Bitcoin & Ethereum Cross-chain Atomic Swap

A Trustless Method of Exchanging Bitcoin For Ether Between Two Peers

Bachelor's Thesis submitted to the

Computer Science Engineering, Software and Multimedia developement orientation of the
Haute Ecole Arc Ingénierie (HES-SO), Switzerland

in partial fulfillment of the requirements for the degree of

Bachelor of Applied Science in Computer Science

presented by

Luca Srdjenovic

under the supervision of

Prof. Ninoslav Marina

co-supervised by

Thomas Shababi

July 2019

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Luca Srdjenovic
Neuchâtel, 26 July 2019

Abstract

Atomic swaps are practical for exchanging different cryptocurrencies in avoiding any trusted third-parties. This project shows a swap between Bitcoin and Ethereum blockchain using payment channels tools like hashlock or timelock. When the protocol is followed by the both participants, it guarantees the swap without any risk. In the opposite, there is no scenario where someone can control both coins.

Keywords : `Bitcoin, Ethereum, Atomic Swap`

Acknowledgements

Thank you

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Bitcoin, a peer-to-peer network

Chapter 3

Ethereum, A Decentralised Computing Platform

3.1 Test

Chapter 4

Atomic Swap, A Method of Exchanging Different Cryptocurrencies

Definition : `Atomic Swap is the process of peer-to-peer exchange of two cryptocurrencies between two parties, without using any third-party service like crypto exchange.`

In few explication, an atomic cross-chain swap is a smart contract distributed where two parties or more exchange two cryptocurrencies across different blockchains. It is called cross-chain because you are no longer dependant on the blockchain. An atomic swap protocol guarantees if both parties follow the protocol, then all swaps take place. But if one of the two parties deviates from the protocol, then no conforming party and the no coalition produce automatically the cancel of the swap. At any moment, no one can control both coins, hence no coalition has an incentive to deviate from the protocol.

4.1 Atomicity

`Atom` comes from Greek and means 'a' -not/un, 'tom' -cut, in other word, no divisible or cuttable. It means that an atomic transactions cannot be splittable into parts. We use the familiar expression `all or nothing` in atomic where it is the same applied concept in bitcoin. For example, Alice pays Bob in one transaction, they all know that either Bob will be paid or either bob won't. There is only two ways, the transaction is confirmed or not but there is no way for having an half-confirmation. That's the reason why the atomicity is fundamental in atomic swap, to protect both parties, there must be no scenario in which one part can control both coins at the same time.

An other example, no atomic transaction for illustrating is when Alice wants buy something in a web store. First, she needs to transfer the money to the site and then waits for the store send her the object back. Here there always is a chance that Alice doesn't get her purchase.

4.2 Difference with Payment Channels

In Bitcoin, Payment Channel is class of techniques designed to allow users to make multiple Bitcoin transactions without committing all of the transactions to the Bitcoin block chain. In a typical payment channel, only two transactions are added to the block chain but an unlimited or nearly unlimited number of payments can be made between the participants.[1] It is faster, cheaper transactions between parties because each transaction doesn't need to be written to the blockchain. Therefore there is only the net result of multiple transactions.

Atomic swap is not a payment channel but uses tools of it like Hashed Timelock Contracts (HTLC), a technique that can allow payments to be securely routed across multiple payment channels that we describe below (see chapter 4.3). It is a concept from the Bitcoin community that is used in the Lightning Network.

4.3 Security by Hashed Timelock Contracts

Atomic swaps uses HTLC (Hashed Timelocke Contracts), which are part of the scripting language used by most major cryptocurrencies in existence right now. Both parties involved in a cross-chain transaction submit their individual transactions to the appropriate blockchain.

HTLC is a kind of smart contracts that allows to eliminate counterparty risk using tools like hashlock and timelock. It enables time-bound transactions between the two parties. A time-bound means when a recipient at the other end of the transaction is required to acknowledge the transaction, the person needs to provide a cryptographic proof. The person also needs to provide that cryptographic proof within a time-frame. In case of failing so will automatically make the transaction null and void. In practical terms, this means that recipients of a transaction have to acknowledge payment by generating cryptographic proof within a certain timestamp. Otherwise, the transaction isn't valid. The cryptographic proof of payment that the receiver generates can then be used to trigger other actions in other payments, making HTLC a powerful technique for producing conditional payments in Bitcoin.

There are many benefits for HTLC :

1. It prevents the person who is making the payment from having to wait indefinitely to find out whether or not his or her payment goes through.
2. The person who makes the payment will not have to waste his or her money if the payment is not accepted. It will simply be returned.
3. The recipient actually helps to validate the payment on the blockchain because cryptographic proof of payment is required for the recipient to accept the payment.
4. The hashes that are created for the HTLC can be easily added to blockchains.
5. The structure of the method allows the people sending and receiving the payments do not have to trust each other or even know each other to make sure that the contract will be executed properly. In other words, each party is protected from counterparty risk.

[1] Micropayment channel: Bitcoin.org Developer Guide

To work, A Hashed Timelock Contract implements several elements from existing cryptocurrency transactions. The concept of signatures, HTLC uses multiple signatures that consists of using a private key and public key to verify and validate transactions. The main elements that make HTLC a powerful method are the concept of `hashlock` and `timelock`.

