UMEÅ UNIVERSITY
Department of Computing Science
Master thesis report

Master thesis presentation document
30HP, 5DV143, VT19

Evaluating Cross-chain Settlement and Exchange in Cryptocurrency

Chapter 1

Abstract

To summerize the entire thesis I would say: "I have experimented with, and studied the aspects of different types of atomic swaps. Wth the goal of formalizing a proposal on how they should be standardized and handled in the future." You probably do not know what an atomic swap is yet, but I will cover that.

The general topics covered are:

- Cryptocurrencies and how they can be used in special use cases.
- Cryptography
- Smart/programmable contracts
- Payment channels and lightning network

Don't worry if you do nt know anything about thee topics as I will cover them in this presentation.

Chapter 2

Introduction & Background

Alright, so the thing that is on everybody's mind is probably: what is an atomic swap? An atomic swap is where two parties exchange assets atomically, which means that either the transaction takes place fully, or the state is reset to the pre- exchange state.

Read from oracle defenition

This is made possible by clever use of cryptography and programmable contracts on the bitcoin network and blockchain. So before we can go into more detail on this I should first cover the basics of Bitcoin.

So, most people, especially in computer science, have heard of Bitcoin, but I could almost count on one hand the number of people I have met that have more than a basic understanding of how it works. I could talk for hours about this subject, but sadly there is no time for that. So I will try to give you the shortest possible version where you can at least understand the rest of my thesis.

The simplest description of Bitcoin is a shared public ledger, that relies on proof-of-work to build network-wide consensus. First of proof-of-work is a way to prove mathematically that work was put into doing something. The most common way of doing this is via hashing of some datatype. The hash has to meet certain criteria to be accepted. There is no known way of producing a wanted hash, so the only way is to try different combinations until a good result is found. So if you have data that produces a certain hash that hash serves as proof that you put work into creating it.

Another thing you have probably heard about before is the blockchain, but just as with Bitcoin overall, people know little about what it actually is. A blockchain is basically a shared datatype. It is very reminiscent of a linked list, but allows for branching, meaning that two elements can link to the same parent. We will come back to this in a moment, but first, let's take a closer look at the blocks.

A block is a data structure that has a header and data. The header contains metadata about the block itself as well as a reference to the previous block in the chain. In Bitcoins case, the data in the block is just a list of transactions, but you could put anything you want into this field. The reference to the previous block is what forms the chain. You can from any block follow the references all the way back to the original block. Anyone can add a block to the chain. But it has to meet the proof-of-work criteria. The Bitcoin network independently calculates something called mining-difficulty. This is represented by a large 256-bit number. For your new block to be accepted the header of the block has to produce a hash that is strictly smaller than the difficulty number. You produce unique hashes by changing a field in the header called nonce, this process is what is referred to as mining.

The mining-difficulty is set so that the sum of all participant's hash-calculating power, or hashrate, will produce new block on average every 10 minutes. The difficulty is adjusted every 2016th block.

So how does proof-of-work ensure that the shared ledger stays consistent? This is where concepts like longest chain comes in. The Bitcoin network only accepts the longest chain as truth, in other words the chain with most accumulated proof-of-work. This works as long as the majority of participants is honest. However due to the probabilistic manner of how new valid blocks are found there is still a chance for contention even if all participants are honest. For example what will the network do in the case where two different blocks are found at almost the same time, in different parts of the network? Now there are two chains of equal length. So how does the network decide which one is correct?

Explain pictures

2.0.1 Elliptic curve cryptography

For cryptographic purposes Bitcoin uses a elliptic curve, known simply as elliptic curve cryptography or shortend ecc. I will not cover the exact details of the maths, it can be found in details in my report, basically all you need to know is that it is an assymmetric cryptography, meaning it has private and public keys.

Though encryption and decryption is possible with ecc it is not the purpose it serves for Bitcoin. What is used is signing. This is a concept that all computer scientists should be familiar with but basically what it means is that you can sign a string of data with a private key, and that signature could be independently verified by someone who has the data prior to signing and the public key belonging to the private key, all without revealing the private key.

Clasically signing is used to prove identity. For example a website could sign a document to prove that it truly was sent by the server it purports to be. This is a very useful feature, but in the world of smart contracts the signature has more significance.

