The Arwen Trading Protocols*

Ethan Heilman, Sebastien Lipmann, Sharon Goldberg
Version 1.0, January 28, 2019

ABSTRACT

The Arwen Trading Protocol is a layer-two blockhchain protocol that allows 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, rather than in the exchange's wallet. The agent of escrow is the blockchain itself. Each individual 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, including BTC, LTC, BCH, ZEC as well as Ethereum and ERC-20 tokens.

1. INTRODUCTION

The promise of blockchain-backed cryptocurrencies is the ability to transact in digital assets without relying on a single trusted party. Blockchains therefore present a technical breakthrough that circumvents a long-standing result in cryptography: namely, that *atomic swaps* are impossible without the help of a trusted third party [26]. In an atomic swap, two parties that do not trust each other swap items, such that either (1) both Alice gets Bob's item and Bob gets Alice's item, OR (2) neither Bob nor Alice gets the other party's item. In fact, atomic swaps of digital assets are possible when the blockchain acts as the trusted third party [7].

Fully realizing the promise of blockchain-backed cryptocurrencies demands that atomic swaps be brought into the mainstream. Today, we still live in a world where the vast majority of cryptocurrency trading requires traders to trust either each other, or a centralized cryptocurrency exchange. This seems absurd. Much of the value of a blockchain-backed cryptocurrency follows because it does not rely on a single trusted party. Why, then, is a single trusted party still required when trading cryptocurrency?

The Arwen Trading Protocol seeks to deliver on this promise by bringing atomic swaps to the mainstream use case of cryptocurrency trading. With Arwen, traders can benefit from the liquidity at a centralized cryptocurrency-to-cryptocurrency exchange without trusting the exchange with custody of their coins. Instead, Arwen traders maintain custody of their cryptographic keys and their coins, and Arwen trades are backed by on-blockchain escrows. Each coin's native blockchain acts as the agent of escrow (*i.e.,* the agent of escrow for bitcoins is the Bitcoin blockchain). Arwen trades are fast because they happen off blockchain, and secure, because they are atomic swaps. Arwen ensures that even a compromised exchange cannot steal a trader's coins, and that a malicious user cannot grief or steal coins from the exchange.

Arwen is designed to align incentives for the trading use case, and to support trading instruments from traditional financial markets. Arwen is focused on the cross-blockchain trading use case, Arwen must support as many coins as possible. For this reason, the Arwen protocols are designed to work even with the "lowest common denominator" of Bitcoin-derived coins without SegWit [39] support. As a result, Arwen supports essentially all "Bitcoin-derived" coins, including Bitcoin (BTC), Litecoin (LTC), Bitcoin Cash (BCH), Zcash (ZEC)), *etc.*as well as Ethereum and ERC-20 tokens.

2. WHITHER ATOMIC SWAPS?

There has been a flurry of activity that seeks to bring *cross-blockchain atomic swaps* to the mainstream. A cross-blockchain atomic swap allows one cryptocurrency (*e.g.,* Bitcoin (BTC)) to be atomically swapped for another cryptocurrency (*e.g.,* Bitcoin Cash (BCH)). Nevertheless, there are number of subtle issues that prevent known solutions from seeing widespread adoption for cryptocurrency trading. We now highlight several of these issues, and explain how Arwen overcomes them.

2.1 The promise of atomic swaps.

Cross-blockchain atomic swaps seek to supplant today's dominant form of cryptocurrency trading: *custodial trading* at centralized cryptocurrency exchanges. With custodial trading, when users wish to trade on a centralized exchange, they must first deposit their coins at the exchange; this is done using an on-blockchain transfer of coins from the user's own wallet to the exchange's wallet. Trading occurs within the databases of the centralized exchange, and is not recorded on the

*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.

blockchain. Finally, once trading is complete, users regain custody of their coins by withdrawing their coins from the exchange; that is, the exchange uses an on-blockchain transaction to transfer coins from the exchange's wallet back to the user's wallet. Custodial trading at a centralized exchange exposes the user to serious counterparty risk—if the exchange is compromised, the exchange may be unable to transfer coin back to the user's wallet. This risk has been realized, starting with the hack of Mt. Gox [38] and affecting several centralized exchanges, *e.g.,* [10, 30, 16, 8, 24, 12, 18, 17, 31, 6, 40].

With cross-blockchain atomic swaps, a user would no longer need to trust a centralized exchange with custody of her coins. Instead, the user could maintain custody of her coins even while she trades, and the user's coins would not be at risk even if her counterparty becomes adversarial in the middle of a trade.

In an atomic swap of 1 BTC for 20 BCH, it is cryptographically guaranteed that: (1) if the Alice transfers her 1 BTC to her counterparty, even a malicious counterparty cannot prevent Alice from claiming her 20 BCH, and (2) if the counterparty transfers his 20 BCH to Alice, even a malicious Alice cannot prevent the counterparty from claiming his 1 BTC.

Atomic swaps are an even stronger security paradigm than "second send" protocols such as ShapeShift. In a ShapeShift trade, Alice first transfers her 1 BTC to ShapeShift's wallet, and then ShapeShift transfers 20 BCH to Alice's wallet. This is not an atomic swap, because an adversarial ShapeShift could always decide to keep Alice's 1 BTC without paying out her 20 BCH. Thus, while such "second send" protocols are sometimes referred to as *non-custodial*, they in fact have brief window of custody.

2.2 The challenge of providing liquidity.

Most decentralized exchange (DEX) protocols, including EtherDelta [1], 0x [37], and SparkSwap [4], are peer-to-peer trading systems, where 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 it means that 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 can be 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. This follows because Arwen trades only involve the movement of coins from the user's wallet to the exchange's wallet, without the involvement of any intermediaries. In other words, Arwen can be thought of as a secure settlement leg for the centralized exchange. Meanwhile, orders are priced as usual, using the centralized exchange's existing orderbook.

2.3 The pitfalls of on-blockchain protocols.

Some of the best-known atomic swaps protocols are *on-blockchain protocols*, where the swap does not actually execute until certain transactions are confirmed by the blockchain.

Ethereum DEX protocols. Ethereum DEX protocols, including EtherDelta [1] and 0x [37], use the Ethereum blockchain to trade one ERC-20 token for another ERC-20 token. Both EtherDelta and 0x uses the following protocol framework. First, Alice broadcasts an order to the network without identifying a counterparty. Some counterparty Bob then sees Alice's broadcast, decides to trade with Alice, adds his information to the order. Bob then posts the order to the Ethereum blockchain, where it is then executed by a smart contract on the Ethereum blockchain.

The Bitcoin TierNolan Protocol. The TierNolan protocol [34] 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 the solution x secret. Next, Alice locks up 1 BTC in an HTLC smart contract on the Bitcoin blockchain which stipulates: "before time $\tau_{\mathcal{A}}$, the 1 BTC can be claimed by a transaction signed by Bob containing the solution to puzzle y". Bob similarly locks up 100 LTC on the Litecoin blockchain in an HTLC, which stipulates: "before time $\tau_{\mathcal{B}}$, the 100 LTC can be claimed by a transaction signed by Alice containing the solution to the puzzle y". The atomic swap executes when Bob reveals the x or when he claims the 1 BTC by signing and posting a transaction to the Bitcoin blockchain that contains the solution x. (For convenience, we will call this a *solve transaction.*) Once this happens, Alice learns the solution x from the Bitcoin blockchain and can sign and post a solve transaction to the Litecoin blockchain that claims Bob's 100 LTC.

Security follows from the cryptographic hash function H (which ensures that no one can learn the solution x given only the puzzle y) and the fact that Bob must reveal x on the blockchain in order to claim his coins.

There are several issues with on-blockchain protocols.

Speed. On-blockchain execution is slow, because it is limited by the speed at which the blockchain confirms blocks. The expected time to confirm a single block is about 10 minutes for Bitcoin (BTC) and Bitcoin Cash (BCH), 2.5 minutes for Litecoin (LTC) and 15 seconds for Ethereum (ETH). Moreover, a single confirmation is often insufficent to ensure that a transaction can not be reversed [13]; for instance, the cryptocurrency exchange Kraken waits 6 confirmations (60 minutes) for BTC and 30 confirmations for ETH (6 minutes) [19]. Even a few seconds of latency can be problematic when trading, especially given that cryptocurrency prices are famously volatile. An atomic swap protocol that is compatible with high-frequency trading strategies must ensure that trades execute instantly.

Scalability. If each individual trade needs to be confirmed on the blockchain, and a healthy trading ecosystem leads to many trades, then the blockchain itself

will be clogged up with transactions resulting from each individual trade. Note that today's custodial trading ecosystem avoids this problem, since the only transactions that need to be confirmed on the blockchain correspond to deposit and withdrawal of coins from the exchange. To avoid clogging the blockchain, an atomic swap protocol should require the blockchain to confirm only the consolidated balance across multiple trades.

Front-running. The EtherDelta and 0x protocols require the trader to broadcast information about the trade to all nodes on the blockchain before the trade is executed by the blockchain. This can lead to race conditions. Any blockchain node can learn the details of Alice's trade with Bob, and attempt to profit from it by front-running Bob's trade with its own trade [37]. These protocols also allow a trader to cancel a order (*e.g.,* due to changing market conditions) by posting a cancellation request the blockchain. The on-blockchain nature of this cancellation is also subject to race conditions, because a blockchain node Charlie could add himself as a counterparty to Alice's order before the blockchain has a chance to cancel it. These race conditions, however, are avoided by the TierNolan protocol because Alice and Bob are required to identify each other counterparties at the moment that they lock up coins in the HTLC smart contract.

The Arwen Trading Protocol avoids speed and scalability pitfalls, because individual trades execute off-blockchain. Arwen also avoids the front-running problem by building trading instruments that complement the puzzle approach pioneered by TierNolan; see Section 2.6.

