The Lightning Network Cross-Chain Exchange: A Decentralized Approach for Peer to Peer Exchange Across Blockchains

by

Jesús Andrés Mathus Garza

B.S. Computer Science
Massachusetts Institute of Technology, 2017

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2018

Author: _____

Department of Electrical Engineering and Computer Science
June 8, 2018

Certified by: _____

Thaddeus Dryja
MIT Digital Currency Initiative Research Scientist
Thesis Supervisor

Certified by: _____

Jonathan Ramsay Key
Charles Stark Draper Laboratory
Thesis Supervisor

Accepted by: _____

Katrina LaCurts
Chairman, Department Committee for Graduate Students

This page intentionally left blank.

The Lightning Network Cross-Chain Exchange: A Decentralized Approach for Peer to Peer Exchange Across Blockchains

by

Jesús Andrés Mathus Garza

Submitted to the Department of Electrical Engineering and Computer Science on May 25, 2018 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Abstract

The development of decentralized blockchain-based systems has unlocked opportunity in untrusted systems. As more blockchains were created, however, a lack of interoperability became apparent. In response, centralized exchanges facilitating transactions across different blockchains emerged, reintroducing trusted third parties that blockchains were in part created to eliminate. Although blockchain capabilities were promising, their emergence resulted in embezzlement, hacks, and scandals that resulted in significant financial losses. A program allowing for peer to peer cross-chain exchanges would reestablish the decentralized foundation upon which blockchains were built and eliminate the risks associated with centralized exchanges. In this work we extend Lightning Network capabilities and develop a protocol enabling secure peer to peer channels to safely transact across blockchains. The system connects individuals using the Lightning Network's channel creation functions, and introduces four new channel commands: Price, Compare, Exchange, and Respond. Together, they integrate hashed timelock contracts that introduce peer to peer negotiations and exchange functionality from one blockchain to another. With this functionality in place, individuals gain more control over their own assets and rely less on third parties, reaffirming decentralization throughout the blockchain ecosystem and laying a new foundation for distributed systems to interact with less friction.

Thesis Supervisor: Thaddeus Dryja
Title: MIT Digital Currency Initiative Research Scientist

Thesis Supervisor: Jonathan Ramsay Key
Title: Draper Cloud Architecture Group Leader

This page intentionally left blank.

Acknowledgements

I would like to thank Tadge Dryja and Ramsay Key, my thesis supervisors, for their continuous help and guidance throughout this project. Each of you provided tremendous support and I truly appreciate the time and effort you contributed.

This page intentionally left blank.

Table of Contents

This page intentionally left blank.

List of Figures

This page intentionally left blank.

1 Introduction

With the world progressing towards an ever more digitized state, it is no surprise that our currencies have followed suit. The transition that banks made from a purely cash based system into an electronic banking system through the use of credit cards served as a stepping stone limited only by the ability to fully secure an online currency system. Satoshi Nakamoto's innovative development of Bitcoin[18], the first blockchain based cryptocurrency, granted society the ability to shed the need for trusted third parties and decentralize the financial system. Moreover, Nakamoto's contribution serves as another step forward into the full potential of distributed systems. Although Nakamoto's initial application was financial, the technology that Nakamoto created has already been extrapolated into many other sectors, such as data management, security, supply chain management, and more. Like most other widespread technologies, blockchains need to develop an interoperability protocol to fully leverage the untapped strength of the broader blockchain network. To further develop this decentralized network, we will construct a system for peer to peer cross-chain transactions.

1.1 Background

Our work builds on the foundation of blockchains, cryptocurrencies, currency exchanges, and blockchain scalability (in particular the Lightning Network) set up. These building blocks have expanded possibilities in fields such as international trade[20] and distributed systems[16], and could have greater impact if implemented together.

1.1.1 Blockchain

We define a blockchain as a decentralized ledger comprised of token creation and state changes. A token is each blockchain's operational unit, used to transact and interact on the network, and a state refers to the present standing of all users on the network based on the documented network history. A complete copy of this digital representation is distributed across all nodes, or devices belonging to the blockchain network. Although this generalized high level structure applies to most blockchains, the specific implementation details differ from blockchain to blockchain, mostly in the way that each blockchain network reaches correct current state agreement.

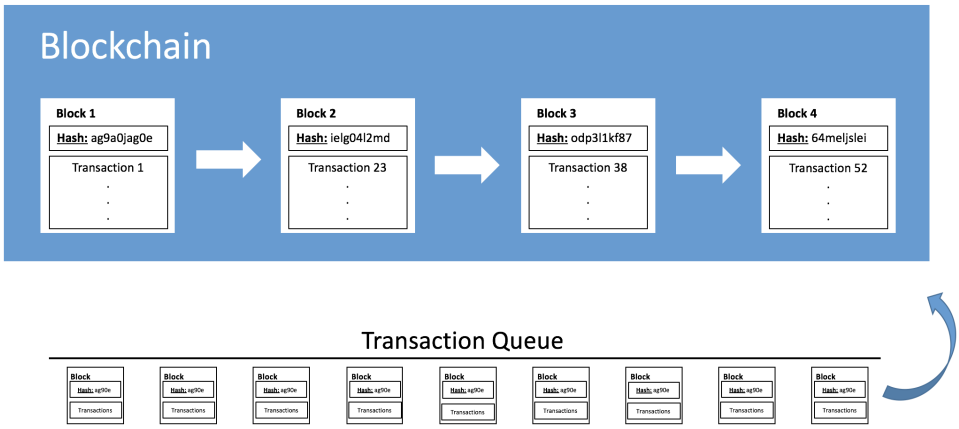

Figure 1: Blockchain representation.

For example, the Bitcoin blockchain utilizes a proof of work scheme for its transactions confirmation. Proof of work is a method that leverages the computing power of users to update the Bitcoin network, cataloging new transactions and effectively reaching new state agreement across the network.. It begins when a token is transferred from one user to another. That specific transaction is communicated to the broader network as an unconfirmed transaction, meaning that users must confirm the transaction

to append it onto the blockchain as a permanently accepted block. As seen in Figure 1, blocks are made up of multiple transactions, and are collected in a queue until they are confirmed and appended to the blockchain. Users that work to process any and all transactions are referred to as miners. Miners dedicate their computing power to hashing[1] a value in unconfirmed blocks known as the nonce[2]. When a miner calculates a hash with a value that is less than a network wide constant, which the system adjusts every 2016 blocks (as defined by the network based on the network wide confirmation rate of about 10 minutes), the miner's block is appended to the blockchain, the block is broadcast to all nodes on the network, and other miners can now start building on this new block (as shown in Figure 2). To append to the decentralized network, the miner sends other nodes a confirmation signal, notifying them of the new block, and the network uses a gossip-like communication protocol to distribute the message throughout the network. However, if the miner produces a hash that is not less than the universal constant, the miner will continue changing the nonce, hash the new nonce value, and iterate until they fulfill the value requirement[18].

Every block also has a blockheader, containing metadata about that block. Block headers are not sent across the network, but are instead computed by each node in order to validate the block's contents. One of the metadata fields is the previous block's hash. Because nodes calculate the previous block's hash for themselves, and because hash values are dependent on the input's exact contents, an adversary cannot swap an existing

[1]Hashing refers to the one way mathematical transformation of data from its original form into a new fixed length hash value. The function makes the hash value easy to compute given the original form, but makes the original form nearly impossible to compute given the hash value. Hashing is usually used for security purposes, but in this case is only used as a complexity device to ensure the proof of work scheme is upheld.

[2]The nonce is a system generated random value that exists as a field in each block. Miners edit the nonce value to change the hash of the block, and to continue mining.

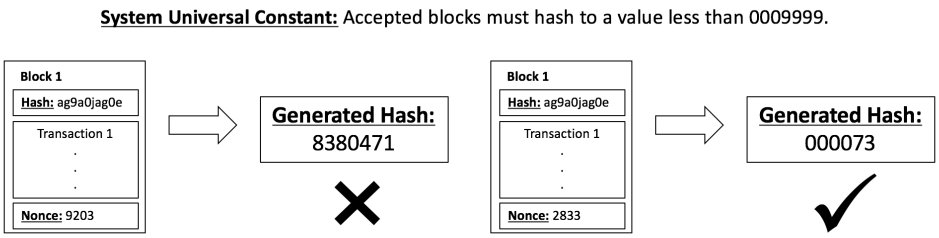

Figure 2: An example of a miner hashing a block.

block with their own. If an adversary tried to do so, the corresponding hashes on all blocks that come after the adversary's swapped block would not match the hash created from the new malicious block.

Iterating over this transaction confirmation scheme will result in the creation of a blockchain, which stores the history of transactions for a token as long as a single node exists in its network.

With decentralization at its core, blockchains thrive as user driven entities that solve some of the challenges of centralization: security, privacy, and government mismanagement. Blockchains avoid the security risks that centralization implicitly holds, as adversaries have no central system to target. If an adversaries targets one node, or even hundreds of nodes, the broader system can remain resilient and unaffected. Miners themselves verify the accuracy of the chain they are mining, and can choose to ignore any chain that has been maliciously affected. This serves as a valuable fault tolerance mechanism that helps provide system integrity. Blockchains also provide a high level of privacy. User wallets and transactions are only recognizable through their obscure number identifiers, which hold little to no connection to any real world identifiers. Even if an adversary tracked user activity and recognized a user's activity pattern, they would have no way to prove that the network activity belongs to the real world user. The

technology is also independent of the government or any governing body, which prevents any government mismanagement (at least as long as the government does not target the users themselves) and censorship from limiting the technology's capabilities. These qualities have helped blockchains, and their cryptocurrency tokens, thrive over the past few years, and leaves many hoping that they will develop further and potentially replace some of our current financial and technological systems.