4.3.1 Hashlock

A hashlock is a type of encumbrance that restricts the spending of an output until a specified piece of data is publicly revealed. Hashlocks have the useful property that once any hashlock is opened publicly, any other hashlock secured using the same key can also be opened. The hashlock is a scrambled version of a cryptographic key generated by the originator of a transaction.

4.3.2 Absolute Timelock

```
IF
    <provider pubkey> CHECKSIGVERIFY
ELSE
    <expiry time> CHECKLOCKTIMEVERIFY DROP
ENDIF
<client pubkey> CHECKSIG
```

Listing 4.1. Example of locking script with CheckLockTimeVerify.

4.3.3 Relative Timelock

The second one is CheckSequenceVerify (CSV). It is not dependent on time. Instead, it uses the number of blocks generated as a measure to keep track of when to finalize a transaction.

```
IF
    <provider pubkey> CHECKSIGVERIFY
ELSE
    <expiry time> CHECKSEQUENCEVERIFY DROP
ENDIF
<client pubkey> CHECKSIG
```

Listing 4.2. Example of locking script with CheckSequenceVerify.

Chapter 5

Protocol

We describe a protocol for an on-chain atomic swap between Bitcoin and Ethereum, but the protocol can be generalized for Ethereum and any other cryptocurrencies that fulfill the same requirements as Bitcoin (e.g. LiteCoin), see **??**. This protocol is heavily based on the `BIP-199` (Bitcoin Improvement Proposal (BIP)) [Bowe and Hopwood, 2017] for the Bitcoin part. For Ethereum the concept is roughly the same but with less prerequisites than Bitcoin. For sending funds, each participant must generate a specific address to lock fund on each chain (cross-chain) where each other party can take control of the funds from the other chain (swap) only.

5.1 Limitations

The most important process of the protocol is the `liveness`. Liveness means that participant must be online for respecting the protocol (at least one participant is still online). In the worst scenario where someone doesn't follow the protocol, it can happen the coalition end up and loose the funds. This happen only if a party is not remained online during the swap or it has not claimed the funds in time.

In an other side, there is an other factor to take on board which is the `Fees`. Each blockchain have different fees because there are built with different internal parameters and transaction complexity. It is also due to a factor, the blockspace that depend of the demand. In this project, we use the Bitcoin Blockchain like a tool, more precisely, we use some advanced features that increase the cost of the transaction for bitcoin side. In general, the transaction is more expensive on Bitcoin than Ethereum, because Ethereum the cost transactions doesn't depend by the user.

The difficult problem with cross chain swaps is the off chain coordination required to have the two parties meet and agree on conditions. This consist to an accord between the two peers by the speed of the protocol (i.g. to considerate that a confirmation is confirmed) but the speed is influenced with the slowness and a number of confirmation required for validating a confirmation in each blockchain side. The protocol is slow but it can be extended by way of setups. The only things we can change from the setups is the ranges of fees that can consume but in any case we cannot deviate the worst scenario that consists to have an amount of fee in each chain.

5.2 Scenario

Alice and Bob want to exchange 1 Alice tokens for 10 Bob tokens. The problem is that they are not in the same blockchain, Alice token is defined in Bitcoin blockchain , whereas Bob token is only present Ethereum blockchain.