2.4 Trading in a layer-two protocol

Because Arwen atomic swaps are executed entirely off-blockchain, Arwen falls into the same class of blockchain layer-two protocols [25] as the Lightning Network [29].

In a blockchain layer-two protocol, parties first lock up their coins in an on-blockchain smart contract. The coins locked in the smart contracts are then transferred between the parties via off-blockchain interactions. Finally, the smart contract is closed on the blockchain, reflecting the parties' balance, consolidated across all the off-blockchain transfers. Blockchain layer-two protocols are fast, because coins are moved off-blockchain, and scalable, because only consolidated balances are posted to the blockchain.

Arwen escrows. Arwen's on-blockchain smart contracts are called *escrows*. Before trading begins, Alice deposits her coins in an on-blockchain *user escrow*, where the agent of escrow is the blockchain itself. The exchange also deposits coins in an on-blockchain *exchange escrow*, to prove to Alice that the exchange holds enough coins to collateralize its trades. Each exchange escrow is specific to a single user (*i.e.,* Alice) and a single coin (*e.g.,* Bitcoin Cash). Escrows are opened and closed by confirming a transaction on the coin's native blockchain. (So, to open a Bitcoin Cash exchange escrow for Alice, the exchange confirms a transaction on the Bitcoin Cash blockchain.)

Multiple fast off-blockchain atomic-swap trades can be executed against a given pair of escrows. Thus, if Alice wanted to trade 1 BTC for 20 BCH, Alice trade would be backed by Alice's user escrow for Bitcoin, and Alice's exchange escrow for Bitcoin Cash. If Alice has multiple open user escrows with a given exchange (*e.g.,* a user escrow for BTC and a user escrow for ZEC), Alice can pair them with any of her open exchange escrows (*e.g.,* an exchange escrow for BCH and an exchange escrow for ETH). This way, Alice can trade across multiple currency pairs (BTC-BCH, BTC-ETH, ZEC-LTC and ZEC-ETH).

To compensate the exchange for locking its coins in escrow, Alice pays an *escrow fee* each time the exchange funds an exchange escrow for her. The exchange could fund exchange escrows from its own inventory of coins. Alternatively, the exchange could fund exchange escrows using deposits provided by custodial users of the exchange, and pay these custodial users a part of the escrow fee. This creates a new interest-bearing feature that the exchange can offer to its custodial users.

Arwen *vs.* the Lightning Network. While Lightning is designed for the use case of fast peer-to-peer Bitcoin micropayments, Arwen is designed for the use case of cryptocurrency trading. Arwen therefore avoids some of the incentive issues that result when layer-two micropayment systems (like Lightning) are repurposed to enable large value trades via atomic swaps.

In the Lightning Network, Alice, Bob and any intermediate nodes on the path first lock their coins in on-blockchain smart contract—one smart contract for every consecutive pair of nodes on the path—and then move coins along the path via a series of off-blockchain atomic swaps. Arwen does not use a path of intermediate nodes; instead, coins are transferred only between Alice and the exchange. This eliminates the risk that an intermediate node will strategically disrupt a trade in order to manipulate market prices.

Lightning only works with blockchains that have SegWit support; such as only Bitcoin and Litecoin. Arwen does not require SegWit support. This means that Arwen works with more "Bitcoin-derived" blockchains, including both coins with SegWit support and coins which have not deployed SegWit such as Bitcoin Cash, Zcash, and others. Technically speaking, supporting "Bitcoin-derived" blockchains that lack SegWit requires Arwen to withstand transaction malleability attacks, as discussed in Section 7. Arwen also has support for Ethereum and ERC-20 tokens [36], as discussed in Section 6.

2.5 Dealing with lockup griefing.

Lockup griefing affects any protocol that requires users to lock coins in a smart contract. We describe the problem, and explain how Arwen solves it via reputation and a novel *escrow fee* mechanism.

Lockup griefing in TierNolan. In the TierNolan protocol, Alice's coins and Bob's coins are locked in their smart contracts until either (1) the trade executes or (2) the timelock on the smart contract expires at time τ. These timelocks $\tau_{\mathcal{A}}, \tau_{\mathcal{B}}$ must at least as long as the time it takes to reliably confirm transactions on the blockchain, *i.e.,* several hours. The fact that coins

must locked for a long time can creates "lock-up griefing" problem, where Alice locks her coin in her smart contract, but Bob refuses to lock his own coins. Alice's coins are thus pointlessly locked in the smart contract until it expires at time $\tau_{\mathcal{A}}$. (Alice could similarly launch a lockup-griefing attack on Bob.) TierNolan lacks a mechanism to incentivize Alice and Bob to avoid lockup griefing. This is especially problematic when Alice and Bob are two random people that met over the Internet.

Lockup griefing in Lightning. Lightning nodes earn fees for transfers of coins across their channel; however, if no transfer of coins occurs, no fees will be earned, which can lead to lockup griefing. Worse yet, a node on the Lightning path between Alice and Bob could also decide to strategically close its smart contract, breaking the path from Alice to Bob. The makes it risky to use Lightning for very high-value trades, where the incentive to disrupt a trade can be very significant. This risk is further exacerbated by the fact that Alice and Bob may have no relationship with the intermediate nodes on the Lightning path.

Using reputation to avoid lockup griefing. Arwen sidesteps the risk of lockup griefing because its interactions only involve the trader Alice and the exchange, rather than a peer Bob or a path of intermediate nodes. The exchange has no incentive to launch a lockup griefing attack against Alice; such an attack harms the exchange's reputation, and prevents Alice from trading, which is the exchange's main source of revenue.

Using escrow fees to avoid lockup griefing. The exchange, however, must protect itself from lockup griefing by a trader Alice, who might ask the exchange to lock up coins in exchange escrows willynilly, without actually executing any trades against those exchange escrows. Arwen therefore requires Alice to lock up her own coins in a user escrow before she can ask the exchange to lock up its own coin in an exchange escrow. Arwen also introduces a novel escrow fee mechanism (see Section 3.5) that compensates the exchange for locking up coins for Alice. Arwen's escrow fees are an in-band mechanism designed to avoid the introduction of out-of-band payments or of a superfluous fee token. The mechanism generates revenue for the exchange from locked-up coins, while encouraging the user to close her exchange escrows (and unlock the exchange's coin) in a timely manner once she is done trading.

2.6 Atomic swaps as trading instruments.

The vision behind Arwen is to use atomic swaps in traditional trading instruments. For this reason, Arwen is specifically designed to avoid a misalignment of incentives. We've already discussed how Arwen aligns incentives when opening and closing escrows; we now focus on the incentives involved in trading.

To understand the importance of designing the incentive structure used in trading instruments, we return again to the TierNolan protocol.

TierNolan as an American call option. The TierNolan Protocol is asymmetric, because only Bob chooses and knows the secret solution x. This means that Bob has the unilateral ability to decide whether or not the atomic swap executes, by revealing x (or not). Because the timelocks $\tau_{\mathcal{A}}, \tau_{\mathcal{B}}$ on the smart contracts must at least 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 τ. 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 the TierNolan Protocol, Bob gets the option for free, resulting in a misalignment of incentives.

This American call option is implicit in many atomic swap protocols. For instance, it exists whenever the Lightning Network is used for peer-to-peer trading between Alice and Bob. By contrast, Arwen swaps are explicitly designed to support different trading instruments. The first version of Arwen supports Request For Quote (RFQ) trading instrument, while limit orders are next on the Arwen roadmap.

RFQ trading with Arwen. Arwen's RFQ trading instrument is a off-blockchain atomic swap. In an RFQ trade, the exchange commits to a price, called the *quote*, before Alice decides whether or not to place an order for the trade. (Request: "How many BCH can I buy for 2 BTC?" Quote: "You can buy 40 BCH, quote open for 1 second") If Alice places the order before quote expires, Alice cannot back out of the trade and the exchange is expected to execute a trade (of exactly 2 BTC for 40 BTC). Importantly, RFQs are inherently asymmetric, because Alice gets to decide whether the trade happens or not. Therefore, to align incentives, the exchange's quote includes a spread around the current market price; this compensates the exchange for any volatility in market prices between the time the quote is given and the time the trade is executed.

Arwen RFQs also include an way out for the exchange. In times of strife, when the exchange is unable to execute a trade against a quote it provided, the exchange does have the option to abort a trade. The abort is possible because Arwen's off-blockchain RFQ trading instrument is built from HTLC smart contracts similar to those of TierNolan. Specifically, the exchange chooses a secret solution x to a puzzle $y = H(x)$, and releases x in order to execute the trade. To abort the trade, the exchange refuses to release x. The aborted trade, however, is ungraceful and costly because it requires the user and exchange to stop trading and close the escrows backing the aborted trade. While no coins are lost, this sufficiently harmful to the exchange's reputation that we would expect an exchange to avoid aborting whenever possible. A full protocol description is in Section 5.

Limit orders with Arwen. Arwen also supports off-blockchain fill-or-kill limit orders, where the user to tells the exchange the lowest price at which she is willing to buy or sell coins. (*i.e.,* Order: "I will sell 2 BTC if I can buy at least 40 BCH") A fill-or-kill order is only executed by the exchange if it can be completely filled (*i.e.,* if all 40 BCH can be bought by the user). The user

also has the ability to cancel the order. This instrument is well aligned with the exchange's incentives, because it can execute the limit order instantly, once market prices align with the user's order. It also aligns with the user's incentives, because she can essentially "name her price" and easily adjust her price based on market conditions (by cancelling the order).