1.1.2 Cryptocurrencies

A cryptocurrency, as implemented on a blockchain, is a unit that can be traded from peer to peer and draws value directly from consumer demand, which in turn stems mostly from the novelty of implementation or functional capabilities. Each token can have different use cases because of the different ways that the underlying blockchain technology can be implemented. For example, Bitcoin has dominated cryptocurrencies as a financial tool[18], but Ethereum has added functionality that better suits it for the creation of smart contracts[3] and blockchain based applications[4].

In addition, cryptocurrencies are novel in that they establish a system where every user gains tangible benefit from expanding the overall user base. As more users join the network, the value of the network increases, deriving its value solely from supply and demand. This encourages participants to grow the network. However, network growth produces other issues that results in one of the most important challenges cryptocurrencies face at the moment: scalability.

The number of Bitcoin transactions has increased dramatically over the past few years, going from around 100,000 per day in early 2015 to a peak of

[3]Smart contracts on the Ethereum blockchain are user uploaded code that serves to digitally enforce user defined agreements.

over 400,000 per day at the end of 2017[6]. As the user base expands, miners receive more transactions to process. However, the average confirmation time remains constant at about 10 minutes for Bitcoin, making it harder for miners to confirm transactions promptly. Users have adjusted the system architecture to increase the maximum number of transactions allowed in a block, but cannot increase it infinitely. The block size must remain within a reasonable size to allow users to easily download, process, and store all transactions. Although changing the system's block size helps, for Bitcoin or other cryptocurrencies to be able to handle a number of transactions at a mainstream scale, more has to be done.

Similarly, the observed Bitcoin transaction fees have increased, going from pennies at then end of 2016, to a peak of over $50 at the end of 2017[5]. With the increase in transaction volumes, users have made use of Bitcoin's transaction fee mechanism. Anyone initiating a transaction can also add a fee for miners. The mechanism exists so that the network can continue to function once the maximum Bitcoin cap is reached (as programmed into the system), but users have begun including fees to incentivize miners to prioritize their transactions amidst higher transaction volumes. Miners then naturally prioritize the transactions they include in their blocks based on the fee per byte ratio of each transaction. With that being said though, transaction fees did drop back down to a few dollars[5], but the volatility is still something that needs to be considered for Bitcoin and cryptocurrencies in general.

With increased transaction volumes and strong fee volatility, the possibility of mircrotransactions[4] on the Bitcoin network has been reduced. For example, if you go to Starbucks to buy your morning coffee, you would

[4]Transactions involving small quantities of money or value.

never choose to use Bitcoin over, say, a credit card, if Bitcoin requires you to pay a $5 fee at some point when your coffee costs $4.50. Many believed that Bitcoin would remain usable for microtransactions, but the issues of scalability and increasing transaction fees has become a contested and controversial issue that the Bitcoin system must work through.

1.1.3 Currency Exchanges

The currency exchange market, otherwise known as the foreign exchange market, is the world's largest market trading over $2 trillion worth of value daily[21]. From its inception, the foreign exchange market broke down barriers to entry between distinct markets and allowed for the global market to expand and take proper form. Efficient international trading and global monetary policy developed, adding value for nations, organizations, and even individuals participating in the global economy[15]. One can think of exchanging currencies as gaining the ability to tap into the specialties that other economic entities possess.

For example, Saudi Arabia and China have a strong exchange of imports and exports based on each of their specialties and needs[9]. Saudi Arabia invested billions of dollars in securing itself as China's primary source of petroleum, having a strong production; likewise, China symbiotically invested billions in exporting metals for Saudi Arabia[17]. Both invest in each other because they had unique specialties that would cost more to reproduce than to tap into through exchange.

Currency exchange makes possible the efficient exchange of goods and services across different economical structures. Imagining a world where blockchain extends to applications and targeted use cases, the inter operation between large-chains and small-chains with specialized capabilities

could serve to eliminate friction from the blockchain ecosystem and unlock value that will benefit all users. Hypothetically, if Bitcoin becomes the most stable and reliable cryptocurrency, and Ethereum becomes a widely used app development platform, exchanging one for the other could promise more financial security or allow an individual to participate in some useful application. The impact exchanges have had on countries is on a completely different scale, but the blockchain field is young and growing, and the full impact that efficient exchange could have on the blockchain field could grow as well.

1.1.4 The Foundation: Lightning Network

The Lightning Network aims to solve Bitcoin's scalability limitations. It is a system of micropayment channels that allow trusted off-chain[5] transactions using a contract/timelock system[24]. Instead of overwhelming blockchains with every single transaction made, users would only broadcast two transactions per created channel: a funding transactoin and a closing transaction. This channel life cycle can be seen in Figure 3 below. The funding transaction represents the creation of a Lightning Network channel, broadcasting to the blockchain that a user is taking out 8 BTC and putting them into a channel. There, that user and their peer can transfer tokens with each other continuously without touching the main blockchain. Once they finish transacting, they broadcast a closing transaction on the blockchain, which places their new token amounts back into their main blockchain balances. All users can create these channels with one another and are able to transact virtually without limit and with minimal trans-

[5]Blockchain transactions that occur off of the blockchain, meaning they are not communicated to the network when they are executed. This implies that there is some trust mechanism in place that ensures that when they are communicated to the blockchain, their integrity can still be guaranteed.

action fees (only the fees from the funding and closing transactions, and from transactions using channel hopping, as covered in Section 6.1.5).

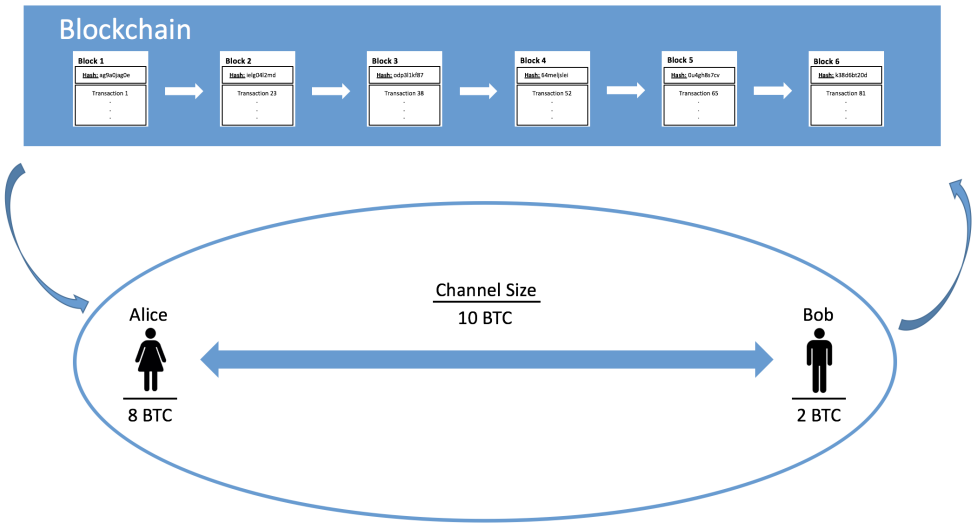

Figure 3: Lightning Network channel life cycle.

State oriented channels are the main tool the network utilizes to replicate blockchain's state defined system, accurately and securely tracking validated events over time. To create one of these channels, two users must first connect with each other. Lets say that Alice and Bob are friends that want to transact on a Lightning Network channel.

Alice and Bob first establish and authenticated data connection. Once the two peers are connected, they can form a transaction channel and finally exchange tokens.

Upon creation, the channel establishes a starting state. The state serves as a safety net, guaranteeing each channel member the ability to close the channel and claim their current balance at any time. In the channel creation process, the Alice's node sends a message to Bob's node to corroborate the existence of the channel between them, and corroborate the value that Alice has pledged to deposit in the channel. Bob's node will sign off on the state

proposal, and once both users agree and sign off, the state will be officially updated and the integrity of the system will be maintained. All future transactions within the channel involve a state change agreement process, state signing process, and exchange of private keys to update the channel's state[24].

When a state is updated, the previous state must be revoked to prevent Bob, for example, from broadcasting an old state where he had more tokens, even though he has spent some tokens since. To revoke an old state, both participants send a message to each other with a private key for that specific state. With this private key, a user can prove that both parties aggreed to revoke that old state. So if a malicious participant attempts to broadcast the old and revoked state, the cooperative participant can use the malicious participant's private key for that revoked state to nullify their broadcast, and penalize them, taking all of their channel tokens.

As mentioned above, the Lightning Network nodes use messages to transfer information from commands. We can see how the message protocol works in our Lightning Network implementation's Push command.

