The Importance of Layer 2, May 27 2018.
33. Keep Network. tBTC A Decentralized Redeemable BTC-backed ERC-20 Token, 2019.
34. Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, TTUD-BS-1999-02, Darmstadt University of Technology, 1999.
35. Andrew Poelstra. Scriptless Scripts. 2018.
36. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. 2017.
37. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. 2016.
38. James Prestwich. Summa.One: Cross-chain Auctions via Bitcoin Double Spends, 2018.
39. Pete Rizzo. Poloniex Loses 12.3% of its Bitcoins in Latest Bitcoin Exchange Hack. *Coindesk*, March 5 2014.
40. Jon Russell. Korean crypto exchange coinrail loses over \$40m in tokens following a hack. *Techcrunch*, June 10 2018.
41. Jeremy Spilman. [bitcoin-development] anti dos for tx replacement, April 20 2013.
42. Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. 2017.
43. TierNolan. Re: Alt chains and atomic transfers. *Bitcoin-talk*, May 21 2013.
44. Peter Todd. BIP 0065: OP\_CHECKLOCKTIMEVERIFY.
45. Will Warren. Front-running, Griefing and the Perils of Virtual Settlement, 2017.
46. Wikipedia. Mt. Gox.
47. Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. *IEEE Security and Privacy. IEEE*, 2019.
48. Wolfie Zhao. Crypto exchange zaif hacked in \$60 million bitcoin theft. *Coindesk*, Sept 20 2018.

## A Bidirectional RFQs

The following Arwen RFQ protocol is *bidirectional* allowing Alice to buy and sell coins from her user escrow, and to buy and sell coins from her exchange escrow. A bidirectional protocol is useful for high-frequency trading strategies, where the trader quickly moves coins back and forth. Figure 4 shows the transaction diagram for user and exchange escrows.

Like the unidirectional protocol of Section 4, each off-blockchain RFQ trade in the bidirectional protocol is backed by a user escrow (with expiry time  $tw_{\mathcal{A}}$ ) and an exchange escrow (with expiry time  $tw_{\mathcal{B}}$ ). The protocol for opening these escrows remains identical to the one in Section 3.1. Both escrows must be open during a trade (not expired, frozen, or closed). We first overview the necessary technical tools, describe the trading protocol, and explain how escrows can be securely closed when one party becomes malicious or uncooperative.

### A.1 Technical tools

We continue to execute trades using HTLCs, where a trade is executed by revealing the solution  $x$  to a puzzle  $y$ , where  $y = H(x)$ . However, but there are several other technical tools needed in order to make this protocol bidirectional.

**Four puzzles transactions.** In the unidirectional protocol, there was only a single type of puzzle transaction that could spend the output of each escrow. In the bidirectional protocol, there are four puzzle transactions per escrow: `payU_postU`, `payU_postE`, `payE_postU`, and `payE_postE`. This is true for both the user escrow and the exchange escrow. The only difference between the puzzles for the user escrow puzzles and the puzzles for the exchange escrow is that user escrow puzzles have  $tw = tw_{\mathcal{A}}$  and exchange escrow puzzles have  $tw = tw_{\mathcal{B}}$ . The user will only post a puzzle transaction if the exchange aborts a trade. The exchange will post a puzzle transaction if the user aborts a trade or if the user fails to cooperatively close an escrow before it expires.

*payE and payU puzzles.* By having both `payE` and `payU` puzzles for each escrow, each escrow can be used to both buy and sell coins. In a `payE` puzzle, the puzzle locks coins that pay to the exchange once the exchange reveals  $x$  in a solve transaction, before time  $\tau_{\mathcal{A}}$ . The `payE` puzzle transaction has an HTLC smart contract that is similar to the one in Figure 4.3 of our unidirectional protocol, using timelock  $\tau_{\mathcal{A}}$  and puzzle  $y$ . In a `payU` puzzle, the puzzle locks coins that pay to the user, once the user reveals  $x$  in a solve transaction before time  $\tau_{\mathcal{B}}$ . The `payU` puzzle has an HTLC smart contract that is similar to the one in Figure 2(b) of our unidirectional protocol using timelock  $\tau_{\mathcal{B}}$  and puzzle  $y$ .

If the user is doing a trade that buys coins from the user escrow while selling coins from the exchange escrow, we would use a `payU` puzzle that spends the user escrow and a `payE` puzzle that spends the exchange escrow. Meanwhile, to buy coins from the exchange escrow while selling coins from the user escrow, we use a `payE` puzzle that spends the user escrow, and `payU` puzzle that spends the exchange escrow.