Signatures tend to hold the following significance:

- **Identity** - A signatures serves as proof that it came from the holder of the private key.
- **Immutability** - A signature makes whatever was signed immutable in a way. If the data is manipulated after the signing the signature will no longer be valid.
- **Agreement** - A signature can serve as proof that the holder of a private key agreed or approved to whatever data was signed. For example signing a digital contract could be seen as agreement to the conditions in the contract. The agreement can't be withdrawn at a later date as the signature proves mathematically that you signed it. There is of course the possibility that the private key was stolen. Just goes to show how important it is to keep the private key private, especially in Bitocin.

Keep the 3 above points in mind. As they are absolutley vital for the functionality of smart contracts.

2.0.2 Script & Transactions

Alright, now that we got consensus and the cryptography down let's take a closer look at how programmable contracts are made possible over the bitcoin network.

Built into Bitcoin there is a programming language called Script. Not the best of names, but let's not think about that for now. Script is the code that defines exactly what a transaction can and cannot do. It is a very bare bones language, with byte sized instruc-

tions, no heap memory, and only a very limited stack memory, that strictly operates like a stack. Very reminiscent of the Forth language.

A piece of Script code can be defined as either valid or invalid. The only way a script could be valid is if at the end of executing the last instruction there is only one value on the stack, and it equates to true. Anything else will make the script invalid, in fact under certain circumstances the script could be marked as invalid before the execution is even finished.

Script has one major limitation compared to other programming languages, it is by design not Turing complete. Basically meaning that it cannot do anything. What is missing from the usual programming language repertoire is loops. Script has no construct that allows for repeated instruction execution. Instead all Script programs are completely linear and always execute in a limited time. The reason for this has to do with the halting problem and a bunch of anti-spam measures. Even with this limitation Script has a very wide use case as you will soon see.

Transactions in Bitcoin are not as straight forward as you might expect. It's not just a matter of moving from one attached name to another. A transaction in Bitcoin is a datatype consisting of meta data, a list of inputs and a list of outputs. An input is a reference to a previous output from an earlier transaction, an output could be seen as a destination, it holds the amount sent and the "destination". Because of the references to previous transactions in the input you can follow transaction all the way back to the generation transaction when the coins were created in the first place.

The inputs and outputs are where Script comes into the picture. A output contains a partial script. A input contains the complementary part forming the entire script. This is what makes Bitcoin transactions so versatile. You could make the output script be anything that is describable in Script.

For example *Show simple script on screen*. This script requires the redeemer to give the answer to this equation to spend the money in the output. *Show redeeming input script*.

Obviously most Bitcoin transactions are not this basic. The most common script type has a name, P2PKH, Pay to public key hash. In this script the output is given a hashed public key, and the one who wants to spend the output has to provide a signature from the private key related to the given public key. *Show script and redeeming script*

A slightly newer development in Bitcoin is P2SH, Pay to script hash. If the previous example can be seen as paying to a bitcoin address, this one can be seen as paying to a script, or a contract. In this case anyone who can provide the contract together with a partial script that makes the contact valid, is allowed to redeem. The output paying to the script contains only the hash of the script, as the name implies.

2.1 Payment channels

With the topics we have covered so far under our belt we could start moving into the more complex areas of Bitcoin. One such concept is payment channels.

Before going any further let's define two very important terms: Off-chain and on-chain. These terms refer to a proccess or data that ocurs on the blockchain or outside the blockchain. FOr example the transactions we have described so far has been on-chain, as in the transaction is broadcast to the bitcoin network and is later included in the blockchain. Something that is off-chain is data or a proccess that are outside the blockchain. A payment channel is something that is typically refered to as off-chain. You will see why soon.

With programmable contracts using clever scripts and signatures we can make a new type of construct that allows for safe transactions being made between two peers without being braodcast to the blockchain. I would love to cover this in more detail, there is sadly not enough time, so I will only cover the basics.

The channel is opend by the peers by broadcasting a funding transaction it's output can only spent by having the signatures of both participants. As long as the funding transaction remains unspent payments can be amde between the two participants in the form of commitment transactions. A commitment transaction spends the funding transaction and splits the money among the two participants in the agreed upon amounts.