The Push command achieves a state change sending tokens from the command caller to their peer using three messages: a DeltaSig message, a SigRev message, and a Rev message. The DeltaSig message transfers a delta and a signature from the Push command initiator. The initiator is sending tokens, and communicates that through a delta, or amount being sent (negative for the initiator and positive for the receiver), and a new signed state reflecting the balance differences. The SigRev message involves the receiver endorsing and singing the initiators proposed state and revoking the previous state. Lastly, the Rev message has the initiator revoke the previous state as well, convincing the receiver that the transfer

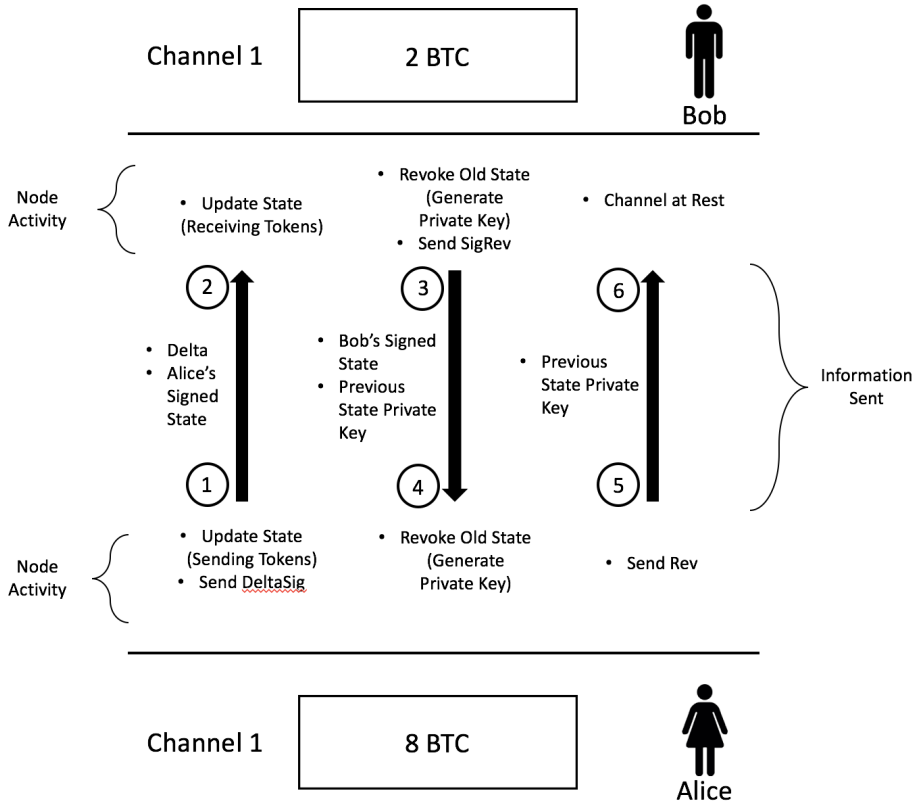

Figure 4: Lightning Network Push command message exchange visualization.

is complete, and achieving agreement on the new channel state.

For example, if Alice and Bob have a channel where they have 8 BTC and 2 BTC respectively, and Alice would like to pay Bob for buying lunch earlier in the day, she might Push 100,000 Satoshi (or 0.001 BTC). Alice inputs this to her machine, and her machine builds a transaction with the updated channel values (meaning she officially declares the outputs and inputs, being -100,000 Satoshi for her and 100,000 Satoshi for Bob respectively), signs the proposed transaction representing the new state, and sends a DeltaSig message to Bob. The message contains the transfer information (a delta of 100,000 for Bob, and the new state signed by Alice) in byte representation, as seen on Figure 4. Bob's machine receives

and interprets the message, performing the same update steps with the given delta. His machine updates channel values, signs the same proposed transaction, and revokes the previous state. This revocation is what we mentioned earlier, where Alice receives a private key from Bob, which she can use to claim all the money from the transaction's outputs if Bob does publish that specific revoked state. Bob sends the private key and the new signed state in a SigRev message to Alice. With the private key, Alice knows that Bob's previous state is revoked, and she sends the last message in the process, the Rev message, providing him with a private key so that he can consider Alice's previous state revoked as well[24]. After all of this, Alice will now have 7.999 Bitcoin, and Bob will have 2.001.

Once Alice and Bob agree that they want to conclude interactions on their channel, they call the Close command and terminate the channel cooperatively, allowing both users to use their tokens immediately. However, if either Alice or Bob wishes to close the channel, but does not want to wait for the other's cooperation, they can call the Break command, forcing the channel closer to wait some period of time before being able to use the money returned from the channel. The other user, however, will still be able to use their money immediately. The Break command ensures user autonomy in case one user becomes unresponsive or uncooperative, allowing the other user to close the channel anyway[24].

In Section 5 we use the Lightning Network's secure, user-generated platform to develop decentralized capabilities for cross-chain transactions.

This page intentionally left blank.

2 Related Work

Although blockchain technology is still in its infancy, its rise in popularity academically, professionally, and across social groups has pushed developers to keep moving the field forward at a rapid pace. With that in mind, most developers knew that they would have to be first to market if they hoped to lead the field in any particular endeavor. This has streamlined the development of new systems within the broader field of blockchain, but the rush brings with it implementations with a narrower range of application. Atomic swaps[6] serve as an example as early development only applied to a handful of cryptocurrencies, but has seen significant step by step progress recently.

On September 19, 2017, the Decred Project sucessfully conducted the first on-chain atomic swap between Decred and Litecoin. This was a very powerful step forward in the cryptocurrency community as the team proved that exchanging different cryptocurrencies without a trusted third party was possible, but there was still work to be done as their implementation fell under limiting specifications. Their open-source code now encouraged developers to refine the approach and create more implementations that could one day transact across blockchains more universally[26].

Shortly after, on October 7, 2017, Altcoin.io did just that, and managed to complete the first atomic swap between ethereum and bitcoin, two tokens on different blockchains[11]. This achievement marked another step forward in decentralizing cross-chain transactions, but the process was still on-chain, meaning that any swap would incur mining and congestion in the system, especially at scale. The blockchain ecosystem has to find a way to

[6] A method for peer-to-peer exchange of different cryptocurrencies without the need for a trusted third party

scale if it is ever going to fulfill financial needs worldwide, which amount to over $2 trillion in currency exchanged per day[22]. The answer to the issue of scaling: the Lightning Network.

Lastly, on November 16, 2017, Lightning Labs (a company working on another Lightning Network implementation named LND) successfully swapped coins across the Bitcoin and Litecoin networks[25]. At this point there are multiple Lightning Network implementations, and each has branched off in development, pursuing unique goals and system designs. Our team is the Massachusetts Institute of Technology Digital Currency Initiative, and we are working on a particular Lightning Network implementation named lit.

As all of these implementations have built off of each other, we too look to push the field forward using the inspiration of those that have pushed us this far up until now. Our implementation is another independent effort to expand blockchain capabilities and contribute to its future.

This page intentionally left blank.

3 The Problem

The core problem we want to address is decentralized cross-chain interactions at scale. All prior approaches to scaling cross-chain interactions involve a centralized exchange. Crypto to crypto exchanges[7] have been established to fulfill cross-chain transaction needs, but add unnecessary risk to the blockchain ecosystem. They centralize the ecosystem through token custody, they exist solely as financial tools, and they have a history of being exploited. Fundamentally, blockchain exchanges undermine the very purpose for which blockchain technology was initially created: to eliminate superfluous third parties[18].

3.1 Centralization Defeats the Purpose of Blockchains

Satoshi Nakamoto states that the reason he or she created blockchain technology is for the purpose of eliminating third party institutions' involvement in financial transactions[18]. Nakamoto developed this system to empower individuals, preventing these third parties from reaping the benefits of establishing trust. The mechanism that blockchains use to accomplish this stems from consensus through decentralization.

Other than the advantage of enabling global digital payment systems, decentralization as a concept adds important value to existing infrastructures. Smart contract development extends transaction capabilities past mere currency exchange to the verifiable exchange of ownership over any product, good, or service[4]. This advanced capability results in the reduction of lawyers or similar third party systems required to verify any type of legal action individuals or organization take, showing promising impact[2].

[7]Exchanges that trade one cryptocurrency for another cryptocurrency.

Blockchain's decentralization also gives rise to a new type of distributed database. As explained earlier, blockchains' system design propagates all data input on the system to each one of its nodes. As distributed databases, blockchains provide positive security capabilities (as described in Section 1.1.1). Their system architecture allows individuals and organizations to effectively distribute data across thousands of endpoints and count on the system's resilience. For example, Follow My Vote is an organization developing secure blockchain voting protocols to collect and publicize voting data for a transparent and corruption free voting system[7].

Established exchanges centralize a decentralized system, reintroducing the limitations and problems that blockchains solve and limiting the potential they offer.

3.2 Lack of Non-Currency Cross-Chain Connectivity

As a disruptive technology, blockchain's purpose has been extrapolated beyond currency exchange. Industries extended blockchain to fields such as supply chain management[14], health care[3], and more as developers find more ways to innovate using this new technology. Ethereum, for example, has cultivated a strong environment for non-monetary blockchain applications[10]. Expanding past the singular purpose of currency management, blockchains will demand a communication protocol that operates without financial backing. When the Bitcoin and Ethereum blockchains interact, there is a transfer of monetary value from one chain to the other, but when supply chain management blockchains interact, there should be a way for those blockchains to transact and exchange their value.

Cryptocurrency exchanges are built to transfer value between monetary blockchains, but have no infrastructure to accommodate transferring infor-

mation from one chain to another. Inherently, this lack of connectivity is not an issue, but cross-chain communication should be established to push the blockchain environment forward and improve the broader blockchain network. Compare the impact of a single website to that of the network of inter-connected websites; the first is limited in reach and scope, but the latter redefined the way we do business, how we communicate, and even our day to day lives. Lack of non-currency cross-chain connectivity may not be a problem for cryptocurrency exchanges, but it is a path that could expand the future of blockchain technology.