*postU and postE puzzles.* A postU puzzle transaction can only be posted to the blockchain by the user. The postU puzzle transaction is initially formed and signed by the exchange, and then sent to user; the user can later post this transaction to the blockchain if the exchange becomes uncooperative. The analogous postE puzzle transaction can only be posted by the exchange

**Two cashout transactions.** The postU cashout is signed by the exchange and sent to the user, and is used to unilaterally close an escrow when all trades against the escrow completed successfully. The postE cashout, which is signed by the user and sent to the exchange, is used to cancel RFQ quotes.

**Cancelling transactions.** Arwen’s bidirectional RFQ protocol no longer uses “the magic of unidirectionality” (Section 4.3). Instead, we “overwrite” transactions from old trades by cancelling them, adapting the idea of *breach remedy transaction* from the Lightning Network [37].

Every postE-type transaction, which is first signed by the user and then sent to the exchange, can be cancelled by the exchange using the *cancel value*  $C_{\mathcal{E}}$ . The cancel value  $C_{\mathcal{E}}$  is randomly chosen and kept secret by the exchange. To cancel the postE transaction, the exchange reveals  $C_{\mathcal{E}}$  to the user. The cancel value  $C_{\mathcal{E}}$  protects the user if the exchange posts a canceled postE transaction to the blockchain, as follows. The postE transaction contains a value  $j_{\mathcal{E}}$  such that  $j_{\mathcal{E}} = H(C_{\mathcal{E}})$ , and every output on the postE transaction that pays out to the exchange also includes the following smart contract:

“Coins may only be paid to the exchange after time  $tw$ , OR the coins return to the user if the user reveals a cancel value corresponding to  $j_{\mathcal{E}}$ .”

Then, if the exchange misbehaves by posting a cancelled transaction, the user has can retaliate via a justice transaction anytime before  $tw$ . The justice transaction reveals  $C_{\mathcal{E}}$  and is posted unilaterally by the user, allowing her to claim *all* the coins in the canceled transaction.

**Previous transactions.** The following notation is useful for our protocol description and analysis. Let  $\text{pay*\_postE}$  be a puzzle transaction from the current trade (where  $*$  is either U or E). For convenience, we use the notation  $\text{prev}(\text{pay*\_postE})$  to indicate the transaction from a previous trade that is (a) held by the exchange and (b) spends the same escrow as the  $\text{pay*\_postE}$ .

**Differences from the Lightning Network.** Our bidirectional protocol has many things in common with the payment channel design of the lightning network. However, there are some important differences. To support coins that don’t have a transaction malleability fix, *e.g.*, SegWit, additional constraints are placed on our design. Unlike lightning we can not use the relative timelocking mechanism `CheckSequenceVerify` [10], instead we can only use the absolute timelocking mechanism `CheckLockTimeVerify` [44]. Thus, our escrows always have a fixed time before they must be closed. Additionally, as discussed in Section C, due to the threat of transaction malleability any transactions which require signatures from both parties must spend from a transaction which is already confirmed on-blockchain. This requirement results in a slightly different breach remedy mechanism. Unrelated to malleability, our protocol is focused on trading

cross-chain at centralized exchanges, rather than on peer-to-peer payments, so we have the escrow fee mechanism to enable traders to open escrows when they do not hold any of the coins they are buying (see Section 3.2).

## A.2 Security assumptions.

**Timelocks.** The security of this protocol requires we set the timelocks to be

$$\tau_A > \max(tw_A, tw_B) + 2\varrho \quad \tau_B > \tau_A + 2\varrho \quad (4)$$

where  $\varrho$  is the time required for transaction be reliably confirmed on a blockchain. Notice that there is no relationship between the escrow expiry times  $(tw_A, tw_B)$ , allowing to us pair any user escrow and exchange escrow, regardless of expiry.

**Closing escrows in a timely manner.** To withstand attacks by a malicious user the exchange must close its user escrows before they expire at time  $tw_A$ , and its exchange escrows before they expire at  $tw_B$ . To withstand attacks by a malicious exchange the user must close her user escrows before they expire at time  $tw_A$ , and her exchange escrows before they expire at  $tw_B$ . If the user forgets to do this, an honest exchange will close the escrow on the user’s behalf, but a malicious exchange may be able to steal coins from the escrow.

## A.3 Off-blockchain RFQ trades.

**Setup.** The exchange chooses random solution  $x$ , computes  $y = H(x)$ , and sends the puzzle  $y$  to the user. The exchange also chooses a two random cancel values  $C_{\mathcal{E},A}$  and  $C_{\mathcal{E},B}$ , computes  $j_{\mathcal{E},A} = H(C_{\mathcal{E},A})$  and  $j_{\mathcal{E},B} = H(C_{\mathcal{E},B})$ , and sends  $j_{\mathcal{E},A}$  and  $j_{\mathcal{E},B}$  to the user. The exchange keeps  $x$ ,  $C_{\mathcal{E},A}$ , and  $C_{\mathcal{E},B}$  secret. The user responds by choosing a random secret cancel value  $C_U$ , computing  $j_U = H(C_U)$ , and sending  $j_U$  to the exchange.