2.7 White paper overview.

The rest of this paper is organized as follows. In Section 3, we start with an overview of Arwen that covers Arwen escrows, escrow fees, and the basics of off-blockchain trading. We then provide a detailed description of Arwen's unidirectional RFQ protocol for "Bitcoin-derived" blockchains, including those that do not support SegWit. Section 7 explains the reasons why this protocol withstands transaction malleability attacks on blockchains like BCH that lack SegWit support. We describe how to port the unidirection RFQ protocol to Ethereum and ERC-20 tokens in Section 6. Arwen's bidirectional RFQ protocol is in Section 8 and bidirectional limit order protocol is in Section 9. We compare Arwen to other atomic swap and layer-two protocols in Section 10.

3. ARWEN OVERVIEW

The Arwen Trading Protocol is a blockchain-backed two-party cryptographic protocol between a user Alice and a centralized cryptocurrency 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 compensates the exchange for locking up its coins, Alice pays and *escrow fee* to the exchange using the coins Alice locked in her user escrow. Alice can now trade across a pair of escrows. Each individual trade is an off-blockchain atomic swap, using one of the Arwen trading instruments described in Sections 5. Finally, the escrows are closed and the coins are released to the wallets of the user and the exchange.

3.1 The Arwen Daemon

The user executes the Arwen Trading Protocol using the Arwen Daemon. The Arwen Daemon is an open-source executable that the user downloads to her local machine. The Arwen Daemon allows the user to engage in the Arwen Trading Protocol without trusting a third-party or a webserver, thus realizing the promise of non-custodial cryptocurrency trading. The Arwen Daemon performs the cryptographic operations involved in the Arwen Trading Protocol, posts and verifies transactions from relevant blockchains, and stores the secret trading keys used to securely trade against the Arwen escrows.

3.2 Opening on-blockchain escrows.

Escrows are opened and closed by confirming a transaction on the coin's native blockchain. (Thus, the transaction that opens an escrow for bitcoins is confirmed on the Bitcoin blockchain.) Opening and closing escrows takes the same amount of time it would take to deposit or withdraw coins from a custodial centralized exchange.

This exposition below assumes that Alice wishes to trade her bitcoins for some litecoins, as shown Figure 1.

User escrow. Alice funds the on-blockchain *user escrow* that is specific to this exchange. The user escrow locks *e.g.,* 5 BTC from the user's wallet on the Bitcoin blockchain until some pre-agreed-upon expiry time $tw_{\mathcal{A}}$. The initial balance in this escrow is 5 BTC owned by the user, and 0 BTC owned by the exchange. While the escrow is open, these coins are locked and cannot be used for anything other than cryptocurrency trades with the exchange. Once the escrow is closed, the balance of the coins owned by the user are released and transferred to the user's wallet, and the balance of the coins owned by the exchange are released and transferred to the exchange's wallet.

Exchange escrow. The exchange funds the *exchange escrow* that is specific to this user Alice. To open the exchange escrow, Alice pays the exchange an escrow fee, as described in Section 3.5. The exchange escrow locks 500 LTC from the exchange's wallet on the Litecoin blockchain until some pre-agreed-upon expiry time tw. The locked coins cannot be used for anything other than cryptocurrency trades with this specific user. The initial balance in this escrow is 0 LTC owned by the user, and 500 LTC owned by the exchange. Once the escrow is closed, the balance of the coins owned by the Alice are released and transferred to Alice's wallet, and the balance of the coins owned by the exchange are released and transferred to the exchange's wallet.

Pairing. Alice can execute multiple fast off-blockchain trades against any (user escrow, exchange escrow) pair. If Alice has multiple open user escrows with a given exchange, she can pair them with any of her open exchange escrows with that exchange. In the example of Figure 1, initially pairs her 5 BTC user escrow and her 500 LTC exchange escrow, allowing her to sell up to 5 BTC and buy up to 500 LTC.

Escrow expiry times. Escrows come with an expiry time that protect each party against a malicious counterparty. As long as the exchange is not compromised, the user can close her escrows early, before they expire. Escrow expiry times can vary, but must be longer than the time it takes to reliably confirm a transaction on blockchain. Thus, expiry times are least several hours after the escrow has been opened.

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

> "coin may be released from escrow via a transaction that is signed by both the user's ephemeral key and the exchange's ephemeral key, OR after time tw, the party that funded that escrow can unilaterally withdraw coins from the escrow by signing a transaction using their ephemeral key."

Each ephemeral key is chosen and used for this specific escrow only. This way, even if a trading key is compromised, there is no risk to the coins that remain in the user wallet or the exchange wallet. The user's ephemeral trading key is stored in the Arwen Daemon on the user's local machine.

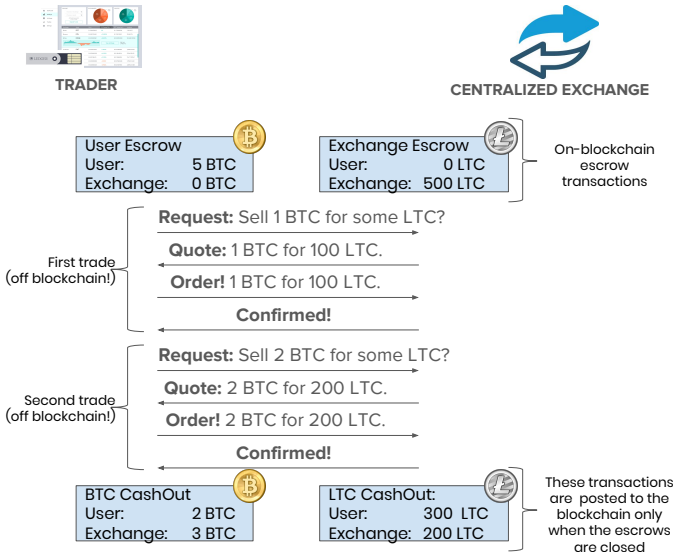

Figure 1: Arwen Trading Protocol for a two RFQ trades between the user and exchange.

Bitcoin script implementation. When Arwen escrows lock coins on blockchains that support the Bitcoin Script language (*e.g.,* BTC, BCH, LTC, ZEC, *etc.*), the Arwen escrow smart contract operates is quite simple, comprising only 16 opcodes. (Bitcoin Script is a highly restricted language, reminiscent of Assembly language.)

User escrows and exchange escrows are transactions confirmed on their native blockchain. The reader might therefore wonder if a third party can inform their own trading strategies by scanning the blockchain to find Arwen escrows. Fortunately, with Bitcoin-fork blockchains, a third party can only observe that a certain amount of coins has been transferred into a smart contract address; the amount of coins is visible on the blockchain, but the details of the smart contract are completely unknown to the third party. This follows because opening an escrow amounts to funding a P2SH-type address on a "Bitcoin-fork" blockchain. A P2SH address is just a cryptographic hash of the smart contract itself. No details of the smart contract are apparent from this hash, other than the fact that it is a smart contract. Moreover, the P2SH-type address is commonly used by other systems, not just by the Arwen Trading Protocol.

Ethereum smart contract implementation. The Ethereum and ERC-20 implementation of Arwen uses an escrow smart contract that mimics the UTXO transaction paradigm that is used on Bitcoin. Once the smart contract is funded, it enters the OPEN state, and remains in the OPEN state until either a transaction is posted that is (1) doubly-signed by the users's ephemeral key and the exchange's ephemeral key (this is 2-of-2 multisig condition of the Arwen escrow), or (2) the escrow expires after time tw, and the party that funded the escrow posts a signed message that unlocks the coins in the escrows (this is the timelock condition of the Arwen escrow). See the description in Section 6.

3.3 Off-blockchain trading via atomic swaps.

Arwen supports several off-blockchain trading instruments, backed by the user escrow and exchange escrows contracts. These trading instruments and their accompanying cryptographic protocols are described in Sections 5,6,9. Arwen trades are fast, because they happen off-blockchain. To trade, the user's Arwen Daemon and the exchange send cryptographic messages between themselves. Because no other party can see these messages, the trade is protected from front-running, griefing, and other strategic manipulations.

Each trade is an atomic swap, and cannot be reversed once it executes. If a trade successfully executes, it results in a pair of off-blockchain transactions that reflect the balance of coins in the escrows after the trade. These transactions protect the user and the exchange in case the other party becomes uncooperative (*i.e.,* malicious or unresponsive), by allowing each party to *unilaterally* close the escrows without the help of the other party.

3.4 Closing on-blockchain escrows.

Once an escrow is closed, the balance of coins owned by the user is released into the user's wallet, and the balance of coins owned by exchange is released into the exchange's wallet. For instance, consider Figure 1, where (1) the user escrow was initially funded with 5 BTC, and (2) the exchange escrow was initially funded with 500 LTC, (3) Alice performed two RFQ trades selling 3 BTC in order to buy 300 LTC, and then (4) decided to close both the user escrow and exchange escrow. When the user escrow is closed, 2 BTC will be transferred into Alice's wallet, and 3 BTC will be transferred into the exchange's wallet. When the exchange escrow closes, 300 LTC will be transferred to the user's wallet, and 200 LTC will be transferred to the exchange's wallet.

Cooperative close. In typical situations, where neither the exchange nor the user is malicious or compromised, then the user escrow and the exchange escrow can be closed at any time, even before they expire. Each

escrow is closed via a cooperative exchange of messages between the user and exchange. The output of this cooperation is a single *cashout transaction*, posted to the blockchain, that reflects the balance of the escrow and is doubly-signed by (1) the user's trading key and (2) the exchange's trade key.

Uncooperative close. The Arwen Trading Protocol is secure atomic swap protocol because it allows the user to unilaterally recover from the two unlikely situations which may occur if the exchange is compromised or unresponsive:

The exchange refuse to close an open escrow. In this case, the user can unilaterally close the escrow.