3.3 Centralized Exchange Vulnerabilities

A more tangible risk factor is the inherent vulnerability of centralized exchanges. With a single system and single party managing up to millions of dollars, blockchains' cryptographic and decentralized security environment gets simplified down to the unregulated cybersecurity capabilities of said trusted party. In addition, the untraceability of public key identification on any blockchain network allows for real world identification obscurity. With this in mind, some exchange managers simply embezzled all the money entrusted to them.

The prime example for all of the above being Cryptsy. Cryptsy was a popular alternative cryptocurrency exchange in 2014 and early 2015. Run by Paul Vernon, the exchange went through months of issues that eventually resulted in the firm's collapse. It was later discovered that Vernon embezzled millions of dollars amid all of the issues leading up to the firm's collapse and that he destroyed evidence of his illegal actions[13].

This page intentionally left blank.

4 The Solution

Our proposed solution is a decentralized cross-chain communication protocol that allows peer-to-peer off-chain transactions leveraging the Lightning Network. Although we aim to eliminate all third parties, including exchanges, we acknowledge that atomic swaps are more limited, dealing only with trade execution, not matching or discovery. However, the protocol combats centralization efforts along with the vulnerabilities that accompany them, and simultaneously establish a foundation for chains to communicate with other chains.

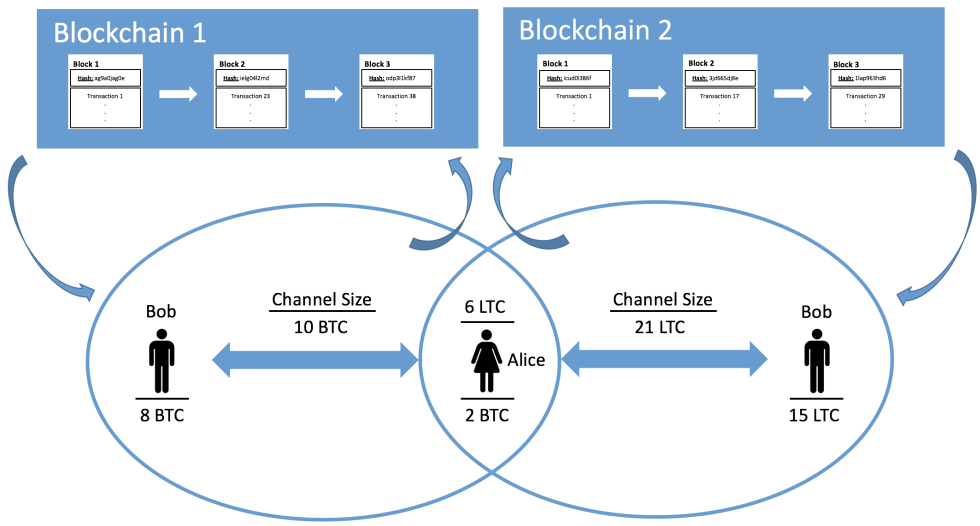

Figure 5: Lightning Network exchange where both Bobs represent the same person, exchanging differenct currencies with Alice.

As seen in Figure 5 above, the current exchange process would focus on a two channel peer to peer scenario where Bob wants to send Alice one type of token in exchange for a different token, with each belonging to a separate blockchain.

The program builds off of Lightning Network infrastructure, focusing on peer-to-peer transactions through user created channels. Nodes in the

network create channels connecting to other individuals on the network. From a channel's inception, each node determines an amount of blockchain value to set aside for the channel's duration. That total amount cannot be changed once the channel is created, but can be transferred between the two nodes indefinitely as long as the channel exists. Our newly developed program sets up the ability to transact across different blockchains, enabling users to trade Litecoin for Bitcoin and any other blockchain tokens that are made system compatible.

Finally, the program's distributed service minimizes the risk associated with vulnerable centralized exchanges using secure cryptographic checks and balances. Our program makes this possible through the implementation of hashed timelock contracts (HTLCs) across Lightning Network channels.

4.1 Hashed Timelock Contracts

If Alice and Bob have a Lightning Network channel open with Bitcoin, and have another channel open with Litecoin, and Alice wants to exchange 0.1 Bitcoin for 10 of Bob's Litecoins, they might find themselves in a stalemate. Even if Alice and Bob both agree on the exchange amounts, whoever sends their amount first has no way of guaranteeing that the other party will honor their agreement afterwards. If Bob were a malicious party, he could simply wait to receive Alice's 0.1 Bitcoin payment and never send her his 10 Litecoins. To fix this uncertainty, we implemented hashed timelock contracts.

HTLCs are vault mechanisms that uphold unilateral global state changes across multiple nodes using hashed secrets to unlock stored funds[24]. A secret, or preimage, is a randomly generated 20 byte array, which we hash

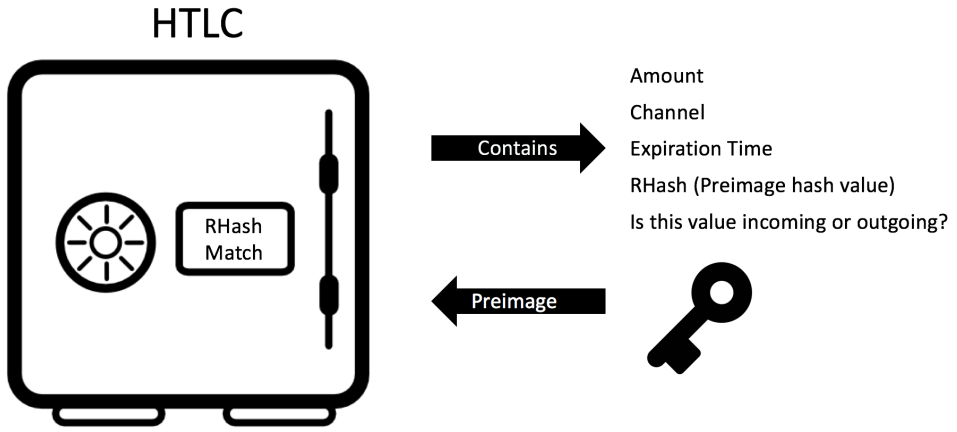

Figure 6: Our HTLC Golang struct representation.

to produce what we call an RHash, or the preimage's corresponding 20 byte hashed array. The RHash acts as the lock to the vault, while the preimage is the key. If we are given an RHash from a preimage that matches our HTLC's RHash, the HTLC unlocks, much like a key unlocking a door that has a matching lock. In effect, the open HTLC can now be accessed, and its contents become available to the user, as displayed in Figure 6. Most importantly, HTLCs are paired together for the purpose of atomic swaps, so that if one HTLC opens, the other does too. The link between HTLC pairs is what helps uphold their integrity for both parties invovled. In our example, Alice and Bob's transaction could be properly executed using HTLCs. Alice would store her 0.1 Bitcoin in an HTLC, and Bob would store his 10 Litecoin in another HTLC, both using the same RHash that Bob would be responsible for generating. The same RHash is used to eliminate the potential vulnerability where one party refuses to make their preimage public and prevents the other party from opening their corresponding HTLC. When either HTLC is opened, the preimage used to generate the correct RHash is published, guaranteeing access to the other

HTLC. Both HTLCs act as a commitment to transfer funds, so our system waits for both commitments to be created, at which point it executes the exchange. In this example, Bob opens the HTLC containing 0.1 Bitcoin, reveals the preimage to Alice, and transfers the 0.1 Bitcoin to himself, and Alice can then choose to open the HTLC containing 10 Litecoin using Bob's preimage and transfer that amount to herself, or to leave the Litecoin alone for Bob to reclaim upon expiration. It is important to note that Bob opening the HTLC with 0.1 Bitcoin then enables Alice to open the HTLC with 10 Litecoin, guaranteeing that both parties gain access to their promised tokens once one party does. The events occur in this order to prevent an initiator, in this case Alice, from attempting to subvert the system and prevent the transfer of funds to Bob after she has received her funds.

In addition, HTLCs are also blockchain compatible. If a channel were to close with existing HTLCs, the HTLCs would also be broadcast to the main chain, where they can eventually be resolved. Once an HTLC is on the main chain, the HTLC sender will have to wait for the the locktime to be reached to get their funds back upon expiration. The HTLC receiver, however, can use the preimage at any time to unlock the HTLC themselves and recover the funds as the receiver. If an HTLC is opened on chain, the preimage will still be publicized so that the other HTLC in the HTLC pair can then also be opened by its corresponding receiver.

With this mechanism in place, users are able to perform peer-to-peer cross-chain exchanges without a trusted third party, reestablishing blockchains' decentralized nature. Users are also able to control how much their tokens are worth, determining their valuations in their exchange agreements, and keep that value secure in their own hands. The Lightning Network cross-chain exchange expands the capabilities of the blockchain ecosystem and

creates growth potential through connectivity using hashed timelock contracts.

This page intentionally left blank.

5 System Overview

To create an effective currency exchange process, our system introduces four new Lightning Network commands: the Price command, the Compare command, the Exchange command, and the Respond command.

5.1 Price Command

The Price command is a simple channel command that allows users to look up the current market value (in USD) of any cryptocurrency available at www.coinranking.com. The command requires the user to provide a currency name. With a currency name, our system connects to the coinranking site, parses the HTML code for specific currency listing fields, and compares the given currency name against the listed currencies on the site. If the name exists, our program retrieves the listed, continuously updated market price, prints it on the user's terminal, and disconnects from the coinranking site.