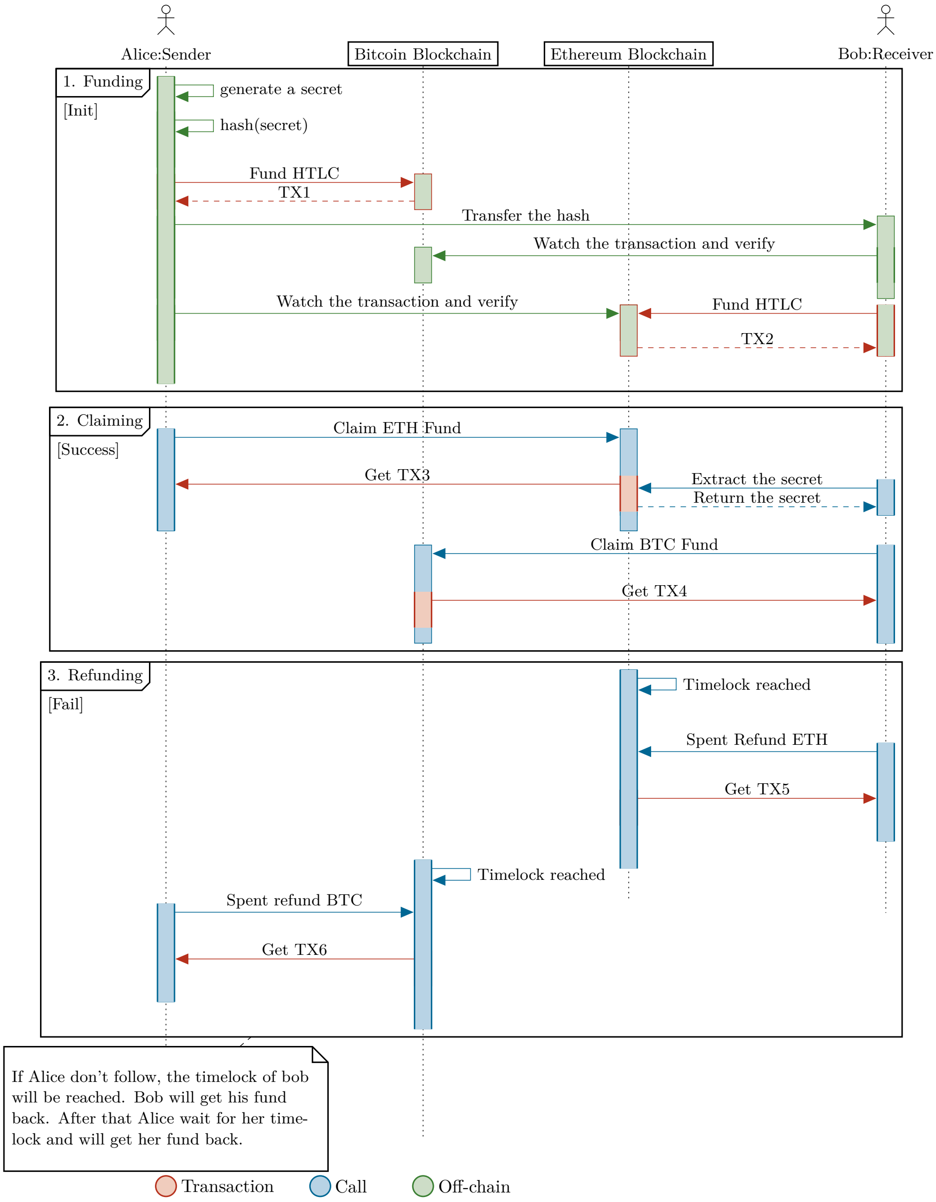

Figure 5.1. Sequence of atomic swap protocol.

Let's see the process in the figure 5.1 :

1. Alice generates a random set of bytes called value or preimage. The proof should

have a size of 32 bytes.
2. Alice hashes the obtained proof to generate the secret.
3. Alice is the instigator of the swap, she starts by initiating the locking script, TX1 in Bitcoin chain.
4. Upon doing this, Alice broadcasts TX1 to the Bitcoin chain and transfer the secret to Bob.
5. Bob defines locking script transaction TX2 to Ethereum using the hash.
6. Only Alice can unlock the ETH in this address because she has the value which generates that particular hash. It's the claiming transaction.
7. Alice can get her ETH by signing a transaction for Bob's contract address and Bob can retrieve the BTC by signing a transaction TX3 for Alice contract address.
8. When Alice signs Bob's contract address with the value, she unlocks the address and reveals the value to Bob as well.
9. Bob, now knowing the value, signs off the transaction TX4 for Alice's address and retrieves his BTC.

To summarize the process , the scenario describes the participants and their incentives. `Alice` the sender owns Bitcoin (BTC) and Bob the receiver owns ether (ETH), they want to swap funds. Alice and Bob have already negotiated the price in advance and are agreed (i.e. amount of bitcoin for amount of ether to swap). They are only two possible ways of execution path for both parties :

- `The protocol succeed` - Alice get her ETH and Bob his BTC.
- `The protocol failed` - both parties keep their fund (they will lost some amounts because they need to pay some fees for each transaction).

5.2.1 Successful swap

For having a successful swap, both parties must follow the protocol. They will be four transactions in total, 2 transactions for Bitcoin blockchain and 2 transactions for Ethereum blockchain :

1. Lock the funds in Bitcoin and make it ready for the swap.
2. Lock the funds in Ethereum and make it ready for the swap.
3. Unlock the funds in Ethereum.
4. Unlock the funds in Bitcoin.

When participants unlock the funds, they take control of the output of the contract in the other chain. here is the most simple and optimal way to perform the protocol. Here, no timelock is required but both participant must care about the minimum number of transaction and for the minimal transaction, the funding transaction (locking fund). Confirmations vary between each chain, so it needs to be considered if both parties are expecting the funding transaction to be considered final and are sure to keep going the protocol.

5.2.2 Swap aborted

The swap is aborted only if one party wants not to continue the process. To get the refund of the locked funds, Alice or Bob must wait for the timelock is reached. When Alice starts, there are no ETH but only BTC locked into a contract. If bob doesn't follow, so Alice wait her timelock and after that she can spent refund. The length of time on each lock is important to ensure that the game can only be played fairly. Alice's time lock should be longer and Bob's lock should be much shorter. This is because Alice knows the hash lock secret and therefore has a major advantage. It is very important because if Alice's timelock had the shorter refund time, Alice could wait until that time expires, refund herself the Bitcoin and after that then enter the secret preimage into Ethereum to claim the ETH that Bob sent. Alice would have both coins and Bob would loose his ETH.

5.2.3 Worst scenario

There is possibility that the protocol can be broken again if a party doesn't follow the rules from the Bitcoin part. If the swap process succeed with Alice claiming ETH funds and Bob doesn't claim his BTC fund before the Alice's timelock, then Alice can spent her refund as soon as her timelock is reached. It will conclude that Bob would lost his funds and Alice would get both coins. In Ethereum this can't happend because when the timelock is reached, claim fund are automatically blocked and Alice cannot claim the fund, only Bob spent the refund to avoid that situation. To resolve this problem, we must implement a protocol that force Bob to be not offline or compensate Bob if Alice doesn't follow correctly the protocol.