Imagine a payment channel open between two fictional characters Alice and Bob. In the channel Alice has 0.5 btc and Bob has 0.5 btc. This would mean that the output in the funding transaction has 1.0 btc and the latest commitment transaction has two outputs both worth 0.5. If Alice wants to send 0.1 btc to Bob they make an agreement, and then construct a new commitment transaction updating the output values. In this case the new values would be 0.4 to Alice and 0.6 to Bob. To close a channel you simply need to broadcast a commitment transaction.

Additional security is needed to prevent things like broadcasting old commitment transactions for monetary gain. The additions are called revocable delivery and breach remedy.

Show images and explain what is going on

2.2 Lightning network

The payment channel I just described allows for unlimited transaction between two people. It is quite usefull if you have two people who make a lot of transaction between each other, but it would be very encumbersome to have a channel open to every person you might ever make transactions to. This is where lightning network becomes useful. With just a few additions to the payment channel it ill allow for transactions that travel through multiple channels, safely of course.

To do this channels add another output to their commitment transactions that pay to something called Hashed timelocked contract or HTLC for short. Just as before I would like to cover the fine details of the contract, but there is no time for that. In simplest possible terms the contract says that each node along the way will receive the money they put up plus a small fee if they can prove that the transaction reached it's destination. This is done via hashes and pre-images, something I will touch upon a bit later in this presentation.

The contract has a built in timelimit, if the transaction never makes it the whole way to it's destination all transactions along the way will be refunded and nobody will receive their fee.

Show basic image, talk about how large the network is

Chapter 3

Implementation of atomic swaps

So finally we get to the implementation part of the presentation. Becauce I did two different types of atomic swaps I have decided to split this part up into two parts. The first parts cover what is known as on-chain atomic swaps, and the second part will be the bit more complex off-chain atomic swaps.

3.1 on-chain atomic swap

THere are several ways to implement a on-chain atomic swap. The one I choose is probably the easiest to understand. Before I go into implementation specifics it would be good to know how this process actually works.

Imagine two people, Alice who has Bitcoins and wants litecoins, and Bob who has litecoins and wants bitcoins. Here a swap is possible. THe methods people start thinking about right away could be for example some sort of third-party that ahndles the swap. Or maybe Alice just sends the Bitcin to Bob and hopes that he doesnt take the money and run. Overall the problem he is trust. Even with a third-party there is a chance that Bob and the third-party are cooperating to steal from Alice. With the help of programmable contracts however, we now have a method of swapping where the only thing you have to trust is numbers.

The process of an on-chain atomic swap is as follows: Alice and Bob agrees to do a swap, 1 bitcoin for 10 litecoins. They also decide that Alice should be the one to initiate the exchange. To start off, Alice generates a random bytestring that will act as a pre-image, let's call this R. She hashes this pre-image and produces H_R. With te help of H_R she constructs a new swap contract with the following clauses. Pay 1 bitcoin to Bob if he can provide the pre-image of H_R (R). If Bob does not claim this output wihin 48 hours, refund the full amount to Alice.

Alice broadcasts this contract transaction to the bitcoin blockchain and notifies Bob of doing so, she also sends the unhahsed contract to Bob. Bob can then fetch the transaction from the blockchain, he then makes sure that the contract hash matches the one on the chain (P2SH) and he validates all the details of the contract.

If something is wrong here or if Bob changed his mind, he does not have to do anything, and Alice can refund her money after 48 hours. He can not try to claim Alices bitcoins as he does not know the pre-image. If everything in the contract is alright however he constructs a new contract with the following details. Alice gets 10 litecoins if she can provide the pre-image to H_R, if she does not claim them within **24 hours**: refund the full amount to Bob. Bob broadcasts this transaction to the litecoin blockchain then notifies Alice and sends over the unhashed contract.

Alice fetches the contract and validates all the information just like Bob did. If something is wrong or she has changed her mind she can do nothing, Bob will get his refund after 24 hours and Alice gets her refund after 48 hours. Otherwise she can claim Bob's litecoins, she does this by constrcting a claim transaction containing the pre-image and a signature from her private key.