5.2 Compare Command

The Compare command is another user oriented channel command that builds on the Price command and allows users to generate an exchange rate for any two cryptocurrencies available at www.coinranking.com. The command requires the user to provide two currency names. With both of those names, our system uses the Price command functionality to retrieve the current market value of each desired currency, and then perform some simple arithmetic to return the compared value of one unit of the first currency to the corresponding amount of units of the second currency (without fees included).

We developed both of these commands in hopes of enhancing the exchange experience for users on our platform. Our exchange protocol allows for users to decide their own exchange amounts. With the Price and Compare commands, we hope to empower our users further and provide information facilitating reasonable exchanges, improving the exchange atmosphere on the Lightning Network.

5.3 Exchange Command

The Exchange command is the first of the two commands we developed to perform cross-chain exchanges. The command requires the user to provide the channel with the first desired currency, an amount for the initiator to send on that channel, the channel with the second desired currency, and an amount for the acceptor to send on that channel. The parameters can be better understood in Figure 7. This command acts as a request. Continuing the example from the HTLC section, if Alice wants to exchange Bitcoin for Litecoin with Bob, she would input "exchange 1 10,000,000 2 1,000,000,000". This means that Alice would send her 0.1 Bitcoin (or 10,000,000 Satoshi) in channel one, and Bob would send his 10 Litecoin (or 1,000,000,000 Litoshi) in channel two. The coin types and value recipients are implied based on the input channels and the command parameter order. The first channel is implied to be the one where the initiator (Alice) will send funds, which she knows to be Bitcoin. The second channel is implied to be the one where the acceptor (Bob) will send funds, which Alice also knows to be Litecoin. This command acts simply as a request, so after Alice enters the command, her node relays the request information to Bob across the first channel that Alice specified.

The information is all converted to bytes, a preceding indicator is set to

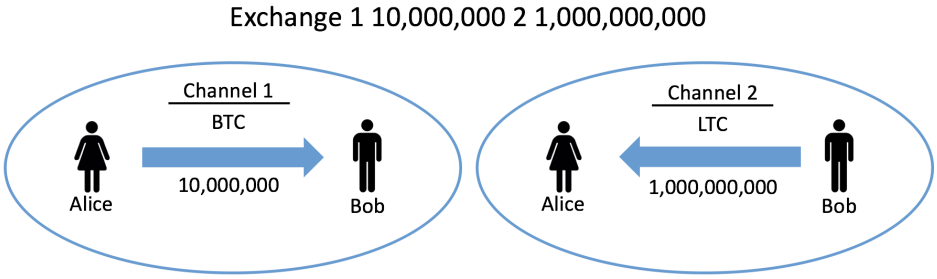

Figure 7: Lightning Network input "exchange 1 10,000,000 2 10" visual representation between Alice and Bob, with the Bitcoin value represented in Satoshi.

flag what function should handle the message once it is reconverted, and the message is finally sent from Alice to Bob. All nodes on the network have a programmed message listener[8], so Bob's Lightning Network node's listener function will detect the incoming message from Alice. The listener converts the message from bytes back into its original format, and uses the preceding indicator to hand off the information to the exchange request handler. The handler then prints the request information on Bob's Lightning Network terminal along with an expiration time and a randomly generated request ID.

The expiration time is set in our implementation to be one minute after the request is initiated (chosen to prevent sizable arbitrage opportunities). The request ID is generated from a random selection of 20 alphanumeric characters. At this point the fate of this transaction is in Bob's hands, but the last thing that the Exchange command is responsible for is monitoring the expiration time. If there were no expiration time, a malicious party could leave the request in place until the corresponding currencies fluctuated in value and the exchange would be greatly in the malicious party's

[8]Described further in the Appendix.

benefit. However, the command is programmed to automatically cancel the request and prevent Bob or any malicious party from abusing the exchange system when the expiration time is reached, and leaving Alice's balance unaffected.

5.4 Respond Command

The Respond command is a continuation of the Exchange command, and the command is the second of two commands we developed to perform cross-chain exchanges. At this point Bob has one minute to decide whether to accept Alice's exchange request. He can do one of three things: respond yes and accept the exchange, respond no and decline it, or let the request time out.

If Bob thinks the exchange is a fair one, as he does in Figure 8, he can input "respond YES 3" (with 3 being the request ID). The program checks that Bob's acceptance comes in before the expiration time of the request. If the response is too late, the program notifies Bob that the request has already expired. If the response comes within the expiration time, the program then takes all of the information that Alice gave in the exchange request and begins the token exchange process.

Exchange Respond Command Format

```
lit-af#
Exchange Request: 0.1 to you on channel 1 for 10 from you on channel 2
Request ID: 3 Expiration Time (UTC): 2018-06-02 15:04:05
To accept, use command: respond <YES> <requestID>
To decline, use command: respond <NO> <requestID>
lit-af# respond YES 3
Exchange accepted!
```

Figure 8: Exchange Respond command example where Bob decides to accept the exchange.

Once Bob inputs "respond YES 3", his machine handles the request information, using the initiator amount and acceptor amount to place the correct value in each HTLC, and the channel ID's to execute the node to node messaging, assigning and unlocking the HTLCs. From here on out, the protocol is broken up into three sections: HTLC creation, HTLC assignment, and HTLC unlocking.

5.4.1 HTLC Creation

We included two different methods for creating HTLCs, both taking place on the acceptor's machine. The first creates an independent HTLC, and the second creates a dependent one. All independent HTLCs will exist with a dependent HTLC pair, and vise versa. Both types have four fields unaffected by dependence: an integer value specifying the exchange amount, a boolean denoting whether that amount is incoming or outgoing from the perspective of the acceptor, a channel identifier to allow for more accurate HTLC assignment, and an expiration time as a fail safe to ensure that malicious parties cannot abuse any HTLCs (all shown in Figure 6). The expiration time also serves as a guarantee for the user pledging that they will reclaim their funds even if the system fails. The fifth and final field, however, is the HTLC RHash, acting as a lock. As stated in Section 4, the RHash is a hash value that comes from a randomly generated 20 byte preimage (or array), which acts as the key to open the corresponding HTLC. As shown in Figure 9, our independent HTLC creation scheme generates a random preimage, hashes it, assigns the hash to the created HTLC, and stores the preimage in the acceptor's machine's memory for use in the unlocking phase. In our dependent creation scheme, we use the preimage hash from the recently created independent HTLC to set the

dependent HTLC's RHash and establish the HTLC pair.

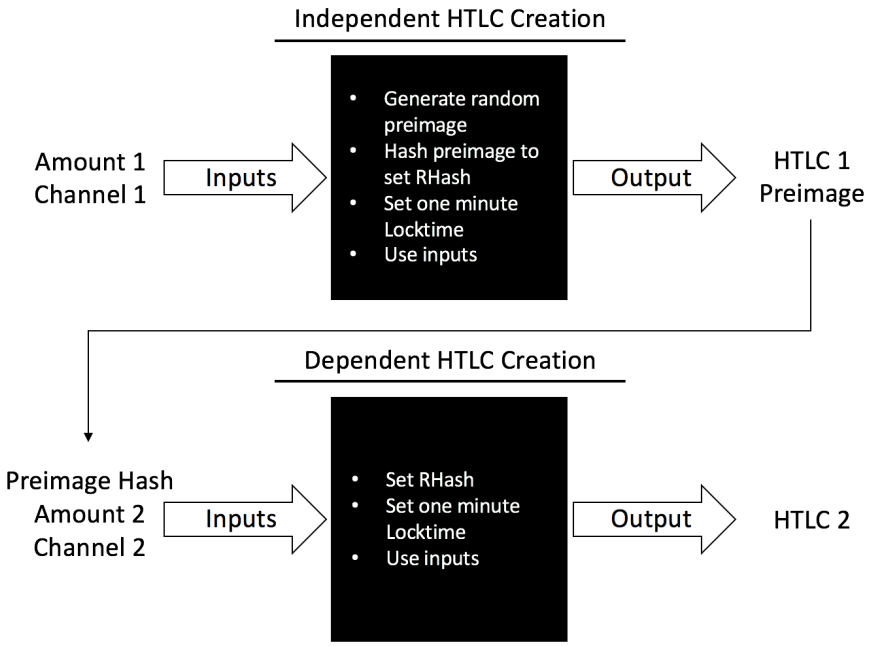

Figure 9: HTLC creation process for independent and dependent versions. Takes place on the acceptor's machine.

This is important for the exchange process because it creates a connection between the two HTLCs exchanging value across peers. That connection prevents one party from receiving funds without sending any in return, as stated previously. Going back to the Alice and Bob example, when Bob accepts Alice's exchange request, the system creates an independent HTLC promising Bob's 10 Litecoin to Alice, and then creates a dependent HTLC promising Alice's 0.1 Bitcoin to Bob. Both HTLCs have the same hash lock, which means that the same preimage key can open both as well. The same preimage and RHash are used because the unlocking and extraction of funds using a preimage simultaneously sends that preimage to the other party participating in the exchange, guaranteeing that they can also extract

their corresponding funds too.

5.4.2 HTLC Assignment

After HTLC creation, the acceptor continues the exchange process, verifying the accuracy of the creation process and including the movement of funds into HTLCs in the co-signed and agreed upon state of each channel. The acceptor runs checks to ensure that each channel is operating as expected, and throws an error if either fails[9]. The acceptor continues checking the values stored in the HTLCs to ensure that they do not exceed or fall under the global maximum or minimum value allowed to be traded. Once they establish channel security and valid HTLC values, the exchange process now proceeds to state change verification.