5.3 Prerequisites

In the chapter 5.2, we describe the conditional process that must be followed to guarantee a swap with atomicity. Bitcoin has a small stack-based script language that allows for conditional execution and timelocks. Whereas Ethereum use the programming language that allows hashing and timelocks too. The challenge is then to implement the BIP-199 in Ethereum.

5.3.1 Bitcoin

The bitcoin transactions in this protocol use Segregated Witness for making broadcast. For any other cryptocurrencies with a bitcoin style UTXO model (e.g. Litecoin), these requirements must be fulfilled for having the same compatibility with this protocol (i.e. Bitcoin Cash is not compatible.)

Pre-image

Generation of a valid pre-image $\alpha \in \mathbb{Z}_{2^{256}}$ of 32 bytes size to a given $h = \mathscr{H}_{256}(\alpha)$ where $\mathscr{H}_{256}$ is the `SHA256` algorithm.

Public key hash

For a public key Q to a given $h_Q = \mathscr{H}_{160}(Q)$ where $\mathscr{H}_{160}$ is the `SHA256` follow by the `RIPEMD-160` algorithm. h_Q is the version of Q that is given to other participant so that they can send it bitcoins. It's shorter than the original public key, and it may provide an extra layer of security for the bitcoins compared for giving the public key direct.

Hashlock

Hashlock is for revealing the secret to the other participant. It is a primitive that includes a value to reveal some data (pre-image) that is associated to given hash and handle the spent the HTLC.

Timelock

The timelock is to enable a execution paths that is predefined by an amount of time. This amount of time is expressed in a number of block `nLocktime` where $t = nLocktime$. We use number of block instead of the amount of time in second for avoiding a problem called `leap second`.

Multi signatures

The signatures of both participants are required for creating HTLC only accessible by them if they agree.

5.3.2 Ethereum

Ethereum doesn't use the same Model as Bitcoin (UTXO) but is based on `Account Model`. Every cryptocurrencies that with this style model can fulfilled there requirements for having the same for having the same compatibility with this protocol. In comparison to Bitcoin, Ethereum use smart contract that handles also timelock and hashlock. However, we doesn't need the `Public key hash` for the verification. We use instead, the `msg.sender` from Smart contract that allows verification.

5.3.3 Elliptic Curve

Bitcoin and Ethereum do use the same elliptic curves. They use the `secp256k1` curve from Standards for Efficient Cryptography (SEC) with the Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm. The curve is described as follow :

$$
\begin{aligned}
&p : \text{a prime number; } p = 2^{256} - 2^{32} - 2^{9} - 2^{8} - 2^{7} - 2^{6} - 2^{4} - 1 \\
&a : \text{an element of } \mathbb{F}_p;\ a = 0 \\
&b : \text{an element of } \mathbb{F}_p;\ b = 7 \\
&E : \text{an elliptic curve equation; } y^2 = x^3 + bx + a \\
&G : \text{a base point; } G' = \\
&(\texttt{0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,} \\
&\ \texttt{0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8})
\end{aligned}
\tag{5.1}
$$

5.4 Hashed Timelock Contract

The overall protocol is as follow : Alice moves her bitcoin into an address where each participant controls a type of transaction using Bitcoin scripting language. Bob do the same into a Ethereum Smart contract that is then used to reveal the secret depending of

Alice who claims the ether. Bitcoin and Ethereum transactions are designed in such a way that if a participant follows the protocol he can't terminate with a loss. If the deal goes through, Alice spends the ether by revealing the secret, thus allowing Bob to spend the locked bitcoin. If the deal is aborted, Bob spends the ether after the second timelock,thus allowing Alice to spend the bitcoin after the first timelock. In both cases, the participants must add transactions fees. The full protocol is described in Table 5.1.

Alice (BTC→ETH)		Bob (ETH→BTC)
	Initialization phase	
$k_A \xleftarrow{R} [1, p-1]$		$k_B \xleftarrow{R} [1, p-1]$
$Q_A \leftarrow k_A G$		$Q_B \leftarrow k_B G$
$h_A \leftarrow \mathscr{H}_{160}(Q_A)$		$h_B \leftarrow \mathscr{H}_{160}(Q_B)$
$s \xleftarrow{R} \mathbb{Z}_{2^{256}}$		
$h_s \leftarrow \mathscr{H}_{256}(s)$		
	$\xleftarrow{\langle \text{send } h_B \rangle}$	
	Lock phase	
create BTX_{lock} $\langle h_A, h_B, h_s, t_0 \rangle$		
sign BTX_{lock}		
broadcast $\text{BTX}_{lock}^{signed}$		
	$\xrightarrow{\langle \text{send } h_s, \text{ watch BTX}_{lock} \rangle}$	
		verify $\text{BTX}_{lock}^{signed}$
		create ETX_{lock} $\langle h_s, t_1 \rangle$
		sign ETX_{lock}
		broadcast $\text{ETX}_{lock}^{signed}$
	$\xleftarrow{(\text{watch ETX}_{lock})}$	
verify ETX_{lock}		
	Swap phase	
unlock ETX_{lock} $\langle s \rangle$		
broadcast ETX_{unlock}		
	$\xrightarrow{\text{watch ETX}_{unlock}, \text{ extract } s}$	
		spent BTX_{lock} $\langle s \rangle$
		broadcast BTX_{unlock}

Table 5.1. Full protocol of cross-chain atomic swap between Bitcoin and Ethereum with Alice and Bob with initialization, lock, and swap phases.