The exchange aborts a trade. An escrow is *frozen* whenever an exchange improperly aborts a trade against that escrow. A frozen escrow cannot be used for trading and must be closed. If the exchange refuses to cooperatively close a frozen escrow, Alice can unilaterally recover coins from the frozen escrow by connecting her Arwen Daemon to the Internet during a specific *coin-recovery time period*. The Arwen Daemon notifies the user about the coin recovery time period at the moment the Arwen Daemon detects that the exchange improperly aborted a trade.

Similarly, Arwen allows the exchange to unilaterally close escrows when the user is malicious or unresponsive.These unilateral recovery procedures are the main technical contribution of the Arwen protocols, and are specific to each of Arwen's trading instruments. See *e.g.,* Sections 5.5,5.6,8.48.5 for the procedures used in Arwen's unidirectional RFQ trading protocol.

3.5 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 closes 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 balance of 301 LTC sent to Alice's wallet and 199 LTC sent to the exchange's wallet.

4. SECURITY MODEL.

Arwen assumes that the exchange is almost always online, while the user is usually not online. Atomic swap security for users of Arwen follows from five assumptions.

1. The traded coins' native blockchain is secure. That is, when trading bitcoins we assume that the Bitcoin blockchain is secure.

2. The user's Arwen Daemon has not been compromised. (Note that the Arwen Daemon is open-source and therefore can be audited.)

3. The user correctly deposits coins in her user escrow. (This is exact same assumption on user behavior that is made whenever a user deposits coins at a cryptocurrency exchange.)

4. The user correctly informs the Arwen Daemon of the wallet addresses where she wants her coins transferred to upon closing each escrow. (This is exact same assumption on user behavior that is made whenever a user withdraws coins from a cryptocurrency exchange.)

5. The user remembers to come 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". (The users coins are at risk only if the exchange is compromised or malicious, and the user forgets to close escrows in a timely manner or to recover coins from frozen escrows.)

These assumptions only related to the blockchain (assumption 1), the user's computing environment (assumption 2) and the user's behavior (assumption 3-5). Security for the user makes no assumptions about the security or correctness of the cryptocurrency exchange or any other party.

Atomic-swap security for the exchange analogously follows from five analogous assumptions, and analogously makes no assumptions about the security or correctness of the user or any other party.

5. UNIDIRECTIONAL RFQS

The following protocol is *unidirectional* [32] because it only allows Alice to sell coins from her user escrow, and buy coins from her exchange escrow. Arwen's more complex *bidirectional* RFQ protocol is described in Section 8. This unidirectional protocol is for "Bitcoin-derived" blockchains, including those without SegWit, and thus withstands transaction malleability attacks for the reasons explained in Section 7. Section 6 explains how to port this protocol to Ethereum and ERC-20 tokens.

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 3.2. Both escrows must be open during a trade (rather than expired, frozen, or closed).

Each new trade generates a pair of timelocked *puzzle transactions* for puzzle y, along with corresponding solution x where x is chosen by the exchange and $y = H(x)$. One puzzle transaction spends the output of the user escrow and has timelock $\tau_{\mathcal{A}}$, and the other spends the output of the exchange escrow and has timelock $\tau_{\mathcal{B}}$. Each pair of puzzle transactions reflect the new balance of coins in the escrows after the trade, and "overwrites" the pair of transactions generated by the previous trade. The puzzle transactions and solution x and allow each party to *unilaterally* close escrows with the correct balance of coins, even if the other party becomes malicious.

5.1 Security assumptions.

Timelocks. The security of this protocol follows because we set the timelocks to be

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

where ϱ is the time required for transaction be reliably confirmed on the blockchain. Importantly, notice that there is no relationship between the escrow expiry times $(tw_{\mathcal{A}}, tw_B)$, which means we can pair any user escrow and exchange escrow, regardless of their expiry time.

Closing escrows in a timely manner. To withstand attacks by a compromised or 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 $(tw_{\mathcal{A}}, \tau_{\mathcal{B}})$.

5.2 Off-blockchain RFQ trades.

This exposition below follows Figure 1. We suppose that the user 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 δ". The exchange has now committed to executing the trade, should Alice choose to place an order before the quote expires at time δ. To align incentives, the quote reflects the current market price at the exchange, plus an additional spread that compensates the exchange for price fluctuations between the moment the quote is provided, and the moment the trade is executed.

To commit to the quote, the exchange chooses a secret solution x and computes a puzzle $y = H(x)$. The exchange sends Alice the following Litecoin *puzzle transaction* for the Litecoin blockchain. The puzzle transaction (1) is signed by the exchange's ephemeral key, (2) spends the output of the exchange escrow, and (3) reflects the current balance in the LTC exchange escrow, except that 200 LTC is locked in an HTLC smart contract stipulating that

> "The coins may only be unlocked via a transaction that contains the solution to puzzle y and is signed by the user's ephemeral key,
> OR
> after time $\tau_{\mathcal{B}}$, the exchange can unilaterally unlock coins from the escrow by signing a transaction using its ephemeral key."

In other words, the puzzle transaction sends 100 LTC to Alice's wallet, locks 200 LTC in an HTLC smart contract under puzzle y, and sends the remaining 200 LTC to the exchange.

(Notice that Alice could sign this puzzle transaction and post it to the Litecoin blockchain. This would release all coins locked in the user escrow, *except* those 200 LTC locked under the puzzle y. However, Alice does not sign or post anything to the Litecoin blockchain until she is ready to close her exchange escrow.)

Order. If the user decides not to place the order, then the escrows remain open and can be used for other trades. (This follows because only the exchange knows the secret solution x. Without x, the Litecoin puzzle transaction is useless to Alice. Nevertheless, if Alice

does decide to post the puzzle solution to the Litecoin blockchain, the exchange can reclaim the 200 LTC locked under the puzzle after its timelock expires at $\tau_{\mathcal{B}}$.)

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 (1) is signed by Alice's ephemeral trading key, (2) spends the output of the user escrow, and (3) reflects the current balance in the user escrow, except that 2 BTC is locked in an HTLC smart contract stipulating that

> "The coins may only be withdrawn via a transaction that contains the solution to puzzle y and is signed by the exchange's ephemeral key, OR
> after time $\tau_{\mathcal{A}}$, the user can unilaterally withdraw coins from the escrow by signing a transaction using its ephemeral key."

At this point in the protocol, Alice cannot abort the order, because 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 as if this trade was executed.)

Execute. If the user placed the order before time δ, then the exchange is expected to execute the trade by sending the solution x to the user. This executes the atomic swap, because now both the user and the exchange hold transactions that allow them to unilaterally close their escrows, reflecting the new balance after the trade. (Specifically, the user can unilaterally close the exchange escrow, and the exchange can unilaterally close the user escrow.) However, 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 the user escrow and the exchange escrows remain open.

What if the exchange does not properly execute the trade by releasing x? In this case, 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 5.6.

5.3 The magic of unidirectionality.

The security of our protocol follows, in part, from an observation first made by Spilman [32]. 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 resulting from the previous trade.

In other words, the user is incentivized to close the exchange escrow before it expires using the transactions resulting from the most recent trade. If the user goes rogue and closes the exchange escrow using transactions from a prior trade, she only hurts herself (because she will have fewer coins), not the exchange (because the exchange will have more coins)!

Similar reasoning explains why the exchange prefers to close the user escrow before it expires using the transactions from the most recent trade.

Paying escrow fees. The magic of unidirectionality also 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 in order 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. Specifically, the cashout transaction (1) spends the output of the user escrow, (2) sends 3.02 BTC to the exchange's wallet, and 1.98 BTC to the user's wallet. The same unidirectional argument means that the exchange is incentivized to have this cashout transaction "overwrite" the puzzle transaction received from the previous trade. This cashout transaction could then be signed by the exchange and posted to the blockchain if the exchange needs to unilaterally close the user escrow.

5.4 Closing escrows.

Cooperative close. If neither the user nor the exchange is unresponsive or malicious, escrows are closed prior to their expiry using the cooperative close procedure outlined in Section 3.4. Specifically, the user sends the exchange a signed *cashout transaction* reflecting the final balance in the escrow. The exchange then signs that transaction and posts it to the blockchain. For example, to cooperatively close the user escrow after the trade described above, the cashout transaction would (1) spend the output of the user escrow, (2) send 3 BTC to the exchange's wallet, and 2 BTC to the user's wallet, and (3) be signed by both the user and the exchange.

Incentives for a cooperative close. Cooperatively closing an escrow is in the interest of both parties, because it allows them to reduce mining fees by closing their escrows using a one transaction, rather than two (*i.e.,* the puzzle transaction and a *solve transaction* that contain the solution x to puzzle y). Moreover, because the puzzle transaction is generated in advance, it must include a highly conservative blockchain mining fee. This is necessary to cope with unpredictable fluctuations in mining fees between the time the puzzle transaction was created during the RFQ trade, and the time it might get posted to the blockchain to unilaterally close the escrow. A cooperative close of the exchange escrow also allows the user to earn a rebate on her escrow fees, as described in Section 3.5.

