Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

Grau en Enginyeria Informàtica

Final degree project

Bitcoin Payment Channels

Progress report I

Author:
David Lozano Jarque
NIU 1359958
uab@davidlj95.com

Tutor:
Jordi Herrera Joancomartí

Version 0.1
April 8, 2017

Contents

List of Figures

A few notes about this document

Because this first stage of the project was developed along Carlos GC[1], another student that is carrying out a very similar project, the document uses the first person pronouns in a plural form[2]. Despite that, and knowing that the final degree project has to be developed individually, all of the progress achieved has been done in an individual way, so the both of us have the exact same knowledge about the Bitcoin environment and have researched the exact same subjects until the writing of the present document and the collaboration is just to provide each other mutual help to understand the complex details that dealing with Bitcoin implies.

Also, and due to the complexity of the Bitcoin concept in depth, this document assumes some basic understanding of the Bitcoin cryptocurrency (despite it can be read by anyone and transmit the basic idea) and will skip several explanations about the some low-level details learnt during the project time lapse as the document is intended to provide a global view of the project development status rather than providing specific details of the tasks performed that include complex or difficult-to-explain Bitcoin special characteristics.

[1] Carlos González Cebrecos <carlos.gonzalezce@e-campus.uab.cat> `https://www.ccebrecos.com`

[2] The unique author of the document, notwithstanding the text is written in first person plural form, is exclusively the author that appears on this document's cover and not anyone else.

Introduction

In this first progress report, we will review the *Bitcoin Payment Channels* project development since the last report, the project initial report, listing the accomplished project goals and tasks, the changes made to the project planning and the problems and solutions implemented during this time to acquire and fulfill with success the project's tasks.

1.1 Brief summary of the current status

In the first place, the project planning changed due to mainly bad timing estimations (very optimistic timings in most cases), rearranging tasks to reduce complexity and improve readability and a few changes to improve visualization. Those changes were performed after the general and low-level Bitcoin cryptocurrency study, assigning more time to the payment channel transactions modelling and testing because of the lack of good documentation about the topic found after hours of searching over the network for good information sources and the time lapse between blocks (approximately 10 minutes[2]) so we can check if our transactions get accepted by the network consensus. Despite that, the project's development schedule has been respected and the project's goals until this report's drafting have been completed with success.

In the following pages, we will give detailed information about all the progress done withing this time lapse between the first report delivery and this report drafting, the changes we had to apply to our project schedule and predictions to the project progress based on our acquired experience.

Bitcoin cryptocurrency study

Following our tutor's recommendation, we started to study the concept and operation of the Bitcoin cryptocurrency and network using a breadth-first strategy. We started with the basic concepts (mainly cryptographical tools and techniques) about Bitcoin. Then, we studied the main concepts and ideas about how transactions are performed, blocks are created, and consensus is reached at a high level.

Finally, we started to research how all concepts are used in detail, by understanding the current Bitcoin protocol implementation byte per byte (focusing just on everything related to transactions, that will allow us to create the payment channels)

2.1 The goals

As the main goal for this stage of study and research, we had to understand at a byte level how a Bitcoin transaction is constructed, therefore understanding also the Bitcoin scripting language

2.2 The basics about Bitcoin

To understand how Bitcoin works, we enrolled in a *MOOC*, a coursera course recommended by our tutor[3]. In these course, we reviewed the tools and techniques, mainly cryptographical needed in order to have the main knowledges to understand Bitcoin, that we already learnt in previous University subjects. After that, a global vision of the idea of cryptocurrency and how Bitcoin puts it in practice is explained and later detailed a little bit more technically but without getting into implementation details. Politics, cryptocurrency and other subjects were skipped as didn't contribute any relevant information to our project. With these course finished, we accomplished the first tasks of learning the Bitcoin basics.

2.3 Bitcoin low-level understanding

Despite there are quite a lot sites with information about the Bitcoin cryptocurrency, the fact is that when dealing with Bitcoin in a low-level detail, information is scarcely found. Most of the information to understand the Bitcoin protocol implementation was found in a Wiki-like site called *Bitcoin.it* [4]. The rest of information was found in developer Q&A sites like *StackOverflow*[3] or the more Bitcoin-specific site *Bitcoin Stack Exchange*[4], and several forums and blogs (that will be properly cited in the report in the following pages and are also properly cited in the software code documentation)

2.4 Transaction low-level understanding

Once we understood how basic concepts were implemented in the Bitcoin protocol, we decided to take sample pay to public key address transactions from the main Bitcoin network (also known as the production network), as most of them are of that kind and regular (we could be unlucky and pick strange transactions from the *testnet*, the development Bitcoin network used for new features testing).