**Request.** Alice requests a quote, indicating what escrow she wants to sell coins from, and what escrow she wants to buy coins from, and the amount of coins she wants to buy<sup>2</sup>. If Alice is selling from the user escrow, this protocol uses payE puzzles on the user escrow and payU puzzles on the exchange escrow. If Alice is buying from the user escrow, we use payU puzzles on the user escrow and payE puzzles on the exchange escrow.

Alice then sends the exchange a payU\_postE puzzle transaction that (a) locks the amount of coins she is buying under puzzle  $y$ . If Alice is buying coins from the user escrow, this payU\_postE puzzle transaction (b) spends the output of the user escrow and (b) can be cancelled under  $C_{\mathcal{E},A}$ . If Alice is buying coins from the exchange escrow, this payU\_postE puzzle transaction (b) spends the output of the exchange escrow and (b) can be cancelled under  $C_{\mathcal{E},B}$ .

<sup>2</sup> For a sell-side RFQ, we could prefix the flow described here with two additional messages: (1) the user indicates an amount of coin she wishes to sell and requests a quote, and (2) the exchange provides the user with the quote with the amount she can buy.

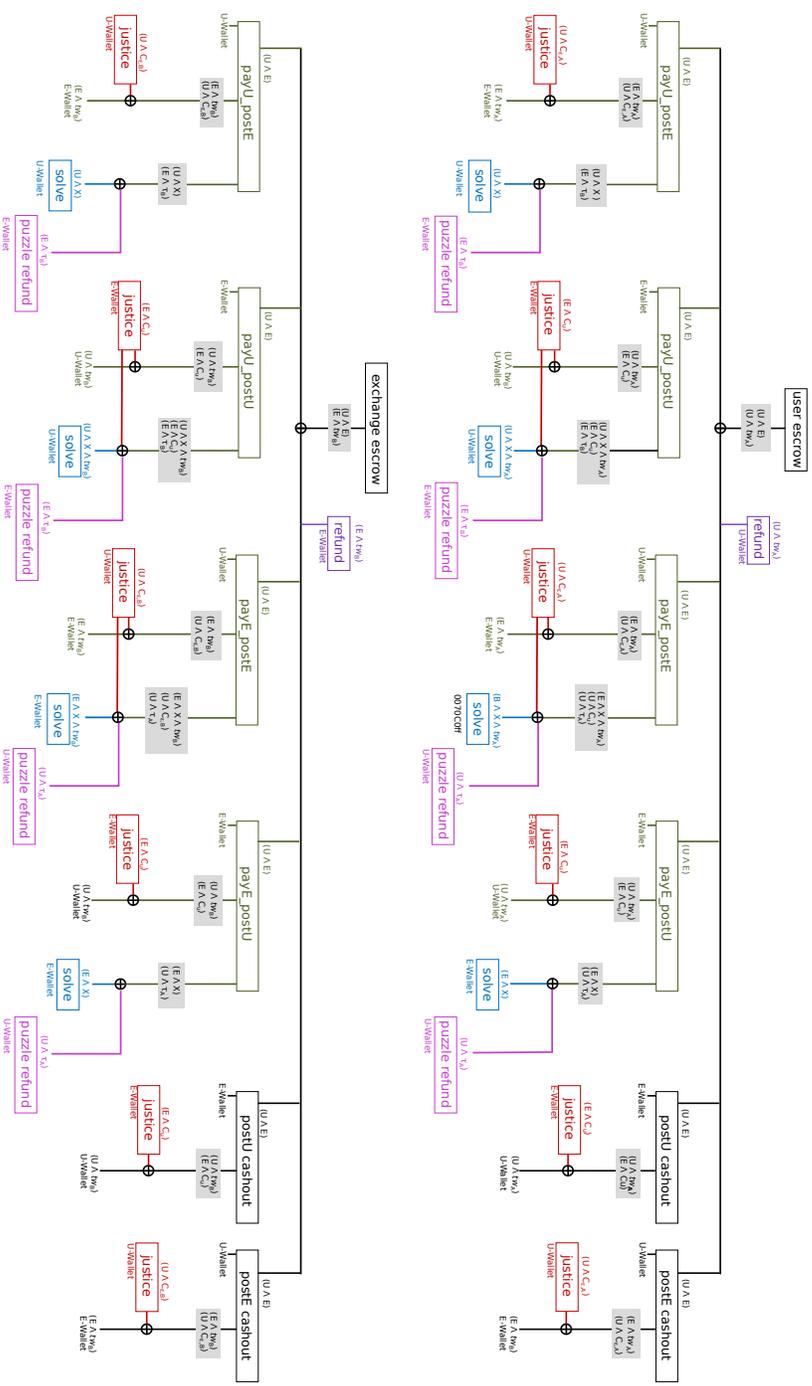


Fig. 4. Transaction diagram for bidirectional RFQ and bidirectional limit order protocols.