Unilateral close. However, if one of the parties refuses (or forgets) to engage in a cooperative close, our protocol ensures that their counterparty can unilaterally close the escrows and obtain *at least* those coins they rightfully hold according to the final balance in the escrow. (In fact, there are cases where a unilateral close allows the counterparty to obtain *additional coins* beyond those coins that they rightfully hold according to the final balance in the escrow.) Each party's procedure for closing an open escrows when the coun-

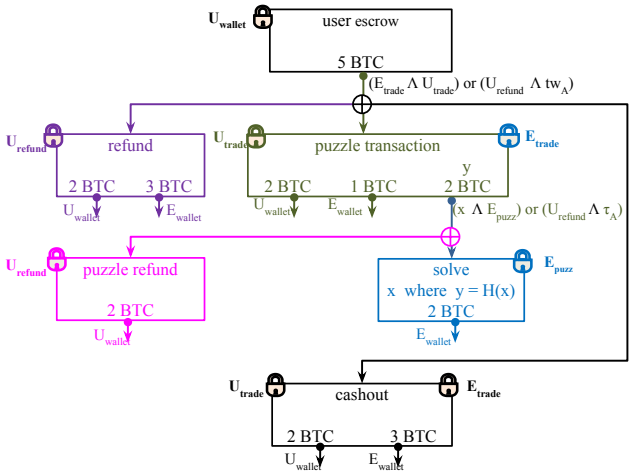

Figure 2: Diagram of transactions pointing to the user escrow for the unidirectional RFQ protocol of Section 5. Per Figure 1, we suppose that the user has already received 1 BTC and is engaging in a trade for an additional 2 BTC. Transactions in color are only posted to the blockchain if a party becomes uncooperative. 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. The transactions shown in green and blue are posted by the exchange to unilaterally close the escrow if the user becomes uncooperative, or forgets to close the escrow before time $tw_{\mathcal{A}}$. The transaction in purple is used by the user to unilaterally close the user escrow after it expires at time $tw_{\mathcal{A}}$; here we depict the case where all trades properly complete but the exchange did not cooperate to close the escrow, so the user benevolently transfers 2 BTC to the exchange's wallet, rather than "stealing" these BTC for herself. The transaction in magenta is used by the user to unilaterally release coins locked in puzzle transaction, after the puzzle expiry time of $\tau_{\mathcal{A}}$; this transaction is only used when the exchange posts a puzzle transaction but fails to post the corresponding solve transaction.

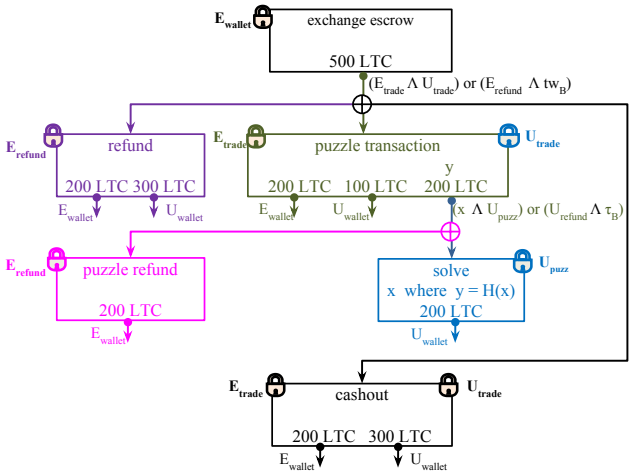

Figure 3: Diagram of transactions pointing to the exchange escrow for the unidirectional RFQ protocol of Section 5. Per Figure 1, we suppose that the user has already received 100 LTC and is engaging in a trade for an additional 200 LTC. Transactions in color are only posted to the blockchain if a party becomes uncooperative. The transactions shown in green and blue are posted by the user to unilaterally close the escrow if the exchange becomes uncooperative. The transaction in purple is used by the exchange to unilaterally close the exchange escrow after it expires at time $tw_{\mathcal{B}}$; here we depict the case where all trades properly complete but the user forgets to close the exchange escrow before it expired, so the exchange benevolently transfers 200 LTC to the user's wallet, rather than "stealing" these LTC for itself. The transaction in magenta is used by the exchange to unilaterally release coins locked in puzzle transaction, after the puzzle expiry time of $\tau_{\mathcal{B}}$; this transaction is only used when the user posts a puzzle transaction but fails to post the corresponding solve transaction.

terparty is malicious or uncooperative is in Section 5.5. The user's procedure for recovering coins from a frozen escrow is in Section 5.6.)

5.5 Unilaterally closing an open escrow

What happens if the user and exchange fail to cooperatively close an escrow before it expires? Here we consider only the case where the escrow is open, *i.e.,* where all trades against that escrow have properly completed. Recovery from an escrow that is frozen due to an aborted trade is described in Section 5.6. There are four relevant cases.

5.5.1 Exchange refuses to close exchange escrow.

In this case, Alice's Arwen Daemon can unilaterally close the exchange escrow before it expires at time $tw_{\mathcal{B}}$, by signing and posting the puzzle transaction and then signing and posting the solve transaction that resulted from the latest trade. (See Figure 3.) This releases coins to Alice's wallet and the exchange's wallet, according to the balance in the escrow after the last completed trade.

Alice's coins are safe if she closes the exchange escrow before time $tw_{\mathcal{B}}$. This follows because only Alice can close the exchange escrow before $tw_{\mathcal{B}}$, which protects her from race conditions with a malicious exchange or any other party. To see why, recall from Section 3.2 that, to close the exchange escrow before it expires, we require a transaction doubly-signed by both the user and the exchange. The RFQ protocol above ensures that only Alice has such a transaction—the Quote message provides the user with a puzzle transaction, signed by the exchange, that spends the output of the exchange escrow. Alice can unilaterally sign this puzzle transaction to create the required doubly-signed transaction, and unilaterally post this transaction to the Litecoin blockchain. Meanwhile, the exchange does not the ability to create such a doubly-signed transaction, because Alice never sends the exchange a signed transaction that spends the output of the exchange escrow.

Once the puzzle transaction is confirmed by the blockchain, Alice can use the corresponding solution x to sign and post a solve transaction to he blockchain, releasing the coins locked in the HTLC smart contract in the puzzle transaction to Alice's wallet. Also, Alice is protected from race conditions because only Alice can spend the coins locked in the HTLC before $tw_{\mathcal{B}}$. This follows because equation (1) is such that the HTLC's timelock $\tau_{\mathcal{B}}$ expires after time tw_B.

5.5.2 User forgets to close exchange escrow.

What happens if Alice forgets to close her exchange escrow before it expires? Alice's coins are at risk only if the exchange becomes malicious.

This follows because an exchange can unilaterally close an exchange escrow after it expires at time $tw_{\mathcal{B}}$ using a *refund transaction* (see Figure 3). The refund transaction (1) is signed by the exchange, and (2) spends the output of the exchange escrows. An honest exchange will (3) form the refund transaction so that it releases coins to Alice's wallet and the exchange's wallet according to the final balance of in the escrow. However, a malicious exchange could instead have the refund transaction release *all* coins to the exchange's wallet; this is why we say the user's coin are risk if she forgets to close her exchange escrow before it expires.

Meanwhile, an malicious user Alice cannot steal the exchange's coin even if the exchange decides to close the exchange escrow significantly after time $tw_{\mathcal{B}}$. This follows because even if Alice decides to race the exchange's refund transaction, Alice can only attempt to close the escrow with a puzzle transaction that either (1) reflects the final balance of in the escrow (*i.e.,* at the same balance at which the honest exchange is attempting to close the user escrow) OR (2) reflects the balance of in the escrow after any earlier completed trade (*i.e.,* a balance where the exchange holds *more coins* than it does in the final balance of the escrow, which is worse for Alice and better for the exchange!)

5.5.3 User forgets to close user escrow.

Suppose the user forgets to cooperatively close a user escrow before it expires at time $tw_{\mathcal{A}}$. Here, the exchange must unilaterally close the user escrow before time $tw_{\mathcal{A}}$. To do this, the exchange signs and posts the puzzle transaction, and then signs and posts the solve transaction, both of which resulted from the latest completed trade (see Figure 2). (If the final off-blockchain transfer against this escrow was due to Alice paying an escrow fee, then the exchange would instead close the escrow using the cashout transaction used to pay the escrow fee; see Section 5.3.) This releases coins to Alice's wallet and the exchange's wallet, according to the final balance in the escrow. The exchange must do this before time $tw_{\mathcal{A}}$ in order to avoid race conditions with a malicious Alice. This follows from reasons analogous to those given in Section 5.5.1.

5.5.4 Exchange refuses to close user escrow.

If the exchange refuses to cooperatively close a user escrow, the user's coins are not at risk. The user just needs to wait until the user escrow expires, at which point her Arwen Daemon will automatically and unilaterally close the user escrow using a refund transaction (see Figure 2). This follows because the user escrow smart contract allows the user to unilaterally unlock the coins in the user escrow after time $tw_{\mathcal{A}}$. Alice is in no rush to unilaterally close this user escrow—her Arwen Daemon can do this at any time after the user escrow expires at time $tw_{\mathcal{A}}$. This follows for reasons analogous to those in Section 5.5.2.

5.6 Recovering coins from frozen escrows.

Suppose that during the last trade, the user placed an order against a quote provided by the exchange, but the exchange failed to properly execute the order by releasing the preimage x. This is an unlikely case, because the exchange is supposed to execute any orders placed against any of its quotes. This last trade is considered to be improperly aborted. Whenever the Arwen Daemon detects an improperly aborted trade, it protects the user's coins by freezing the user escrow and exchange escrow that backed this trade. Frozen escrows cannot be used for trading.

Next, the user's Arwen Daemon immediately asks the exchange to cooperatively close the user escrow (on the Bitcoin blockchain) and exchange escrow (on the Litecoin blockchain) that backed this trade, under the assumption that the balance in these escrows is as it was prior to the final, improperly-aborted trade. If the exchange agrees to close both escrows, then no further action is required from the user.