5.4.1 Time parameters

$$\begin{aligned} t_0 &: \text{ Alice's timelock} \\ t_1 &: \text{ Bob's timelock} \end{aligned} \tag{5.2}$$

We use two timelocks t0 and t1 that are defined during lock swap. t0 sets the time window during which it is safe to execute the trade. When t0 is passed, the refund may start. t1 sets the response time during which Alice is required to react and reveal her preimage to get her ether. After t1, Bob can get his ether back and allow Alice to redeem her bitcoin.

5.4.2 Bitcoin Script

Swaplock

Pay To Script Hash (P2SH) is used to lock funds and defines the two base execution paths :

1. swap execution [success].
2. refund execution [fail].

The script is defined with Bob's h_B public key hash, Alice's h_A public key hash and the preimage h_s in the Listing 5.1:

```
OP_IF
    OP_SHA256 <h_s> OP_EQUALVERIFY OP_DUP
    OP_HASH160 <h_B>
OP_ESLE
    <t_0> OP_CHECKSEQUENCEVERIFY OP_DROP OP_DUP
    OP_HASH160 <h_A>
OP_ENDIF
OP_EQUALVERIFY
OP_CHECKSIG
```

Listing 5.1. Swaplock script.

The Swaplock is executed when OP_IF reads a true value from the stack. It expects a secret value, an ECDSA signature and the Public Key Hash (PKH). It hashes the secret and checks that it matches a given hash, then it checks PKH followed by the signature against the given public key.

Claim Fund

Bob takes control of bitcoin in using the pre-image s and his public key hash h_B from Alice to redeem the `Swaplock P2SH`. To redeem the HTLC this way, Bob use the following script in the input of a transaction::

```
<sig_B> <h_B> <s> OP_TRUE
```

Listing 5.2. Bob's script signature

Spend Refund

With this contract Alice can spend this output with her public key hash h_A after the timelock t_0 with the script signature :

```
<sig_A> <h_A> OP_FALSE
```

Listing 5.3. Alice's script signature

5.4.3 Ethereum Smart Contract

Ethereum doesn't use script language Bitcoin but the programming language for the smart contract. The smart contract allows to create functions from the uml diagram in figure 5.2 :

Function lock()

Function that will create a contract with all the prerequisites and lock it with the address of the sender `Bob` and the address of the receiver `Alice`.

Function unlock()

Function when it is called, check if the pre-image s is correct and then transfer the fund to `Alice`.

Function refund()

Function when it is called, check if the timelock t_1 is reached and then transfer the fund to `Bob`.

5.4.4 Transactions

All the transaction are described in the figure 5.3.

Funding transaction

The funding transaction is the transaction sending fund to the contract address. BTX$_{lock}$, bitcoin transaction with 1 or more inputs from Alice and the output (vout) to the `Swaplock P2SH`. ETX$_{lock}$, ethereum transaction from Bob that sends funds to the Smart Contract address.