As an example, refer again to Figure 1, and suppose after the second trade in the Figure Alice requests a quote “Buy 2 BTC for some LTC.” In this case, Alice would send the exchange a `payU_postE` puzzle transaction that (a) locks 2 BTC under puzzle  $y$ , (b) spends the output of the user escrow, and (c) can be cancelled using cancel value  $C_{\mathcal{E},\mathcal{A}}$ .

**Quote.** The exchange responds with the quote—“2 BTC can be bought for 200 LTC, open for time  $\delta$ ”. To commit to the quote, the exchange signs and sends Alice the following three items.

1. A `payU_postU` puzzle transaction that (a) locks the amount of coins Alice is buying under puzzle  $y$ , and (b) can be cancelled using cancel value  $C_U$ . If Alice is buying coins from the user escrow, this `payU_postU` puzzle transaction (c) spends the output of the user escrow. Otherwise, this `payU_postE` puzzle transaction (c) spends the output of the exchange escrow. (This puzzle is analogous to the puzzle sent to Alice during the quote phase of the unidirectional protocol in Section 4.2.)
2. A `payE_postU` puzzle transaction that (a) locks the amount of coins Alice is selling under puzzle  $y$ , and (b) can be cancelled using cancel value  $C_U$ . If Alice is selling coins from the user escrow, this `payU_postU` puzzle transaction (c) spends the output of the user escrow. Otherwise, this `payU_postE` puzzle transaction (c) spends the output of the exchange escrow.
3. If Alice buying from user escrow, the cancel value  $C'_{\mathcal{E},\mathcal{A}}$  from the previous trade. Otherwise, the cancel value  $C'_{\mathcal{E},\mathcal{B}}$  from the previous trade. (This cancels `prev(payU_postE)`, so that the current `payU_postE` puzzle is the only uncanceled puzzle transaction held by the exchange for the escrow spent by this puzzle.)

At this point, the user can either place the order, or cancel the quote.

**Order** To place the order, the user sends the exchange the following two items.

1. A `payE_postE` puzzle that (a) locks the amount of coins Alice is selling under puzzle  $y$ . If Alice is selling coins from the user escrow, this `payE_postE` puzzle transaction (b) spends the output of the user escrow and (c) can be cancelled under  $C_{\mathcal{E},\mathcal{A}}$ . Otherwise, this `payE_postE` puzzle transaction (b) spends the output of the exchange escrow and (c) can be cancelled under  $C_{\mathcal{E},\mathcal{B}}$ . (This is analogous to the puzzle sent to the exchange during the Order phase in Section 4.2.)
2. The cancel value  $C'_{\mathcal{E}}$  from the previous trade. (This cancels `prev(payU_postU)` and `prev(payE_postU)`.)

Once the user places the order, she cannot back out of the order. This is because (1) the only uncanceled puzzles Alice holds are the current `payU_postU` and `payE_postU` puzzles for this trade, and (2) the exchange knows the solution  $x$  and thus has the ability to unilaterally claim the coins the user is selling in this trade by (a) posting the `payE_postE` transaction to the blockchain, and then (b) revealing  $x$  in a solve transaction.

**Execute.** Once the order is placed, the exchange sends the user the following four items. This execute phase is not performed if the user cancels the quote.

1. If Alice selling from the user escrow, the cancel value  $C'_{\mathcal{E},\mathcal{A}}$  from the previous trade. Otherwise, the cancel value  $C'_{\mathcal{E},\mathcal{B}}$  from the previous trade. (This cancels  $\text{prev}(\text{payE\_postE})$ . At this point in the protocol, the only uncanceled puzzle transactions held by the exchange are the current  $\text{payE\_postE}$  and the current  $\text{payU\_postE}$  puzzle transactions.)
2. The solution  $x$  for the current trade.
3. Two  $\text{postU}$  cashout transactions, one for the user escrow and one for the exchange escrow. Both of the transactions reflect the balance in the two escrows after this trade. (This last step is done because the solution to the  $\text{payU\_postU}$  puzzle transaction may only be posted by the user after time  $tw$ ; to avoid requiring the user to come online at time  $tw$  to post the solution, we instead have the exchange release a cashout transaction.)

**Cancel Quote.** If the user does not want to place an order, then the user sends the following after receiving the Quote message.