However, suppose that the exchange refuses to close the frozen exchange escrow. In this case, the user's Arwen Daemon will immediately and unilaterally close the exchange escrow by posting the puzzle transaction from the aborted trade. Doing this, however, means that the coins in the exchange escrow, that were involved the aborted trade, are still locked in the puzzle transaction's HTLC smart contract until time $\tau_{\mathcal{B}}$. We call these coins the *outstanding coins*. The outstanding coins rightfully belong to Alice if the exchange sneakily executed the aborted trade on the Bitcoin blockchain without telling Alice; otherwise, the outstanding coins rightfully belong to the exchange. The remainder of this procedure is allows Alice to claim the outstanding coins whenever they are rightfully hers.

What happens next depends on whether the exchange also agreed to cooperatively close the user escrow. If so, no further action is required from Alice. This follows because the user escrow was closed according to the balance *before* to the aborted trade, so the outstanding coins rightfully belong to the exchange. The exchange can unilaterally claim the outstanding coins once the timelock $\tau_{\mathcal{B}}$ expires by posting a *puzzle-refund transaction*, signed by the exchange, that sends the outstanding coins to the exchange's wallet.

Otherwise, the exchange refused to cooperatively close the user escrow. Alice must come online during time window $(tw_A, \tau_{\mathcal{B}})$ to allow her Arwen Daemon to claim the outstanding coins. Thus, the Arwen Daemon scans the Bitcoin blockchain looking for transactions that spend the user escrow. There are several cases:

- *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 the Arwen Daemon. This follows because the outstanding coins rightfully belong to the exchange, and the exchange can unilaterally claim them once the timelock $\tau_{\mathcal{B}}$ expires via a puzzle-refund transaction.

- *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. The Arwen Daemon learns the solution x from solve transaction on the Bitcoin blockchain. The Arwen Daemon then uses x to form and sign the solve transaction for the Litecoin blockchain, claiming the outstanding coins for Alice. Importantly, the Arwen Daemon must complete this action before $\tau_{\mathcal{B}}$, because the outstanding coins can be unilaterally claimed by the exchange after $\tau_{\mathcal{B}}$, which puts Alice's coins at risk.

- *User escrow partially closed.* The exchange posted the puzzle transaction for *any* trade made against this user escrow, but the coin locked in this puzzle transaction on the *Bitcoin blockchain* are unspent. In this weird case, the exchange has not executed the aborted trade without telling the user, and so the user will not be able to claim the outstanding coins on the Litecoin blockchain. Nevertheless, the Arwen Daemon must still recover the coins locked in the puzzle output from the *user escrow* on the *Bitcoin blockchain* and send them back to the Alice's wallet. The user can unilaterally do this after the timelock on puzzle transaction expires at time $\tau_{\mathcal{A}}$, by posting a puzzle-refund transaction to the Bitcoin blockchain that sends locked coinsback to Alice's wallet. Note that $\tau_{\mathcal{A}} = tw_{\mathcal{A}}$ for all trades made against this user escrow, which means that Arwen Daemon can take this action during the time window $(tw_{\mathcal{A}}, \tau_{\mathcal{B}})$.

- *User escrow not closed.* Here the output of the user escrow is unspent. This again means that exchange has not executed the aborted trade, so the user cannot claim the outstanding coins. However, the Arwen Daemon must still recover the coins locked in the user escrow. The Arwen Daemon will post the refund transaction that releases coins to Alice's wallet and the exchange's wallet, according to the balance in the escrow prior to the aborted trade. The can be done anytime after time $tw_{\mathcal{A}}$ when the user escrow expires, which means that Arwen Daemon can take this action during the time window $(tw_{\mathcal{A}}, \tau_{\mathcal{B}})$.

Once this process is complete, both the user escrow and the exchange escrow are closed.

6. ETHEREUM UNIDIRECTIONAL RFQS

We now describe how to port the unidirectional RFQ protocol of Section 5 to Ethereum and ERC-20 tokens [36]. 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: (OPEN, PUZZLE, CLOSED). Coins are locked when the escrow is OPEN, and released when the escrow is CLOSED. The PUZZLE state is for trading.

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 $tw_{\mathcal{A}}$, (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 PUZZLE 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 τ (which are used for atomic swap trading). Then, a user escrow can move from the PUZZLE 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 $\tau_{\mathcal{A}}$

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 5. 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 5 (see also Figure 2). As in Section 5, 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 5.5, 5.6.

ERC-20 smart contract. Our ERC-20 implementation of Arwen is identical to the Ethereum implementation, with the following key modification.

The ERC-20 implementation of the Arwen escrow smart contract includes an additional state, UNFUNDED, which is used to lock ERC-20 tokens in Arwen escrows. (Recall that an ERC-20 token is created via the implementation of a single standard ERC-20 smart contract on the Ethereum blockchain, where the state of the ERC-20 contract tracks the number of tokens held by each account holder of the token. Thus, the Arwen escrow smart contract must alter the state of the ERC-20 smart contract in order to lock coins in escrow.)

To see how this is done, we describe the three-step process used by Alice to lock 5 tokens in a user escrow. First, Alice posts the user escrow smart contract to the Ethereum blockchain; at this point, the escrow is in the UNFUNDED state. Second, Alice call the *approve* method in the token's ERC-20 smart contract, indicating that she approves the transfer of 5 tokens from Alice's address to the user-escrow smart contract's address. Third, Alice calls the *fund-escrows* method in the user escrow smart contract, which interacts with the ERC-20 smart contract to transfer the required 5 token into the user escrow smart contract. Similarly, when the user escrow is CLOSED, it interacts with the ERC-20 smart contract to release the balance of tokens from the user-escrow smart contract address into both Alice's address and the exchange's address.

7. 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 σ on message m.) This means that a party that holds the secret signing key can easily produce multiple valid signatures $\sigma, \sigma', \ldots$ on a single message m. Worse yet, even a party that does not know the secret signing key can take a valid signature σ on a message m, and maul σ to obtain a different valid signature σ' 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 [39], 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 call 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 [25].)

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 2,3,4.

This is also why we can not use relative timelocks *i.e.,* CheckSequenceVerify [9] 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 [35].

8. BIDIRECTIONAL RFQS

The following Arwen RFQ protocol is *bidirectional*, because it allows Alice to both buy and sell coins from her user escrow, and to both 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.

Like the unidirectional protocol of Section 5, 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 3.2. Both escrows must be open during a trade (rather than expired, frozen, or closed). In what follows, we first overview the technical tools used in our bidirectional protocol, describe the off-blockchain trading protocol, and explain how escrows can be securely closed when one party becomes malicious or uncooperative. The transaction diagram for the user escrow and exchange escrow is in Figure 4.

8.1 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. We outline some tools below.

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 2 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 τ_B. The payU puzzle has an HTLC smart contract that is similar to the one in Figure 3 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 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 relies on "the magic of unidirectionality" (Section 5.3). Instead, we "overwrite" transactions resulting from old trades by cancelling them, adapting the idea of *justice transactions* from the Lightning Network [29].

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:

> "The coins may only be paid to the exchange after time tw, OR the coins return to the user if the user reveals the 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 anytime before time tw (see Figure 4).

Previous transactions. The following notation is useful for our protocol description and analysis. Let pay$*$_postE be a puzzle transaction from the current trade (where $*$ is either U or E). For convenience, we use

the notation prev(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 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 [9], instead we can only use the absolute timelocking mechanism CheckLockTimeVerify [35]. Thus, our escrows always have a fixed time before they must be closed. Additionally, as discussed in Section 7, 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 even when they do not already hold any of those coins (see Section 3.5).

8.2 Security assumptions.

Timelocks. The security of this protocol follows because we set the timelocks to be

$$\tau_{\mathcal{A}} > \max(tw_{\mathcal{A}}, tw_{\mathcal{B}}) + 2\varrho \qquad \tau_{\mathcal{B}} > \tau_{\mathcal{A}} + 2\varrho \qquad (2)$$

where ϱ is the time required for transaction be reliably confirmed on the blockchain. Importantly, notice that there is no relationship between the escrow expiry times $(tw_{\mathcal{A}}, tw_B)$, which means we can pair any user escrow and exchange escrow, regardless of their expiry time.

Closing escrows in a timely manner. To withstand attacks by a compromised or malicious user:

> The exchange must close its user escrows before they expire at time $tw_{\mathcal{A}}$, and its exchange escrows before they expire at $tw_{\mathcal{B}}$.

To withstand attacks by a compromised or unresponsive exchange:

> The user must close her user escrows before they expire at time $tw_{\mathcal{A}}$, and her exchange escrows before they expire at $tw_{\mathcal{B}}$.

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. Also, if a trade aborts or the exchange refuses to cooperatively close an escrow, the user must sometimes come online at a later time in order to recover her coins. This coin recovery periods are only required if the exchange becomes unresponsive or hacked, and the user will always learn exactly what the coin recovery period is at the time she attempts to close her escrows.

8.3 Off-blockchain RFQ trades.

Before each RFQ begins, we have the following off-blockchain setup step. This setup could happen at any time (even immediately after the previous trade).

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 $C_{\mathcal{E},\mathcal{A}}$ and $C_{\mathcal{E},\mathcal{B}}$, computes $j_{\mathcal{E},\mathcal{A}} = H(C_{\mathcal{E},\mathcal{A}})$ and $j_{\mathcal{E},\mathcal{B}} = H(C_{\mathcal{E},\mathcal{B}})$, and sends $j_{\mathcal{E},\mathcal{A}}$ and $j_{\mathcal{E},\mathcal{B}}$ to the user. The solution x and cancel values $C_{\mathcal{E},\mathcal{A}}, C_{\mathcal{E},\mathcal{B}}$ are kept secret by the exchange. The user responds by choosing a random secret cancel value $C_{\mathcal{U}}$, computing $j_{\mathcal{U}} = H(C_{\mathcal{U}})$, and sending $j_{\mathcal{U}}$ to the exchange.