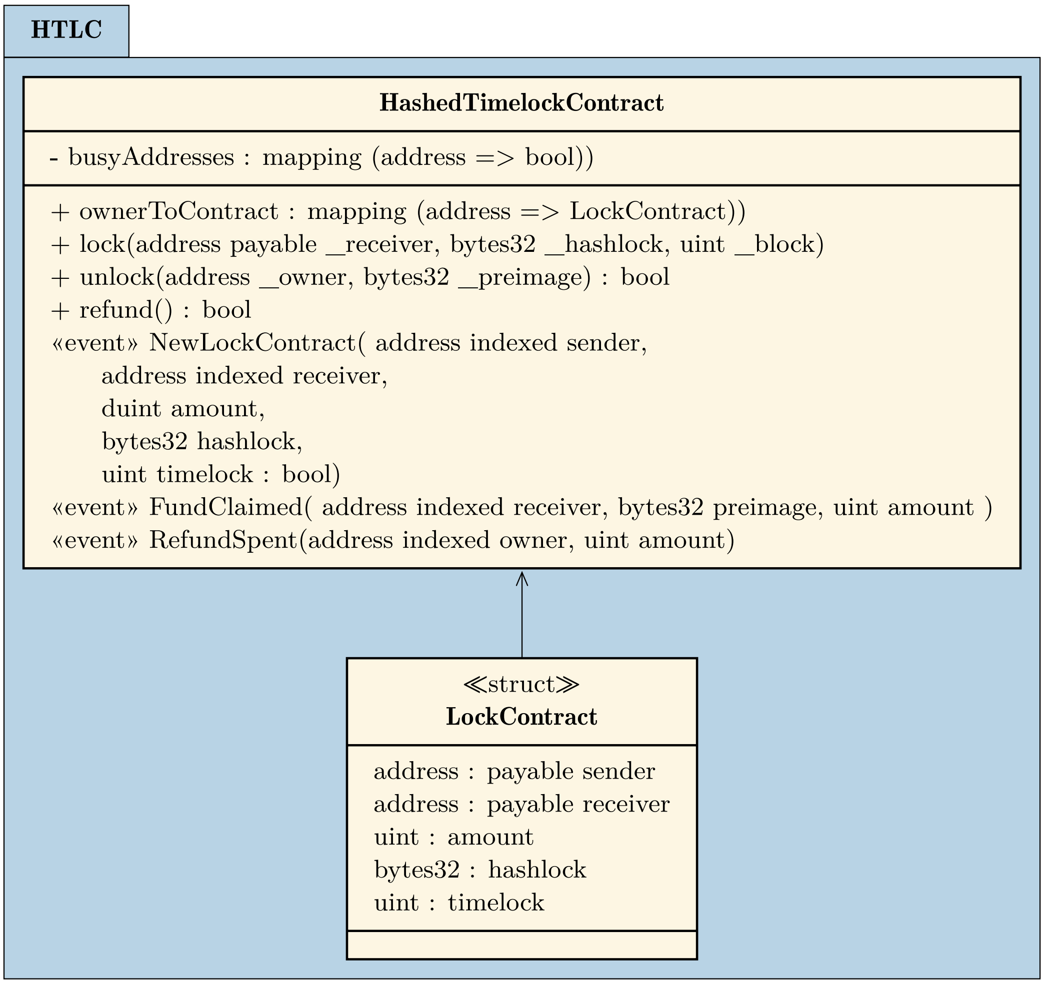

Figure 5.2. UML class diagram of the `Smart Contract` reference Implementation in Ethereum.

Claim Transaction

The claim transaction is a transaction that allows the sender to spend the funds. BTX_{unlock}, bitcoin transaction with 1 inputs consuming `Swaplock P2SH` (BTX_{lock}) and 1 output vout to Bob. ETX_{unlock}, ethereum transaction from the `Smart Contract` that call the function `unlock` to send the funds to Alice.

Refund transaction

The refund transaction is a transaction that allows the sender to abort the swap and get his funds back. BTX_{refund}, bitcoin transaction with 1 inputs consuming `Swaplock P2SH` (BTX_{lock}) and 1 output vout to Alice. ETX_{refund}, ethereum transaction from `Smart Contract` that call the function `refund` to send back the funds to Bob.

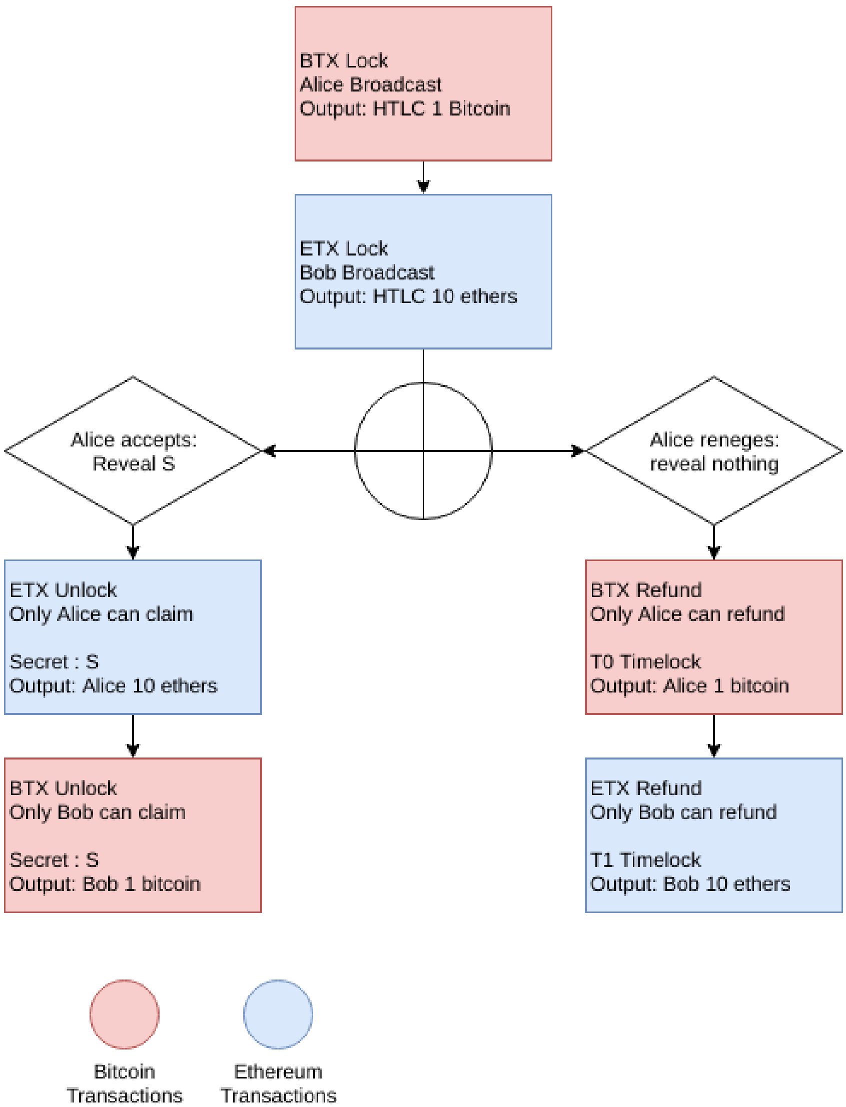

Figure 5.3. List of transactions.