1. A  $\text{postE}$  cashout that (a) resets the balance of the escrow as it was before the aborted trade. If Alice is buying coins from the user escrow, the  $\text{postE}$  cashout (b) spends the output of the user escrow and (b) can be cancelled under  $C_{\mathcal{E},\mathcal{A}}$ . Otherwise, the  $\text{postE}$  cashout (b) spends the output of the exchange escrow and (c) can be cancelled under  $C_{\mathcal{E},\mathcal{B}}$ . (This cashout replaces the  $\text{payU\_postE}$  transaction that Alice sent to the exchange as part of the Request message in the current trade.)
2. The cancel value  $C_{\mathcal{U}}$  from the current trade. (This cancels both the current  $\text{payU\_postU}$  and the current  $\text{payE\_postU}$  puzzles. However,  $\text{prev}(\text{payU\_postU})$  and  $\text{prev}(\text{payE\_postU})$  remain valid.)

#### A.4 Closing an escrow.

The process for cooperatively closing these escrow is identical to that of unidirectional RFQ protocol of Section 4.4. We now sketch how each party can unilaterally close escrows, assuming that all trades against these escrows completed successfully. Recall that these procedures are only required if one party becomes malicious or unresponsive.

**Unilateral close for the user.** The user must remember to close the both escrows before they expire at time  $tw$ . If the outputs of both the user escrow and the exchange escrow are unspent, then the user can unilaterally close both the escrows by posting the most recent  $\text{postU}$  cashout transaction for that escrow.

Suppose that upon attempting to close an escrow, the user sees that exchange has posted a cancelled  $\text{postE}$  transaction spending the output of that escrow. In this case, the user can immediately use the cancel value to post a justice transaction that claims, for the user, all coins in the cancelled transaction. This

is possible because the user is expected to close the escrow before it expires at time  $tw$  AND all coins in a `postE` transaction that pay to the exchange are locked until time  $tw$ . This is why it is never in the interest of a party to post a cancelled transaction!

Next, suppose that upon attempting to close an escrow, the user sees that exchange has posted the `payE_postE` puzzle transaction from the current trade, but the output of the other escrow is unspent. In this case, the user immediately posts the `postU` cashout transaction to unilaterally close the other escrow. No further action is required from the user; the exchange can recover the coins locked in the current `postE_payE` puzzle transaction by posting the solve transaction after time  $tw$ .

Finally, suppose that upon attempting to close an escrow, Alice sees that exchange has posted the `payU_postE` puzzle transaction from the current trade. Both escrows must be frozen. If the output of the other escrow is unspent, then the user immediately posts the current `payE_postU` puzzle for that escrow. The user then comes online between time  $\tau_A$  and  $\tau_B$  to recover the coins from the current trade, using a procedure similar to that used to recover from frozen escrows in Section 4.5. The same is done if the other escrow is spent with a `payU_postU` transaction. If the other escrow is spent using any other transaction, then no further action is required from the user.

*Unilateral close after a Cancelled Quote* What happens when Alice must unilaterally close an escrow after a Cancel Quote? This case is very similar to the case described in Section A.5. Before  $tw$ , the user must post the `payU_postU` puzzle from the Cancelled-Quote trade. The exchange must then come online after time  $\tau_B$  for the current trade, in order to claim the coins locked in the puzzle by posting a puzzle-refund transaction. That closes one of the escrows. To close the other escrow, the user can post the `postU` cashout transaction from previous completed trade before time  $tw$ .

**Unilateral close for the exchange.** The exchange must also remember to close the both escrows before they expire at time  $tw$ .

If either output is spent using a cancelled transaction, the exchange immediately posts a justice transaction that claims all the coins in the escrow.

If the outputs of both escrows are unspent, the exchange can unilaterally close each escrow by posting the most recent `payE_postE` and `payU_postE` puzzle transactions. If one escrow is already spent using a current `payE_postU` puzzle transaction, the exchange immediately posts the current `payU_postE` puzzle for the other escrow (assuming that escrow is unspent). If one escrow is already spent using a current `payU_postU` puzzle transaction, the exchange immediately posts the current `payE_postE` puzzle for the other escrow (assuming that escrow is unspent).

In all of the above cases, the exchange must come online between time  $(tw, \tau_A)$  and post the solve transaction for the `payE` puzzle, releasing the coins locked in the `payE` puzzle to the exchange's wallet. The exchange can unilaterally release the coins locked in the `payU` puzzle by posting a puzzle-refund

transaction after time  $\tau_B$ . An honest exchange would send these coins to the user's wallet (because they rightfully belong to the user), but a malicious exchange would claim these coins for itself. This is why a user must remember to close her escrows before they expire (at time  $tw < \tau_B$ )!

### A.5 Dealing with an aborted trade.

What happens when a trade aborts? A trade can abort after the Request, after the Quote, or after the Order. If the exchange elects not to provide the user with a Quote, nothing happens and the user can keep trading. If the user elects not to place an Order and also refuses to send a Cancel Order message, the exchange must stop trading and close the escrows. Finally, if the exchange aborts after the Order is placed, the user must stop trading, freeze and then close the escrows.