Then, each RFQ is an off-blockchain protocol comprising the following four steps. In the following protocol description, we assume that Alice specifies the amount of coin she wishes to *buy* in the Request stage.[1]

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. 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 the following:

1. 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 $C_{\mathcal{E},\mathcal{A}}$. 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 $C_{\mathcal{E},\mathcal{B}}$.

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 δ". 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 $C_{\mathcal{U}}$. 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

[1]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.

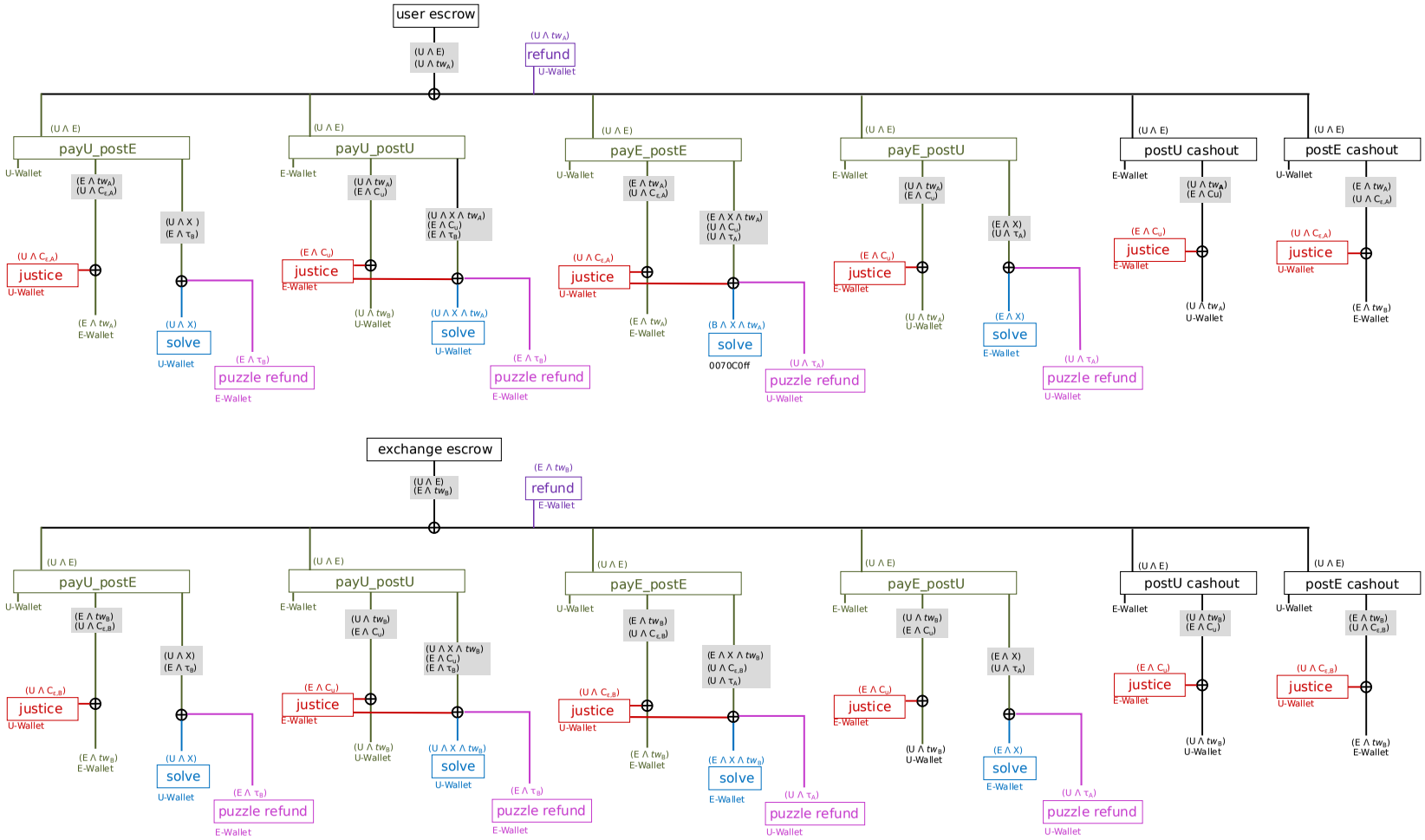

Figure 4: Transaction diagram for bidirectional RFQ and bidirectional limit order protocols.

puzzle sent to Alice during the quote phase of the unidirectional protocol in Section 5.2.)

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 $C_{\mathcal{U}}$. 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 uncancelled 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 5.2.)

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 uncancelled 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 prev(payE_postE). At this point in the protocol, the only uncancelled puzzle transactions held by the exchange are the current payE_postE and the current payU_postE puzzle transactions.)

2. The solution x for the current trade.

3. Two 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 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 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 postE cashout (b) spends the output of the user escrow and (b) can be cancelled under $C_{\mathcal{E},\mathcal{A}}$. Otherwise, the 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 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 payU_postU and the current payE_postU puzzles. However, prev(payU_postU) and prev(payE_postU) remain valid.)

8.4 Closing an escrow.

The process for cooperatively closing these escrow is identical to that of unidirectional RFQ protocol of Section 5.4. 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.

8.4.1 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 postU cashout transaction for that escrow.

Suppose that upon attempting to close an escrow, the user sees that exchange has posted a cancelled 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 $\tau_{\mathcal{A}}$ and $\tau_{\mathcal{B}}$ to recover the coins from the current trade, using a procedure similar to that used to recover from frozen escrows in Section 5.6. 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 8.5.2. Before tw, the user must post the payU_postU puzzle from the Cancelled-Quote trade. The exchange must then come online after time $\tau_{\mathcal{B}}$ 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.

8.4.2 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 $(tw, \tau_{\mathcal{A}})$ 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 $\tau_{\mathcal{B}}$. 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 $tw < \tau_{\mathcal{B}}$)!

8.5 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.

8.5.1 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.

8.5.2 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 $\tau_{\mathcal{B}}$ 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 $\tau_{\mathcal{B}}$). 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 $\tau_{\mathcal{A}}$ 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 $\tau_{\mathcal{B}}$ 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 8.4). 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 $\tau_{\mathcal{B}}$.

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 uncancelled 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).

8.5.3 *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 5.3.) (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 $(tw, \tau_{\mathcal{A}})$ if the exchange decides it wants to execute the trade on the blockchain.) The user would then come online between time $\tau_{\mathcal{A}}$ and $\tau_{\mathcal{B}}$ and use the usual procedure for recovering from frozen escrows.

9. BIDIRECTIONAL LIMIT ORDERS

The following protocol is for *bidirectional fill-or-kill limit orders*. The protocol allows Alice to place a single limit order for a specified amount and limit 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. Alice also has the option to cancel the limit order at any time.

Technically speaking, this protocol is almost identical to the bidirectional RFQ protocol of Section 8; the key differences is that Alice can now cancel an order once it is placed. The security model and timelocks for this protocol are identical to those in Section 8.2.

9.1 Off-blockchain Limit Order trades.

Limit Order. To place the limit order, Alice specifies the amount and the limit price. For example, "I will sell 200 LTC at the price of 2 BTC". The limit order remains open until either (1) the order is executed by the exchange, or (2) the order is cancelled by Alice. To place the limit order, Alice and exchange execute the "Setup", "Request", "Quote" and "Order" steps of the bidirectional RFQ protocol in Section 8.3.

Execute Limit Order. To execute the order, the exchange performs the "Execute" step of the bidirectional RFQ protocol in Section 8.3. This fills the order at the limit price for the specified amount.

Cancel Limit Order. Alice can cancel the limit order at any time. Cancelling is cooperative—Alice cannot prevent the exchange from refusing to participate in the protocol specified below. However, if the exchange does complete the protocol, the order cannot be executed even if the exchange becomes unresponsive or malicious. The protocol is as follows.

1. The user chooses a random secret cancel value $C^{\dagger}_{\mathcal{U}}$, computes $j^{\dagger}_{\mathcal{E}} = H(C^{\dagger}_{\mathcal{U}})$, and sends $j^{\dagger}_{\mathcal{U}}$ to the exchange. The exchange chooses a two random cancel values $C^{\dagger}_{\mathcal{E},\mathcal{A}}$ and $C^{\dagger}_{\mathcal{E},\mathcal{A}}$, computes $j^{\dagger}_{\mathcal{U},\mathcal{A}} = H(C^{\dagger}_{\mathcal{E},\mathcal{A}})$ and $j^{\dagger}_{\mathcal{U},\mathcal{B}} = H(C^{\dagger}_{\mathcal{E},\mathcal{B}})$, and sends $j^{\dagger}_{\mathcal{E},\mathcal{A}}$ and $j^{\dagger}_{\mathcal{E},\mathcal{B}}$ to the user.

2. Alice sends the exchange a postE cashout that overwrites the payE_postE puzzle that the exchange obtained from during the Limit Order step. This postE cashout (a) reflects the balance in the escrow before the limit order was placed. If Alice is selling coins from the user escrow, this postE cashout transaction (b) spends the output of the user escrow and (c) can be cancelled under $C^{\dagger}_{\mathcal{E},\mathcal{A}}$. Otherwise, it (b) spends the output of the exchange escrow and (c) can be cancelled under $C^{\dagger}_{\mathcal{E},\mathcal{B}}$.

3. The exchange sends Alice a postU cashout transaction for the user escrow and a postU cashout transaction for the exchange escrow. Both of these cashout transactions can be cancelled under $C^{\dagger}_{\mathcal{E}}$. These postU cashouts reflects the balance in the escrow before the limit order was placed.