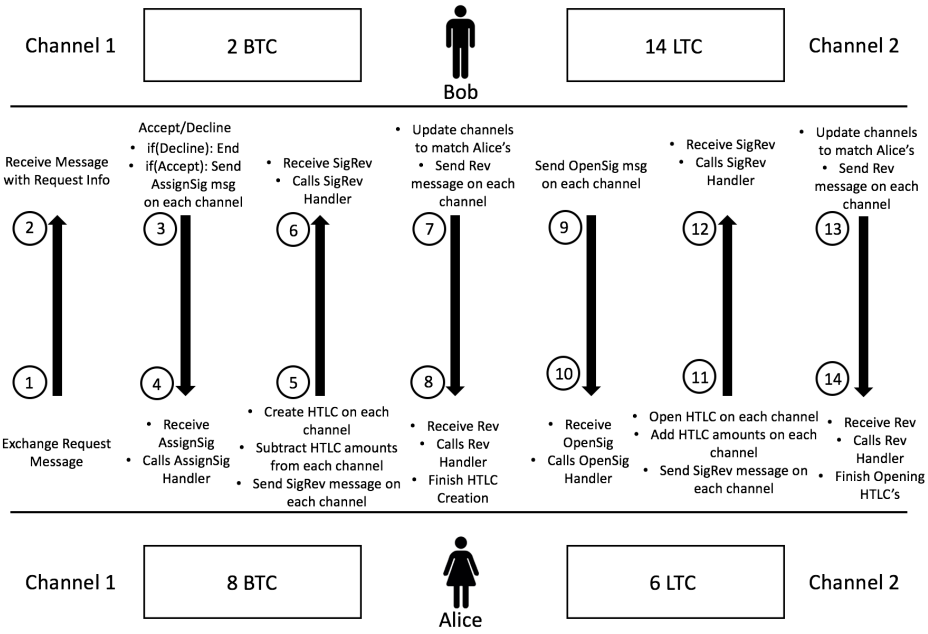

Figure 10: Depiction of the message exchanges that make up the cross-chain exchange procedure. The figure does not take into account any errors that could occur during the exchange process.

[9]Failures include the channel not being at rest state, the channel currently executing another command, the channel no longer existing, etc.

The state change verification process consists of three autonomous phases: the AssignSig phase, the SigRev phase, and the Rev phase. These three phases are represented by steps 3 through 8 in Figure 10 for HTLC creation, and are sent once on each channel for each HTLC. Even though steps 3 through 8 start are executed once on each channel, the execution starts on the acceptor's machine both times. However, the acceptor marks the process as incoming or not incoming based on whether they are initializing the HTLC that they are supposed to receive or the HTLC that they are supposed to send. To initiate the assignment process, the acceptor constructs an AssignSig message. The AssignSig phase consists of storing value in a HTLC and building a transaction with the new proposed balances, the value stored in the HTLC, and a new signed state. The HTLC value will be subtracted from the sender's old balance to generate their new proposed balance and maintain the overall channel value, and the sender is determined based on the acceptor's incoming flag. For the Exchange command, we use an AssignSig message instead of a DeltaSig message because there is no delta. Instead, we assign value from the sender to a HTLC, and then mark that HTLC as incoming for the HTLC recipient. The newly built transaction is signed and sent as a message to the initiator, who processes the new proposed transaction and validates the proposed state (checking for an unchanged channel balance, valid user balance updates, etc). After the new state is approved and agreed upon, the recipient also updates their channel state, signs the proposed change, and sends their agreement back as a message with the new signed state and a revocation private key, transitioning into the SigRev phase. In the SigRev phase, the acceptor considers the initiator's previous state revoked, and sends a Rev message back to revoke their own previous state. The initiator receives a private

key in the Rev message, and return the channel to a rest state so that it is ready for later use.

As mentioned, the whole process outlined above happens once on each channel, having each user store their proposed amount in an HTLC for the other user. After the process concludes for each channel, both HTLCs are assigned to their corresponding channel and marked as incoming for the corresponding recipient, and Bob's machine initiates the unlocking of the HTLCs.

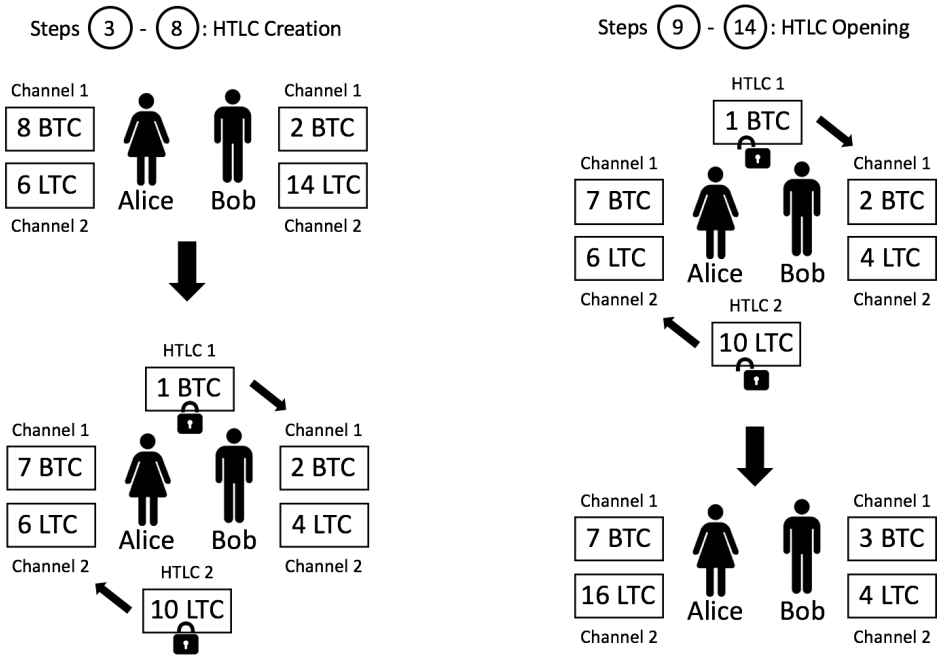

Figure 11: HTLC assignment and unlocking process.

5.4.3 HTLC Unlocking

We mentioned previously that the preimage originally used to create both HTLCs was stored in the acceptor's machine's memory, and at this point, the acceptor's machine retrieves that preimage and passes it to each node. The nodes then hash the preimage, and cross reference the produced

hash with the corresponding RHash in the stored HTLC marked as incoming for each node. If the hashes do not match, an error is thrown and the HTLCs remains locked. If both hashes match, both HTLCs are considered unlocked and the exchange process continues into its final stage.

With both HTLCs considered unlocked, each HTLC recipient is given the value from their corresponding HTLC and their states are updated. Instead of the DeltaSig or AssignSig messages, the unlocking phase includes the OpenSig message. This is because, again, there is no delta, and we are no longer assigning values to HTLCs, but instead opening HTLCs and extracting value. The OpenSig message clears both linked HTLCs and updates the final exchange balances as shown in steps 9 through 14 in Figure 10, again being sent on each channel for each HTLC. The OpenSig message is sent on each channel from Bob's machine, and it checks the incoming tag in the HTLC to know which node's balance it should increase in each channel. Much like the assign process, the OpenSig message sends a new signed state to the initiator and clears the HTLC. The initiator then runs the SigRev message to sign the new state and send back a private key, and the acceptor responds with a Rev message and a private key of their own.

For a potentially clearer representation of both the assignment and unlocking phases, Figure 11 breaks down how the system structures behave, and how the tokens move from one user to another. The diagonal arrows in that picture show how steps 3-8 and steps 9-14 happen once on each channel, meaning that the messages in each of those steps are sent two times in total to set up and clear each HTLC on each channel.

5.5 User Interface

As this builds off of the lit implementation of Lightning Network, the system UI includes terminal operation or the use of several GUIs. Users keep the same experience that they had on the lit implementation of the Lightning Network originally, but now have four new functions in their channel commands.

It is worth noting that our implementation currently relies on user awareness to a certain extent. Users are in charge of understanding the format of the Exchange commands, and in turn, understanding exactly what currencies they are potentially exchanging and with whom. Both of those components are implied based on the selected channels and the order in which the channel/amount appear in the input command. Although these specifications are explained in the lit command guide accessed through the Help command, it admittedly is not as naturally intuitive as we'd hope it to be. However, the focus on of our work to date has been on the protocol implementation.

Another important detail that is not directly communicated but that we would like to point out is that the initiator and acceptor have no obligation to accept a request. For example, the acceptor doesn't need a valid reason for refusing to accept a request, and the initiator doesn't need to fulfill a request that the acceptor accepts. Both parties are free to turn down a request at any point before tokens have been exchanged.

This page intentionally left blank.

6 Discussion and Future Work

We have constructed the full cross-chain exchange protocol pipeline on the Lightning Network as a proof of concept, and it remains future work to merge our developments to the main Lightning Network code base. Unfortunately, we did not have time to test the system, and we acknowledge that although the base functionality is implemented, the system is by no means ready for public use. If we had more time, we would simulate live exchanges across various different chains, focusing on testing of edge case inputs, and we would attempt to optimize the system further based on our results.