**Exchange aborts after Request** Suppose the exchange elects not to provide a Quote after a Request message is sent. This is not a problem, because the exchange will not want to post the current `payU_postE` received during the Request. This follows because the current `payU_postE` transfers coins to from the exchange to Alice (*i.e.*, it is a `payU` puzzle), and thus will result in more coins for the user and fewer coins for the exchange. Instead, the exchange will always prefer to use the puzzles from the previous completed trade.

**User aborts after Quote** Suppose Alice decides to abort after receiving a Quote, while refusing to send the Cancel Order message. In this case, the exchange should immediately close the escrows involved in this trade, as follows.

First, the exchange posts the `payU_postE` puzzle from the current trade. The exchange must then come online after time  $\tau_B$  for the current trade, in order to claim the coins posted in the current `payU_postE` puzzle by posting a puzzle-refund transaction. (If the exchange cannot post the `payU_postE` puzzle because its escrow is already spent, it follows that the user must have either (a) posted a cancelled transaction, (in which case the exchange can reclaim its coins through a justice transaction) or (b) posted a `payU_postU` puzzle (in which case the exchange's again posts the puzzle-refund after time  $\tau_B$ ). That closes one of the escrows.

What about the other escrow? There are two cases:

*Case 1:* The exchange holds an uncancelled `payE_postE` puzzles from the previous trade that spends this escrow. The exchange must post this `payE_postE` puzzle before time  $tw$ . The exchange must then must come online between time  $tw$  and  $\tau_A$  for the previous trade, and claim the locked coins by posting a solve transaction.

*Case 2:* The exchange holds an uncancelled `payU_postE` puzzles from the previous trade that spends this escrow. The exchange must post the `payU_postE` puzzle from the previous trade before time  $tw$ . Alice must then must come online between time  $tw$  and  $\tau_B$  for the previous trade, in order to claim the coins in

the puzzle transaction by posting a solve transaction. This is possible because the exchange has revealed the solution  $x'$  to Alice as part of the previous trade. Importantly, Alice will always know that she is supposed to take this action, because she is expected to close this escrow before time  $tw$  using the cashout from the previous trade (see ‘Unilateral close after cancelled quote’ in Section A.4). If Alice finds that she cannot do this because the exchange has already posted the `payU_postE` from the previous trade, then Alice knows she must come online between time  $tw$  and  $\tau_B$ .

Finally, if the exchange cannot close this escrow because its output is already spent, it follows that (a) the user posted a cancelled transaction (in which case the exchange can reclaim its coins through a justice transaction) or (b) posted a cancelled `payE_postU` puzzle (in which case the exchange does as in Case 1), or (c) posted an uncancelled `payU_postU` puzzle (in which case the exchange does as in Case 2), or (d) posted an uncancelled `postU` cashout transaction from the previous trade (in which case the exchange has earned its rightful balance of coins).

**Exchange aborts after Order.** Suppose the exchange decides not to Execute after an Order. This causes the user to freeze the escrows.

After the Order message is sent, there are five uncancelled puzzles: the four puzzles from the current trade, plus one additional puzzle from the previous trade, *i.e.*, the `prev(payE_postE)` puzzle. We argue that the `prev(payE_postE)` puzzle would never be posted. Why? This follows because exchange would always prefer to post the current `payE_postE` puzzle over the previous puzzle. There are two cases. (1) The previous puzzle is a `payE_postE` type puzzle. In this case, it follows that the current `payE_postE` puzzle pays the exchange more coins than then previous puzzle, and so the exchange would prefer to post the current puzzle. (This is the “magic of unidirectionality”, see Section 4.3.) (2) The previous puzzle is a `payU_postE` type puzzle. In this case, the current `payE_postE` puzzle pays out to the exchange, while the previous puzzle pays out to the user. It follows that the current `payE_postE` puzzle pays the exchange more coins than then previous puzzle, and so the exchange would prefer to post the current puzzle.)

Therefore, only the four puzzles from the current trade matter, and we have essentially reduced back to the frozen case from the unidirectional protocol. Thus, if the exchange aborts after the Order, the user would try to cooperatively close her escrows using the balance from the previous trade. If that fails, she would unilaterally post the `payU_postU` puzzle. (This allows Alice to get paid if the exchange decides to execute the trade on-blockchain by revealing the solution  $x$ .) Once that puzzle is confirmed on the blockchain, the user would then post the `payE_postU` puzzle. (This forces the exchange to reveal the solution  $x$  between time  $(tw, \tau_A)$  if the exchange decides it wants to execute the trade on the blockchain.) The user would then come online between time  $\tau_A$  and  $\tau_B$  and use the usual procedure for recovering from frozen escrows.

## B Ethereum Unidirectional RFQs

We now describe how to port the unidirectional RFQ protocol of Section 4 to Ethereum. We use an escrow smart contract that mimics the UTXO transaction paradigm that is used on Bitcoin.