Meanwhile Bob has been monitoring the litecoin blockchain and when Alice claims the litecoins he will see the transaction together the pre-image data. Now when he has the pre-image he can construct his own claim transaction that claims the bitcoins from the initial contract.

The safety of this swap comes from the reveal mechanism of the pre-image being the blockchain and the decreasing timelocks. Those of you who have paid close atention will notice that the mechanisms are very much like the ones used in hashed timelocked contracts.

3.1.1 Implementation specifics

So let's take a closer look at how I implemented this. To work with bitcoin and litecoin you need a very specific setup. I had one Bitcoin node and one Litecoin node running on my raspberry pi 3 at home. Both running the latest version of their respective core implementations. The nodes were used for interacting with the blockchain (broadcasting, reading, monitoring etc...). BOth of them ran on the testnet version of the respective blockchains. This is a global test network where the coins are considered worthless, but the important parts like blocktime and consensus works exactly the same. THe testnet exists exactly for the purpose of testing contracts and other bitcoin software before use in the real world.

My implementation was entirely writen in go-lang. I had never written any code in go so at the start I spent a day or two just learning how to operate. THe reason for choosing go was mostly because of the many great libraries that are available when programming towards bitcoin and other cryptocurrencies. The most used library was the btcd-suite. This is the code used for the go-lang specific implementation of a bitcoin node, it is modularized in such a way that you can basically use it for any Bitcoin specific task. In my case most of the coding was in regards to constructing the specialized transactions.

The on-chain atomic swap implementation actually consists of three small programs. Two of the programs are from ALices perspective and the other two are from Bobs perspective. In my little example the bargening and agreement parts are set in stone, meaning that the exchange rate, timelocks etc are hardcoded. Both participants are represented as data structures that contains stuff like their private key and their public key.

The first program constructs the very first contract transaction that Alice will broadcast to the bitcoin blockchain. THe btcd libarary provides the proper datastructures for the transaction related stuff, the contract has to be constucted manually however. This is the entire contract script for Alice: *Show on screen*

None of the programs actually broadcast the transactions, instead they print it in hex-format. THe hex-formatted transaction can be broadcast manually by the user by running a simple command in the Bitcoin-core node.

The other programs are just part of the steps I described earlier. The second program creates the Bob side of the contract, the contract looks exactly the same but the timelock as been altered to 24 hours, and of course the payout is in 10 litecoins. The third and fourth program constructs the claim transactions.

Overall making this into several programs was pretty pointless. On the bright side it works

just as intended. I tested all possible scenarios and they all worked just like intended.

3.2 Off-chain atomic swaps

How an on-chain atomic swap functions should be pretty clear. The script never does anyhting I would call super complex. Off-chain atomic swaps however is a bit of a different story and it can be pretty hard to grasp, escpecially with the simplified explenations I have given so far.

An off-chain atomic swap is as you might imagine an atomic swap, but if it goes as planned it occurs entirrly without any script being executed on the blockchain. In fact this version of an atomic swap does not use any specialized scripts at all, instead it uses the clever mechanisms in htlc contracts to perform it, even across chains.

As you might remember from payment channels and lightning network a multi-hop transactions relies on the htlc contract to be safe. THe htlc contracts needs the same hashed pre-image and a strictly decreasing timelock to be safe. This holds true even in the case where a multi-hop transaction travels across chains. The requirements is that ALice and Bob run specialized lightning network software, that runs on both chains. And bot of them has to have an open channel on both cryptocurrencies. Also there needs to be a viable path between eachother on both networks.

THis is perhaps best demostrated with a diagram

Show image and explain

3.2.1 Off-chain implementation details

my implememtation of this used most of the same setup as the on-chain swap implementation, still used golang, still used the same libraries, still used the same nodes. A new addition was the lnd library. This is a golang specific implementation of the BOLT specification. Te BOLT specification is the official document detailing the protocol and software that drives the lightning network, I will come back to this shortly.

My implementaton does not properly connect to the lightning network, doing it that way would take to much time and it would be very likely to fail for a newcomer such as me. Instead I implementated my own proper payment channels, it could be seen as my own private lightning network. I did not use all the proper protocol stuff, as I was in controll of both nodes and I was in full control.