6.1 Future Work

To add onto the work we cover in this paper, we would focus on improving the following: system fault tolerance, HTLC revocation, non-currency chain communication protocol[10], a more friendly user interface, exchange channel hopping functionality, and system generated exchange rates.

6.1.1 Fault Tolerance

As stated in Section 6.1, the exchange process still lacks the ability to be fully functional in live settings. Our code pipeline does not share the concurrency checks and balances that exist throughout the rest of the Lightning Network yet, so we must add checks into each step of the Exchange command pipeline to cross reference current channel states and assess process misalignment.

[10]We imagine an ecosystem where individuals can share information stored on blockchains with tokens that don't act as currency, such as IBM's proposed blockchain for fraud detection[19].

For example, if Bob's channel receives a SigRev message, but his channel's delta currently equals zero, then something has undoubtedly gone wrong. The contents of the message are what's important for the system to be able to execute each step correctly. Bob's delta being equal to zero means that his channel is at rest, and is expecting an instruction to initiate some action (such as a change in delta). However, the SigRev message includes a key and a signature, which would signal that Bob should revoke the previous state and sign the new state generated based on his node's current delta. His current delta being zero, though, means that revoking the previous state and signing a new state would not make sense as both would be equal. The exchange process should then be able to evaluate whether the received SigRev message was a duplicate, whether Bob's delta was not properly updated earlier, etc.

This type of message discrepancy can affect many of the step in Figure 10. Currently, we only provide fault tolerance through the HTLC locktime, which sends funds back to their depositor upon expiration. Although expiration can help users recover their funds, we hope to introduce more checks and balances to handle problems, such as message discrepancies, for more of these steps.

6.1.2 HTLC Revocation

Our pipeline is configured to process a full exchange request, but currently cannot revoke a request once it has been placed. If an error occurs along the way, as mentioned above, our users would be unable to revoke the value they stored in the HTLCs. In a finalized implementation of the exchange process, we envision users being able to unlock HTLCs that hold tokens belonging to them on chain and on the Lightning Network. We

should also make sure that whatever revocation protocol we implement does not give rise to malicious activity. For example, if Alice and Bob are exchanging tokens, and Bob receives Alice's tokens first, there should be no race condition in which Bob could revoke the tokens he promised Alice before they are sent to her.

6.1.3 Non-Currency Chain Communication Protocol

With few established non-monetary chains, we did not configure a non-currency chain communication protocol, but as an open-source impact-driven system, the Lightning Network is malleable and ready for that change if and when it comes.

Trading currency provides clarity when exchanging tokens. Alice and Bob simply have to agree on the tokens' values based on market prices and their individual supply and demand. However, we did not get a chance to tackle the issue of non-currency tokens, which would likely revolve around the exchange of data.

It is difficult to formulate a marketplace where peers can accurately and securely exchange data and information. How exactly would users go about valuing their data? How would users be able to prove that value without compromising the data they are hoping to exchange? Our initial approach would leverage the Lightning Network's existing ability to transfer data between users, but we would have to improve data security throughout the network if we hope to guarantee the integrity of valuable data being exchanged.

6.1.4 Improve User Interface

To focus on UX design[11] is to focus on a future where people adopt our technology and integrate it into their lives. Studies have proven that improved UX leads to success for both the user and the entity building its user base. Users are able to perform tasks more quickly and efficiently, they are happier while performing those tasks, and they are even subconsciously encouraged to return and complete those tasks again[1]. In turn, the user base grows as retention stays high and a product or service evolves from a useful tool into an everyday necessity.

Our exchange protocol has potential, but for that potential to be realized, users must feel the same way. The best course of action would be to update our channel interface to either replace channel numbers with channel names or contact names, and to denote currency along with amounts. This would go a long way in appealing to usability for users and would improve their experience with our system.

6.1.5 Channel Hopping Functionality

Although not covered in this paper, the Lightning Network can simplify its own network through channel hopping. The channel hopping mechanism allows users to take advantage of the network of channels and transact with anyone reachable through existing channel connections. For example, if Alice has a channel open with Bob, and Bob has a channel open with Charlie, as seen in Figure 12, then Alice could transact with Charlie without having a channel directly with him by sending tokens through Bob. Channel hopping could dramatically reduce the number of channels that

[11]UX refers to user experience, and is an ideology that focuses on improving the experience that all users have with a particular device or technology across all points of contact in order to improve efficiency, retention, and satisfaction

all users would need to have to transact with a high number of peers.

With that being said, our current exchange implementation does not support channel hopping, but we hope to add channel hopping functionality in the near future as it simplifies the network and decreases the global number of channels.

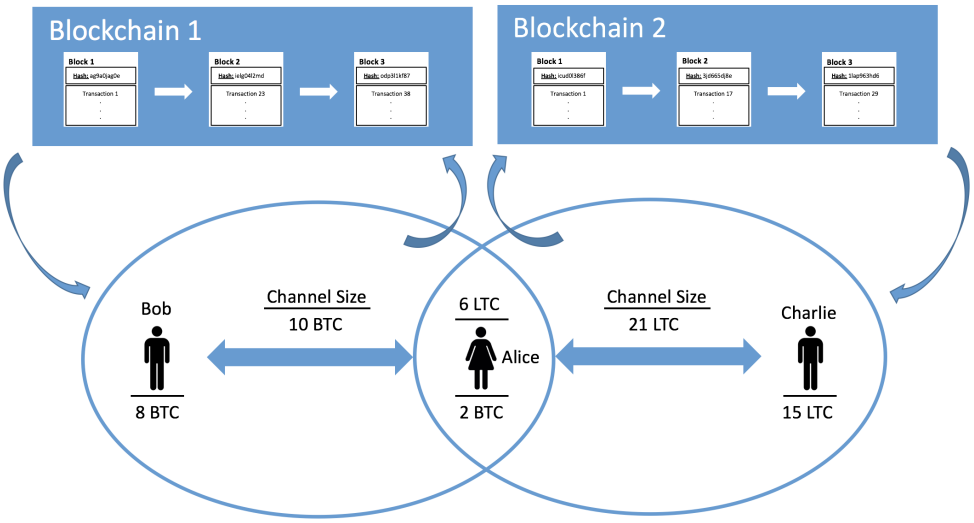

Figure 12: Visual representation of connected channels with hopping functionality.

6.2 Real World Applications

To provide more tangible examples of our solution's use cases, we cover what our system's role would look like in two real world applications. These are not by any means the only use cases applicable to our project, but they are two that exemplify our system's abilities best.

6.2.1 Delivery vs. Payment

A delivery vs. payment (DVP) transaction refers the relationship between the delivery of goods and the payment of currency in a multiparty

transaction[8]. This concept affects businesses to different degrees all across the world. From the payment and shipment of Alice's latest Amazon order, to the payment and exporting of steel from China to the US, to even the payment and delivery of Bob's last Starbucks order, we encounter DVP universally as a product of comparative advantage. For simpler use cases, such as buying coffee in the morning, DVP occurs at the point of contact between the two parties and resolves itself easily. At the scale of exporting goods to another country, however, DVP can pose various challenges.

Similar to our cross-chain exchange platform, third parties fill the void in this scenario, and offer confirmation services to facilitate the DVP exchange. However, a system could be developed to ensure proper payment at the point of delivery, even in a scenario as complex as exporting and importing.

We imagine a system configured on hand held devices running our Lightning Network off-chain. These devices could then read and confirm barcodes printed on the goods themselves or on their shipping containers, and communicate within the Lightning Network channel to confirm their current owner. For example, Alice could scan a barcode and establish her ownership of those goods as a token in a channel. If Bob was the peer in that channel, she could send him the goods, placing them in an HTLC, and in another channel with Bob, have him place some amount of BTC in exchange. Both tokens would be held in their respective HTLCs until Bob received the goods, scanned the barcode, unlocked both HTLCs, and transferred both sets of tokens to their new respective owners. If Bob received the goods but refused to scan the barcode, Alice could cancel the HTLC exchange, reclaim her token representing her goods, and pursue legal action as she has proof of ownership of the goods. Now replace Alice

and Bob with China and the US, and the system's potential seems more promising.

We admit that this is a simplified hypothesis, but it serves as a use case example that could potentially simplify and improve the DVP environment.

6.2.2 Currency Exchange

As mentioned, currency exchange is a clear application for this project. Exchanges already exist that fulfill most needs for this use case, but as covered in Section 2, they create friction in an ecosystem designed to be frictionless. Our solution fosters a monetary exchange ecosystem in which third party systems would no longer be needed to facilitate currency exchanges. Empowering individuals in the network to execute exchanges themselves, our decentralized currency exchange expands financial inclusion in a scalable fashion, a driving force for economic growth and development[12].

Examining cryptocurrency's financial inclusion effects in Nigeria highlights the extent to which this type of technology could serve as an economic catalyst. Of Nigeria's 180 million people, only 17% (about 30 million) have proper access to financial services (such as bank accounts). The majority of these 30 million individuals struggle to deposit more than $4 a week due to lack of trust in the integrity of banks and the financial system, yet Paxful, a Bitcoin supplier, now handles over $10 million a week in transactions between Nigeria and China alone. Paxful opened the door for thousands of individuals unable to meet financial requirements to participate and contribute to the Nigerian economy[23]. Involving these previously ignored parties is shown to hold valuable potential and is a value that we hope our project can eventually contribute[12].

This page intentionally left blank.