**Ethereum Smart contract.** Each user escrow and exchange escrow is a smart contract that can be in one of three states: (UNFUNDED, OPEN, PUZZLEPOSTED, CLOSED). The payer first computes the contract address and funds the contract address before the contract is created. This technique is called “counterfactual instantiation” by the ethereum community and relies on the contract address being deterministic. Then once the contract address has been funded, the payer posts another transaction which creates the escrow smart contract using the *Create2* contract creation call [12,19]. The escrow smart contract constructor checks if the contract address has been funded and if it has it changes the state from UNFUNDED to open. If on the other hand, the contract address has not been funded the constructor throws an exception and doesn’t create the smart contract.

The user escrow can move from the OPEN state to the CLOSED state via a:

1. *refund transaction* which is signed by the user and posted to the blockchain after time  $tw_A$ , (thus fulfilling the timelock condition of the Arwen escrow)
2. *cashout transaction* which is doubly-signed by the user and by the exchange, (thus fulfilling the 2-of-2 multisig condition of the Arwen escrow)

Meanwhile, the escrow smart contract moves from the OPEN state to the PUZZLEPOSTED state when a *puzzle transaction*, that calls a method in the escrow smart contract, is confirmed on the Ethereum blockchain. The puzzle transaction is doubly-signed by the user’s ephemeral key and the exchange’s ephemeral key (thus fulfilling the 2-of-2 multisig condition of the Arwen escrow) and contains a puzzle  $y$  and an puzzle timelock  $\tau$  (which are used for atomic swap trading). Then, a user escrow can move from the PUZZLEPOSTED state to the CLOSED state via:

3. *solve transaction* which contains solution  $x$  and is signed by the exchange
4. *puzzle-refund transaction* which is signed by the user and posted to the blockchain after time  $\tau_A$

The exchange escrow can be analogously arrive in the CLOSED state in four ways (*i.e.*, via solve, puzzle-refund, refund, or cashout transaction).

The RFQ protocol is essentially identical to that of Section 4. The four ways that the user escrow can be closed are identical to the four ways that an escrow can be closed in the protocol of Section 4 (see also Figure 4.3). As in Section 4, the cashout transaction is used for cooperatively closing an escrow, while the puzzle, solve, refund, and puzzle-refund transactions are used to unilaterally close escrows per Section 4.5.

## C Transaction Malleability

Arwen withstands transaction malleability attacks on “Bitcoin-derived” blockchain that do not have SegWit. We now explain the transaction malleability problem, discuss why it affects layer-two blockchain protocols, and explain how Arwen avoids it. Withstanding transaction malleability allows Arwen to support more Bitcoin-derived blockchains.

**Transaction malleability.** Consider a transaction  $T_2$  that spends the output of a transaction  $T_1$ .  $T_2$  therefore contains a pointer to  $T_1$ , called the TXID. This TXID is malleable: the TXID can be changed (“mauled”) by anyone, without affecting the validity or contents of transaction  $T_1$ .

We explain why as follows. The TXID on  $T_1$  is the hash of (essentially) the entire  $T_1$ , including any signatures on  $T_1$ . Most “Bitcoin derived” blockchains use elliptic curve digital signatures, which are not deterministic. (That is, a random value  $r$  is used to compute the signature  $\sigma$  on message  $m$ .) This means that a party that holds the secret signing key can easily produce multiple valid signatures  $\sigma, \sigma', \dots$  on a single message  $m$ . Worse yet, even a party that does not know the secret signing key can take a valid signature  $\sigma$  on a message  $m$ , and maul  $\sigma$  to obtain a different valid signature  $\sigma'$  on  $m$ . Now, because TXID is the hash of (essentially) the entire  $T_1$ , mauling the signatures on  $T_1$  results in a completely different TXID for  $T_1$ . Additionally, some parts of the transaction that are included in the TXID hash are *not* covered by the signature on the transaction, which creates an additional malleability problem.

**SegWit.** With SegWit [15], the TXID hash is *not* computed over the signatures on the transaction. This solves the malleability problem, because now mauling the signature has no effect on the TXID. SegWit also removes other malleability vectors (*i.e.*, parts of the transaction that are not covered by the signature).

**Impact on layer-two protocols.** If  $T_1$  is already reliably confirmed on the blockchain, the security of the blockchain ensures that no one can maul the signatures on  $T_1$ , and transaction malleability is irrelevant.

Now consider a layer-two protocol where Alice holds  $T_1$ , an off-blockchain transaction that spends an on-blockchain transaction  $T_0$ . Next suppose that Alice transfers coins to Bob by sending Bob an off-blockchain transaction  $T_2$  that is signed by Alice and contains a pointer to  $T_1$ . Transaction malleability means that Alice’s signature on  $T_2$  is completely useless. This follows because Alice can break the TXID pointing from  $T_2$  to  $T_1$  by mauling the signatures on  $T_1$ ; this means that  $T_2$  becomes an invalid transaction but  $T_1$  remains valid. Thus, Bob could not use  $T_2$  to claim coins from  $T_0$ . (This is exact reason why the Lightning Network, as currently designed, only works with blockchains that support SegWit, see Appendix A of [32].)