   The exchange also sends the cancel value for the payE_postE puzzle transaction—namely, $C_{\mathcal{E},\mathcal{A}}$ if Alice is selling coins from the user escrow, and $C_{\mathcal{E},\mathcal{B}}$ otherwise. At this point the Limit order is canceled, since the exchange can no longer claim coins from the PayEPostE puzzle. The remaining steps in this cancel protocol return the escrows to the open state; if one party aborts, the counterparty must freeze and then close the escrows.

4. Alice sends the exchange $C_{\mathcal{U}}$, cancelling the payE_postU and payU_postU from the Limit Order.

   Alice sends the exchange a postE cashout that overwrites the payU_postE puzzle transaction that the exchange obtained as part of the Limit Order. This cashout (a) reflects the balance in the escrow before the limit order was placed. If Alice is

buying from the user escrow, this postE cashout transaction (b) spends the output of the user escrow and (c) can be cancelled under $C^{\dagger}_{\mathcal{E},\mathcal{A}}$. Otherwise, it (b) spends the output of the exchange escrow and (b) can be cancelled under $C^{\dagger}_{\mathcal{E},\mathcal{B}}$.

5. The exchange sends the cancel value for the payU_postE puzzle transaction from the limit order—namely, $C_{\mathcal{E},\mathcal{A}}$ if Alice is buying coins from the user escrow, and $C_{\mathcal{E},\mathcal{B}}$ otherwise.

10. RELATED WORK

Atomic swap protocols. The first description of an atomic swap is commonly attributed to TierNolan's 2013 forum post [34]. A number of works have since explored on-blockchain atomic swaps for various purposes, beyond payments [32, 29, 11, 27, 20, 22, 3], including improved fungability [21, 15], trading across blockchains [5, 4] and forks [23] or between ERC-20 tokens on the Ethereum blockchain [28].

Layer-two protocols. A layer-two blockchain protocol [25] binds off-blockchain transfers of funds to an on-blockchain smart contract. Layer-two protocols typically 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 [32, 29, 11, 15, 27, 20], 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 5.3. Meanwhile, bidirectional payment channels for Bitcoin payments were first proposed by [11, 29], and significant progress has been on made on the Lightning Network [2]. Today's Lightning Network requires SegWit (Section 7), and thus only supports Bitcoin and Litecoin, while Arwen supports more Bitcoin-derived coins, including BCH, ZEC. Finally, while [32, 29, 11, 15] all rely on HTLC smart contracts (*i.e.,* that use puzzle y with solution x such that $y = H(x)$), [27, 20] show how to build layer-two protocols "scriptlessly", without smart contracts, by cleverly leveraging the digital signatures on transactions.

Smart contracts on Ethereum are Turing-complete, and thus support a dramatically richer set of operations than smart contacts written in Bitcoin Script. (Bitcoin script is similar to Assembly language, with a few cryptographic operators.) Thus, it is no surprise that Ethereum also supports unidirectional and bidirectional channels ("state channels") [22], as well a payment channel network called Raiden [3]. Plasma [28] is a proposal for a layer-two decentralized exchange protocol on Ethereum. Truebit is another fascinating approach, where computations (rather than payments) are moved off the Ethereum blockchain via a layer-two protocol [33]. Generally speaking [22, 3, 28, 33] are for Ethereum and ERC-20 tokens only, and so they leverage and adapt to the full 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.

BOLT [14] is a layer two protocol with very strong privacy guarantees designed to extend the privacy properties of Zcash to a layer-two payment channel.

Because Arwen is focused on the cross-blockchain trading use case,Arwen must support as many coins as possible. For this reason, the Arwen protocols are designed around the "lowest common denominator" of Bitcoin-derived coins. For this reason we do not assume SegWit support.

Fees. Layer-two protocols, that are focused on payments, typically structure incentives around transaction fees, *i.e.,* fees earned when payments are made. This does not solve the problem of lockup griefing (Section 2.5), because no fees are earned if no payments are made. Arwen solves the lockup griefing problem via escrow fees and reputation.

Komodo [5] also looks to solve the lockup griefing problem for on-blockchain atomic swaps between peers. The Komodo approach has Alice first pay Bob an on-blockchain fee before the swap begins, while Bob must deposit coins as a collateral during the swap, which will be returned to Bob once the swap completes. Komodo fees are less efficient because they are paid on-blockchain; Arwen escrow fees are paid off-blockchain. 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.

Sparkswap [4] is a peer-to-peer trading platform for BTC and LTC built on top of Lightning. Sparkswap uses "deposit fees" to address incentives in its trading instruments (as opposed to premium-free American call options, see Section 2.6), but lacks a solution to the lockup griefing problem (Section 2.5).

11. CONCLUSION

Arwen is a layer-two blockchain protocol that allows traders to benefit from the liquidity at a centralized exchange without trusting the exchange with custody of their coins. Instead, Arwen trades are backed by on-blockchain escrows, and executed via fast off-blockchain atomic swaps. Arwen solves many of the incentive issues that emerge when payment protocols are repurposed for cryptocurrency trading. Arwen also supports a wider spectrum of coins, including Bitcoin, "Bitcoin fork" coins that do not have SegWit support (Bitcoin Cash, Zcash, *etc.*), Ethereum, and ERC-20 tokens.

12. ACKNOWLEDGEMENTS

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

13. REFERENCES

[1] Etherdelta. https://etherdelta.com/.
[2] Lightning daemon (lnd). https://github.com/lightningnetwork/lnd.
[3] Raiden network. https://raiden.network/.

[4] Sparkswap. https://sparkswap.com/.

[5] Komodo. *White paper*, June 3 2018.

[6] Clare Baldwin. Bitcoin worth $72 million stolen from bitfinex exchange in hong kong. *Reuters*, August 3 2016.

[7] 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.

[8] Bloomberg. Coincheck hack: How to steal $500 million in cryptocurrency http://fortune.com/2018/01/31/coincheck-hack-how/. *Fortune*, January 31 2018.

[9] Mark Friedenbach BtcDrak and Eric Lombrozo. Bip 0112: Checksequenceverify (2015). *URl: https:// github.com/ bitcoin/ bips/ blob/ master/ bip-0112.mediawiki(visitedon2019-01-27)*, 2015.

[10] Vitalik Buterin. Bitfloor hacked, $250,000 missing https://bitcoinmagazine.com/articles/bitfloor-hacked-250000-missing-1346821046/. *Bitcoin Magazine*, September 5 2012.

[11] 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.

[12] Ian DeMartino. Mintpal hacked 'considerable amount' of vericoin stolen. *Cointelegraph*, Jul 13 2014.

[13] 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.

[14] 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.

[15] 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.

[16] Stan Higgins. Details of 5 million bitstamp hack revealed https://www.coindesk.com/unconfirmed-report-5-million-bitstamp-bitcoin-exchange/. *Coindesk*, July 1 2015.

[17] Stan Higgins. Cryptsy threatens bankruptcy, claims millions lost in bitcoin heist. *Coindesk*, Jan 15 2016.

[18] Stan Higgins. Bitcoin exchange youbit to declare bankruptcy after hack. *Coindesk*, Dec 19 2017.

[19] Kraken. How long do digital assets/cryptocurrency deposits take? *URl: https:// support.kraken.com/ hc/ en-us/ articles/ 203325283-How-long-do-digital-assets-cryptocurrency-deposits-take-(visitedon2019-01-28)*.

[20] 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.

[21] Gregory Maxwell. Coinswap: Transaction graph disjoint trustless trading. *https:// bitcointalk.org/ index.php?topic=321228*, 2013.

[22] Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018.

[23] 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.

[24] David Z. Morris. Bitgrail cryptocurrency exchange claims $195 million lost to hackers http://fortune.com/2018/02/11/bitgrail-cryptocurrency-claims-hack/. *Coindesk*, February 11 2018.

[25] Neha Narula. The importance of layer 2. https://medium.com/mit-media-lab-digital-currency-initiative/the-importance-of-layer-2-105189f72102, May 27 2018.

[26] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999.

[27] Andrew Poelstra. Scriptless scripts. *https:// download.wpsoftware.net/ bitcoin/ wizardry/ mw-slides/ 2018-05-18-l2/ slides.pdf*, 2018.

[28] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[29] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5*, 9:14, 2016.

[30] Pete Rizzo. Poloniex loses 12.3% of its bitcoins in latest bitcoin exchange hack https://www.coindesk.com/poloniex-loses-12-3-bitcoins-latest-bitcoin-exchange-hack/. *Coindesk*, March 5 2014.

[31] Jon Russell. Korean crypto exchange coinrail loses over $40m in tokens following a hack. *Techcrunch*, June 10 2018.

[32] Jeremy Spilman. [bitcoin-development] anti dos for tx replacement. https://en.bitcoin.it/wiki/Contract#Example_7:_Rapidly-adjusted_.28micro.29payments_to_a_pre-determined_party, April 20 2013.

[33] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *URL: https://people. cs. uchicago. edu/teutsch/papers/truebit pdf*, 2017.

[34] TierNolan. Re: Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, May 21 2013.

[35] Peter Todd. Bip 0065: Op_checklocktimeverify (2014). *URl: https:// github.com/ bitcoin/ bips/ blob/ master/ bip-0065.mediawiki(visitedon2019-01-27)*, 2016.

[36] Fabian Vogelsteller and Vitalik Buterin. Erc 20 token standard. *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland*, 2015.

[37] Will Warren. Front-running, griefing and the perils of virtual settlement (part 1) https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-1-8554ab283e97, December 22 2017.

[38] Wikipedia. Mt. gox. https://en.wikipedia.org/wiki/Mt._Gox.

[39] Pieter Wuille. Segregated witness and its impact on scalability https://www.youtube.com/watch?v=NOYNZB5BCHM, December 14 2015.

[40] Wolfie Zhao. Crypto exchange zaif hacked in $60 million bitcoin theft. *Coindesk*, Sept 20 2018.