7 Conclusion

In this paper, we established a foundation for the peer to peer exchange process across blockchains, eliminating the need for trusted third parties and reaffirming the decentralized nature of blockchains. Trusted third parties filled the market demand for intermediaries facilitating the exchange of tokens from one blockchain to another, but they proved to be risky institutions that incurred substantial financial losses. The same parties also served as centralizing agents, stunting the blockchain ecosystem's fundamental decentralized nature. Working off of the existing Lighting Network code base, we developed a cross-chain exchange protocol that can also operate off-chain. Our system allows users to transact across compatible blockchains, and to do so on their own terms. The software implemented in this thesis can be accessed through the link in the Appendix.

Our work is a good first step forward, but we understand that there is more to be done to improve the peer to peer exchange process across blockchains. The system we developed establishes the foundation, but we are excited to see what changes and improvements are made to strengthen individuals, and in turn the whole, through the decentralized efforts of the blockchain ecosystem.

Individuals no longer depend on obscure third parties to enable the exchange of one token for another. Anyone with a computer now has the ability to transact across blockchains independently, rekindling blockchain's decentralization spirit and liberating valuable dead capital.

This page intentionally left blank.

8 Bibliography

[1] Wilbert O Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design*. Wiley Publishing, Inc., 1997. ISBN: 9780470053423.

[2] Primavera De Filippi Aaron Wright. *Decentralized Blockchain Technology and the Rise of Lex Cryptographia*. URL: https://www.intgovforum.org/cms/wks2015/uploads/proposal_background_paper/SSRN-id2580664.pdf.

[3] John D. Halamka Ariel Ekblaw Asaph Azaria. *A Case Study for Blockchain in Healthcare: MedRec Prototype for Electronic Health Records and Medical Research Data*. URL: https://www.healthit.gov/sites/default/files/5-56-onc_blockchainchallenge_mitwhitepaper.pdf.

[4] Vitalik Buterin. *Ethereum White Paper A Next Generation Smart Contract and Decentralized Application Platform*. URL: http://www.the-blockchain.com/docs/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[5] Bit Info Charts. *Bitcoin Avg. Transaction Fee Historical Chart*. URL: https://bitinfocharts.com/comparison/bitcoin-transactionfees.html.

[6] Bit Info Charts. *Bitcoin Transactions Historical Chart*. URL: https://bitinfocharts.com/comparison/bitcoin-transactions.html.

[7] Adam Kaleb Ernest. *The Key to Unlocking the Black Box: Why the World Needs a Transparant DAC*. URL: https://followmyvote.com/wp-content/uploads/2014/08/The-Key-To-Unlocking-The-Black-Box-Follow-My-Vote.pdf.

[8] Bank of Japan European Central Bank. *Securities Settlement Systems: Delivery-Versus-Payment in a Distributed Ledger Environment*. URL: https://www.ecb.europa.eu/pub/pdf/other/stella_project_report_march_2018.pdf.

[9] Australian Government Department of Foreign Affairs and Trade. *Saudi Arabia*. URL: http://dfat.gov.au/trade/resources/documents/saud.pdf.

[10] Michel Rauchs Garrick Hileman. *Global Cryptocurrency Benchmarking Study*. URL: https://www.jbs.cam.ac.uk/fileadmin/user_upload/research/centres/alternative-finance/downloads/2017-global-cryptocurrency-benchmarking-study.pdf.

[11] Andrew Gazdecki. *The First Ethereum ¡¿ Bitcoin Atomic Swap*. URL: https://blog.altcoin.io/the-first-ethereum-bitcoin-atomic-swap-79befb8373a8.

[12] Adegbola Dare Harley Tega Williams Adetoso J. Adegoke. *Role of Financial Inclusion in Economic Growth and Poverty Reduction in a Developing Economy*. URL: https://www.researchgate.net/publication/321197681_ROLE_OF_FINANCIAL_INCLUSION_IN_ECONOMIC_GROWTH_AND_POVERTY_REDUCTION_IN_A_DEVELOPING_ECONOMY.

[13] Stan Higgins. *Cryptsy CEO Stole Millions from Exchange, Court Receiver Alleges*. URL: https://www.coindesk.com/cryptsy-ceo-millions-digital-currency-steal/.

[14] IBM. *The Benefits of Blockchain to Supply Chain Networks*. URL: https://www-01.ibm.com/software/commerce/offers/pdfs/Blockchain_3-15-2017.pdf.

[15] The Levin Institute. *Trade and Globalization*. URL: http://www.globalization101.org/uploads/File/Trade/tradeall.pdf.

[16] Peter Loop. *Moving to Distributed Systems: Blockchain and the Standards Opportunity*. URL: https://www.infosys.com/insights/services-being-digital/Documents/moving-distributed-systems.pdf.

[17] Carola McGiffert. *Chinese Soft Power and Its Implications for the United States*. URL: https://csis-prod.s3.amazonaws.com/s3fs-public/legacy_files/files/media/csis/pubs/090403_mcgiffert_chinesesoftpower_web.pdf.

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

[19] Dr. Indranil Nath. *Fight Insurance Fraud: Data Sharing With Blockchain Technology*. URL: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=IUW03049USEN.

[20] Alessandro Nicita. *Exchange Rates, International Trade and Trade Policies*. URL: http://unctad.org/en/PublicationsLibrary/itcdtab57_en.pdf.

[21] Pramod Kumar Rajesh Chaudhury Udhaya Kumar. *Foreign Exchange Markets*. URL: http://www.pondiuni.edu.in/storage/dde/downloads/ibiv_forex.pdf.

[22] David Scutt. *CHARTS: Here's How Much Currency is Traded Every Day*. URL: https://www.businessinsider.com.au/charts-heres-how-much-currency-is-traded-on-average-every-day-2016-9.

[23] Allan Smith. *Bitcoin is Becoming a Tool for Financial Inclusion in Africa*. URL: https://www.huffingtonpost.com/entry/bitcoin-is-becoming-a-tool-for-financial-inclusion_us_5a0e%20f83be4b0e30a9585062c.

[24] Joseph Poon Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. URL: https://lightning.network/lightning-network-paper.pdf.

[25] Aaron van Wirdum. *The Lightning Network Now Supports Transactions Across Blockchains*. URL: https://www.nasdaq.com/article/the-lightning-network-now-supports-transactions-across-blockchains-cm878904.

[26] Jake Yocom-Piatt. *On-Chain Atomic Swaps*. URL: https://blog.decred.org/2017/09/20/On-Chain-Atomic-Swaps/.

This page intentionally left blank.

9 Appendix

README.md

Lightning Network Exchange Command GitHub Repo:

https://github.com/jmathus/lit/tree/HTLCswaps

Our project code can be found at the repo above, and here, a quick overview is as follows:

The command starts in the lit-af.go file at line 115. This line initializes a continuous for loop that listens for user input. The user input is read, split by white space, and sent to the shell.go file. The shell file parses the inputs, checking the first item in the split string list and comparing it against all existing commands. This file is the first point where our Price, Compare, Exchange, and Respond commands are recognized.

If any of the commands are recognized, all input information is forwarded to the chancmds.go file under the lit-af directory, where the inputs are converted to their proper Golang typing. The Price and Compare commands are both fully executed here, accessing the previously mentioned website and retrieving the desired information. The Exchange and Respond commands, however, continue execution past this file.

Once converted, the information is passed to and RPC call in the chancmds.go file under the litrpc directory for those two commands. Both functions share the commands' name, and both process the given inputs (checking for improper input format such as negative values, incorrect response values, etc.) before loading the correct user channels for command execution. The Exchange RPC function uses the two loaded channels and the input Exchange command amounts to create a message with the SendExchangeRequest() function in the exchange.go file in the qln directory. SendExchangeRequest() builds a request message (as we defined in msglib.go under the lnutil directory) and forwards it to the node's OmniOut, starting the messaging phase.

The OutMessager() listens for and handles all OmniOut messages, sending them out to the network. To process incoming messages, each node has a LNDCReader() in the msghandler.go file under the qln directory that listens and parses incoming messages. The parsed messages are then passed to the PeerHandler() in the same file, which then forwards the message accordingly.

At this point the messaging phase ends, and the PeerHandler() calls our ExchangeHandler() at the bottom of the same file. Because it receives an exchange request message, the handler calls the ExchangeRequestHandler() in exchange.go, which outputs the command line text seen in Figure 8 on the acceptor's terminal.

Now the acceptor should respond "YES" or "NO" as seen in Figure 8, and their command will follow the same pipeline up to the RPC in chancmds.go under litrpc, except this command will invoke the Respond RPC. The command will do nothing if the acceptor responds "NO", but will the exchanging of tokens if the acceptor responds "YES". This will once again load the appropriate channels (on which to exchange), and will create the independent HTLC and dependent HTLC using the makeHTLCNoPreimage() and makeHTLCWithPreimage() functions. With the channels and HTLCs, Respond() then calls AssignHTLC() in exchange.go for each channel. This function executes the material covered in Section 5.4.2, and does so in the htlcassign.go file under the qln directory; htlcassign.go contains the AssignSig, SigRev, and Rev message execution code, and each message struct and specifications can be found in msglib.go.

After both AssignHTLC() functions terminate, Respond() calls OpenHTLC() in exchange.go, again, for each channel. This function executes the material covered in Section 5.4.3, and does so in the htlcopen.go file under the qln directory; htlcopen.go also contains its own OpenSig, SigRev, and Rev message execution code, and each message struct and specification can also be found in msglib.go. When the two OpenHTLC() functions terminate, the exchange process is over, and the channels are put back to rest state.