Chapter 6

Implementation in Bitcoin with TypeScript

For this project, we describe a very simple method of constructing and executing smart contracts that sacrifice some privacy and potentially some security. The implementation is spread into five main components (i) a `generation of a wallet`, (ii) `how estimate the fees in Bitcoin`, (iii) `implementation of the funding transaction`, (iv) `implementation of the claiming transaction`, and (v) `implementation of the refunding transaction`. Noted that the current implementation is NOT production ready. This implementation is a more a educational work than an implementation for production and needs to be reviewed and tested more thoroughly before being used in production.

This chapter refers to the implementation available on GitLab at `https://gitlab.com/Skogarmadr/atomic-swap/tree/master` at the time of writing. The sources may evolve after writing this report, to be sure to read the latest version of the code check out the sources directly on GitHub.

6.1 Configuration

The project employ NodeJS with modern Javascript, for the simplicity of use with the installation of dependance modules. The library used is `bitcoinjs-lib` to manage all parts of Bitcoin. The incentive of this library are, (i) compatible with Typescript, (ii) still active, (iii) has a large number of contributors and releases and (iv) has a folder full of clear examples.

There is `config.json` file that contents all the data needed for the environnement works (i.e. Alice and Bob entropy for getting back the key pair). It is important to note that in production must saved only in client side because it is private data. Note also that the project work with the `Tesnet Network` because he is still in development and not testing for the production.

```
{
    "alice": {
        "entropy": "d399c2cbabdc9fa8790ec5111bb05da6",
        "phrase": "squeeze sock real fish size stage tomorrow suffer
            baby talk blast erupt",
        "seed": "4973ee8e81e0047a633f3b0d3cec61ec3b028846ffb648f509d5
            5be97e72a2e502e35148b463770cea5394bb7fe53a72344f7b5ffe08f
            3b208c27adb33f95d92",
        "prvWIF": "cPmUdWJYPESULvX3s2tFBfwJ5FWiWHm2wcx8pmiPiK6a2
            yKsrrb6",
        "xprv": "tprv8ZgxMBicQKsPd4RZif2NZzSMcFYdtEbCL8nSovoBYzTobGy9
            Pyjx6kK1BhZjYGRyvoMgyshBSNBrAkaaax6fGF7YAoq2i74ZEdjR2
            NfVKcQ",
        "xpub": "tpubD6NzVbkrYhZ4WXTMcJgxyQ6UBH4a3Zn6uSPE6SqUyGGCRmDv
            2NZYHEvsMqqNbbzKJYh2Lacgh37J2EMbDrmxtBSZfP5hbQBRMmDfshtPH
            3G"
    },
    "use_testnet": true,
    "APINETWORK" : "BTCTEST/"
}
```

Listing 6.1. Example of a config file for Bitcoin.

6.2 Generation of a wallet

In this project we use always the same set of addresses, but in production the practice says that the user shouldn't reuse the same addresses. For that, we need a wallet. We decide to create our own wallet to see how a wallet is built from the beginning to the end. In order to do anything with Bitcoin, you need a private and a public key pair. The public key can be used by other people to send you Bitcoin, and the private key can be used by you to send Bitcoin to someone else by verifying who created the transaction.

We generate our own mnemonic and with this we create two different BIP32 Hierarchical Deterministic (HD) wallets for Alice and Bob, each containing one distinct ECDSA key pairs. From each public key is derived one set of Bitcoin address for each type of PKH output.

```
const bip32 = require('bip32');
const bip39 = require('bip39');
const crypto = require('crypto');

// 128 bit entropy => 12 words mnemonics
// Generate random entropy
const alice_randomBytes = crypto.randomBytes(16);
const bob_randomBytes = crypto.randomBytes(16);

const alice_entropy = alice_randomBytes.toString('hex');
const bob_entropy = bob_randomBytes.toString('hex');

const funding_path = "m/44'/0'/0'/0/0";

 // Get mnemonic from entropy
 var mnemonic = bip39.entropyToMnemonic(alice_entropy);

 // Get seed from mnemonic
 var seed = bip39.mnemonicToSeedSync(mnemonic);

 // Get BIP32 master from seed
 var master = bip32.fromSeed(seed, NETWORK);

 // Get child node
 var child = master.derivePath(funding_path);

 // Get child wif private key
 var wif = child.toWIF();

  // Get child extended private keys
  var childXprv = child.toBase58();

  // Get child EC public key
  var ECPubKey = child.publicKey.toString('hex');
```