The payment channels were setup using proper contracts as defined by the BOLT specification. THis time everything was packed into one neat program, with proper datastructures for the simulated users and their channels. As before the program did not braodcast anything by it self, instead it printed the hex-formated transactions and the user can optionally braodcast them if they want to try something.

In my tiny network only Alice and Bob existed and they had two channels between them, one in bitcoin and one in litecoin. I implemented it so that they could make normal transactions in the payment channels. When I got tht to function properly I performed my first off-chain atomic swap, using the same method as I showed earlier. It worked just as intended. I tried all outcomes and it functioned as expected.

Chapter 4

Results & Proposal

I'd say the results I got were good overall. And I learned a lot doing it. Just as with the implementaion part of the presentation i will try to slplit up this section into the on-chain and the off-chain results and proposal.

4.1 On-chain results and proposal

After my experiments I'd say that on-chain atomic swaps have the following traits:

- Easy to understand
- Easy to perform
- However, very time consuming
- And in the futur might be very expensive.

Once you have studied Bitcoin and surounding technologies constructing atomic swap contracts and transactions is fairly trivial. And should be very easy to implement in wallet software. They do however have pretty major flaws, unrelated to their safety ad clearity, and that is time and cost. As it is now the available space in blocks is limited and for a transaction to be included in a block it costs a fee based on transaction size. Right now the average transaction fee to be included into the next 6 blocks is about 29 cents. This fee has been known to grow when the traffic is high. So perofming an atomic swap could be seen as being fairly expensive depending on the time of transaction.

What also has to be taken into account is the time it takes. Under the example Bitcoin to liteocin which has the average blocktimes 10 minutes and 2.5 minutes the shortest possible time to perform an atomic sawp is around 25 minutes, as at the very least two blocks are required to pass on both chains. The maximum amount of time an atomic swap can take depends on the timelocks the participants agreed upon. In my implementation the timelocks were 24 and 48 hours so the swap could take up to 48 hours to complete. This however assumes that one of the participants willingly waited more time than neccessary.

4.1.1 proposal

My proposal is fairly simple, either extend or build a brand new wallet software with the following new traits:

- Construction of the new contracts. Obviously
- THe active swaps needs to be kept track of somehow, easiest way is to extend the already existing databases with new tables or key-values etc.
- Monitoring function, if the counter part is not revealing the proper information the wallet needs to be able to monitor the blockchains for the transactions and relevant

data. Looking for it manually is too time costly and complicated to be considered.
- the Wallet needs to support all the different cryptocurrenies that might be swapped.

In general I recommend that each participant has their own nodes running on the cryptocurrencies that they want to exchange. If you rely on others for that data there is a risk that you could be tricked.

4.2 Off-chain atomic swaps

I found the results from the off-chain atomic swaps to be much more promising. Under the assumption that no channel needs to be closed the fees are very tiny, and because each peer can negotiate with the other even in the case of a non-cooperative participant. The most interesting result that becomes clear with the usage of the regular htlc is that the oter peers on either side of the network, (with the exception of Alice and Bob), do not need to be part of the atomic swap in any way, for them the swap appears like a regula multi-hop transaction. This is very promising as no seperate netowrk is needed specificly for swaps. Instead the the two swapees need special software and can use the main lightning networks for both crypto-currencies. my proposal relies on the extensible BOLT-specification and the plugin system reasently added to the lightning node implementation c-lightning.

in the BOLT-specification they have intentionally left spaces open in flags and protocol messages, this space goes under the "its ok to be odd" rule. Meaning flags and protocol messages with odd positions or numbers are optional, and can be filled with functionality that is not officially supported by the lightning network. My proposed solution involves adding a new flag that signals to others in the network that your node can, and is willing to perform atomic swaps.

The proposed plugin would add this new flag, and it would handle atomic swap transactions a bit differently than normal. If you have agreed to do a swap with a certain exchange rate etc the plugin will keep track of this, along with the pre determined pre-image hash used in the swap. When a new htlc arrives that matches one of the saved pre-image hashes the plugin knows to handle it differently compared to a regular muti-hop transaction. Instead it will send a htlc contract on the other chain back to the original sender, using the same pre-image hash and continuing on the same decreasing timelock, as seen in the previous explantation.

This method of doing it is very non disruptive as the othe rnodes in the network remian entirly uneffected.