**How Arwen avoids this problem.** To avoid this problem, Arwen ensures that parties *never* need to send each other signatures on off-blockchain transactions that point to other off-blockchain transactions. Instead, parties only send each other signatures on off-blockchain transactions that point directly to the Arwen on-blockchain escrows. This further implies that if an off-blockchain transaction

comes with a smart contract (*e.g.*, like the HTLC smart contract on the off-blockchain puzzle transaction), then each clause on that smart contract must require the signature of only one party. If a single clause required the signature of more than one party, then parties would need to send each other signatures on off-blockchain transactions that point to other off-blockchain transactions, which is vulnerable to transaction malleability. The reader is invited to check that Arwen is robust to transaction malleability, by checking that each clause on a smart-contract in an off-blockchain transaction only requires the signature of a single party; see Figures 4.3,2(b),4.

This is also why we can not use relative timelocks *i.e.*, `CheckSequenceVerify` [10] in Arwen. `CheckSequenceVerify` (which is used extensively in Lightning) provides a timelock which is relative to the time another transaction is confirmed on-blockchain. `CheckSequenceVerify` is therefore not safe to use on blockchains vulnerable to transaction malleability attacks, like BCH or ZEC. Instead, Arwen can only use absolute timelocks *i.e.*, `CheckTimeLockVerify` [44].

## D Smart Contracts

The source code for the Ethereum smart contracts can be found at: <https://github.com/cwcrypto/arwen-eth-contracts>.

### D.1 Unidirectional RFQ

Smart contracts *i.e.*, Bitcoin-script P2SH redeem-scripts used in our unidirectional RFQ (Section 4).

#### User Escrow:

```
OP_DEPTH, OP_3, OP_EQUAL
OP_IF
  OP_2, OP_PUSH <User Trade Key>, OP_PUSH <Exchange Trade Key>, OP_2, OP_CHECKMULTISIG
OP_ELSE
  OP_PUSH <twA>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_PUSH <User Refund Key>, OP_CHECKSIG
OP_ENDIF
```

#### Exchange Escrow:

```
OP_DEPTH, OP_3, OP_EQUAL
OP_IF
  OP_2, OP_PUSH <Exchange Trade Key>, OP_PUSH <User Trade Key>, OP_2, OP_CHECKMULTISIG
OP_ELSE
  OP_PUSH <twB>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_PUSH <Exchange Refund Key>, OP_CHECKSIG
OP_ENDIF
```

#### User Escrow Puzzle Transaction (Puzzle Output):

```

OP_DEPTH, OP_2, OP_EQUAL
OP_IF
  OP_RIPEMD160, OP_PUSH <puzzle>, OP_EQUALVERIFY
  OP_PUSH <Exchange Puzzle Pubkey>, OP_CHECKSIG
OP_ELSE
  OP_PUSH <tauA>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_PUSH <Alice Puzzle Pubkey>, OP_CHECKSIG
OP_ENDIF

```

#### **Exchange Escrow Puzzle Transaction (Puzzle Output):**

```

OP_DEPTH, OP_2, OP_EQUAL
OP_IF
  OP_RIPEMD160, OP_PUSH <puzzle>, OP_EQUALVERIFY
  OP_PUSH <Alice Puzzle Pubkey>, OP_CHECKSIG
OP_ELSE
  OP_PUSH <tauB>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_PUSH <Exchange Refund Pubkey>, OP_CHECKSIG
OP_ENDIF

```

## **D.2 Unidirectional Limit orders**

Smart contracts *i.e.*, Bitcoin-script P2SH redeem-scripts used in our unidirectional limit orders (Section 5). All scripts are the same as our unidirectional RFQ protocol smart contracts given in Appendix D.1 except for the user escrow puzzle transaction.

#### **User Escrow Puzzle Transaction (Puzzle Output):**

```

OP_1, OP_EQUAL
OP_IF
  OP_PUSH <tauC>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_RIPEMD160, OP_PUSH <puzzle>, OP_EQUALVERIFY
  OP_PUSH <Exchange Puzzle Pubkey>, OP_CHECKSIG
OP_ELSE
  OP_PUSH <Alice Refund Pubkey>, OP_CHECKSIGVERIFY
  OP_DEPTH, OP_1, OP_EQUAL
  OP_IF
    OP_RIPEMD160, OP_PUSH <cancel>, OP_EQUALVERIFY
  OP_ELSE
    OP_PUSH <tauA>, OP_CHECKLOCKTIMEVERIFY, OP_DROP
  OP_ENDIF
OP_ENDIF

```