Listing 6.2. Generation of a HD wallet.

```
[
  {
    "username": "alice",
    "wallet": [{
      "wif": "cPg4ssEMrxSELzpD6hK52y1tfPcYfJvU1R153HhP2yWoyUNihitK",
      "pubKey": "0307560f56d2652c309b732394fa1c460fae7231f12adb271532
          9241028cca888a",
      "pubKeyHash": "ea2da066b9fdadea872af0d4ac138b8c1d4181ae",
      "p2pkh": "n2sB58biJzTCpbtzPgdDvCPCSYbt9oRuCs",
      "p2sh-p2wpkh": "2MwfFe3fmjoe7AMEMTxcVZChMvKSJC9ocoB",
      "p2wpkh": "tb1qagk6qe4elkk74pe27r22cyut3sw5rqdwyq5dk6"
    }]
  },
  {
    "username": "bob",
    "wallet": [{
      "wif": "cUpa4EZsjN4YL25ZfgqT9LR3jRzwPrD2Hc7E8HnPizX2A83PhMno",
      "pubKey": "039d04be8039c20e3799af14a61c5cdc86d2103c4d45b80a7ac8
          f25b4834722475",
      "pubKeyHash": "a007223750ccd2cead48848a731f062839b1cc7a",
      "p2pkh": "mv76xQGhSj2CChqkAg9tcCuMNVDdqCRuhA",
      "p2sh-p2wpkh": "2N7PRqTkPDgtj2XQL3tXMZVQtHwLTs1KfnP",
      "p2wpkh": "tb1q5qrjyd6senfvat2gsj98x8cx9qumrnr6qdska7"
    }]
  }
]
```

Listing 6.3. Example of a wallet.

In the Listing 6.3, Wallet Import Format (WIF) is the version of the compressed private key. With that, we can get our Elliptic Curves (EC) key pair. With the key pair, we can derivate all data that we need (e.g. the address of each person).

6.3 Getting free Testnet coins

For making new transactions, we need bitcoin. Testnet Network is useful tool that offers Testnet Bitcoin, a valueless coin for testing our application without spend any money. For that, we need to get theses free Testnet Bitcoins. We use `faucet` to get theses bitcoin Testnet, they can be hard to found sometimes. The one we use is `https://tbtc.bitaps.com/` for the two reasons: (i) it give us 0.01 Bitcoin every 5 minutes and (ii) it is compatible with the Segregated Witness (SegWit) addresses.

6.4 Reading data from the Testnet blockchain

To read data from a public blockchain, we use a block explorer that is a third-party REST API instead of running our own node in local and explore it ourself. Here, for this project we use (https://chain.so/api because they have a solid API, a excellent documentation site, and you don't need an API key. This allows us to get e.g. the unspent transaction output for having got the balance of an address.d

```
async function getUnspentTx(address: string) {
    try {
      return $.get('https://chain.so/api/v2/get_tx_unspent/' +
          APINETWORK + address);
    } catch (error) {
    }
  }
```

Listing 6.4. Get Unspent Transactions Output from API.

6.5 Estimate fees

For building a valid transaction, there is an important part to not forget in Bitcoin transaction : the fee. The fee is important because if you set it too low then the transaction won't be attractive to miners, and might take a long time to appear in the blockchain or it might even never get accepted. This is not such a problem on the Testnet where there isn't so much traffic, but is a big issue for the Bitcoin main network. We set the fee with a REST API that supplies a list of fee recommended per byte of transaction data. We use always the fastest fee rates because we don't need to be precious with the satoshis of Testnet Bitcoin but we need include them for the practice.

We can practise setting a sensible fee using 21.co's recommended fees.

You can also choose halfHourFee or hourFee, but (a) this is the testnet so we don't need to be precious with satoshis, and (b) these are the recommended fees for the main network anyway, so entirely unrelated to the Testnet. We're just including them here for practice.

The result from 21.co tells you the fee recommended per byte of transaction data. I'm multiplying by the median transaction size (again for the main net, not the Testnet) for convenience, but if you were creating large transactions you'd probably want to work out how large your actual transaction is so that the recommended fees perform as expected.

6.6 Funding transaction

6.7 Claiming transaction

6.8 Refunding transaction

6.9 Pushing transaction into the network

Chapter 7

Implementation in Ethereum with Solidity and Web3x

Chapter 8

Observation

Test Web3 for testing.
Test [Buterin, 2013]. # Conclusion
Conclusion

List Of Abbreviations

BIP Bitcoin Improvement Proposal. 11

EC Elliptic Curves. 26

ECDSA Elliptic Curve Digital Signature Algorithm. 15, 18, 24

HD Hierarchical Deterministic. 24

HTLC Hashed Timelock Contracts. 8

P2SH Pay To Script Hash. 18

PKH Public Key Hash. 18, 24

SegWit Segregated Witness. 26

UML Unified Modeling Language. 20, *Glossary:* Unified Modeling Language

WIF Wallet Import Format. 26

Glossary

Segregated Witness Segregated Witness is an update to the Bitcoin software, designed to fix a range of serious issues such as solving transaction malleability, a well-known weak spot in Bitcoin software and improving scalability[**?**].. 14

unspent transaction output UTXO is an unspent transaction output that can be spent as an input in a new transaction.. 27

Web3 Web3 often refers to `web3js`, the Javascript implementation of the Ethereum JSON-RPC. It may also refer to other implementation in different languages. Overall it is the technology aiming to build the next and more decentralised version of the web 2.0 we know today. 31

Bibliography

Sean Bowe and Daira Hopwood. Hashed time-locked contract transactions, Mars 2017. URL `https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki`.

Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2013. URL `https://github.com/ethereum/wiki/wiki/White-Paper`.