We chose to understand low-level, byte per byte, pay to public key address transactions first as they are the most simple transactions: they spend funds by signing using a private key and send funds by specifying (the hash of) a public key. We have to understand fully the most simple transactions to create complexer transactions with specific spend and payment conditions (also called *smart contracts*)

We studied the meanings and the encodings used (little endian most of them) of all the fields in a transaction and compared our knowledge with real transactions we took from the *mainnet* using a block explorer [5] [6].

The summary is that a Bitcoin transaction contains a version field, inputs field, outputs field, and a locktime field, with each field having a specific byte format and representation. The inputs field contain a list of inputs, where each of them references to the transaction (UTXO, unspent transaction output) that will be spent, its spent script (signatures and public keys in P2PKH cases) and a sequence field. In the case of the outputs, it specifies a list of outputs fields where each output contains a value and a script containing the conditions that the spender must satisfy to spend it (in P2PKH cases, the hash of the public key whose paired private key will sign the spending transaction).

[3] `https://stackoverflow.com`
[4] `https://bitcoin.stackexchange.com/`

The following picture illustrates a low-level transaction and its fields:

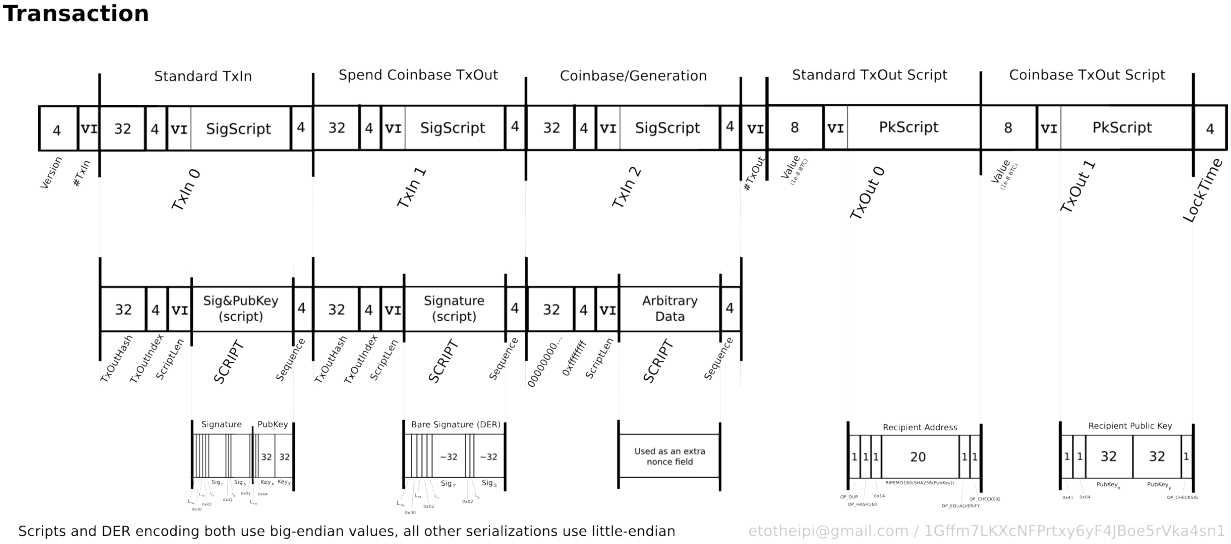


Figure 2.1: A Bitcoin transaction in low-level, byte detail[1]

Understanding the version, number of inputs and outputs, references to outputs in inputs and specification of output values was mostly a reading task to know the byte format and encoding from the *Bitcoin Wiki*[7]. Some fields like *sequence* or *locktime* were not researched depthly as were not relevant in that moment. The most difficult pieces to understand where the scripts, that specify how the transaction outputs can be spend and how the transaction inputs (that point to unspent outputs) get spent.

2.5 Script low-level understanding

To understand scripts, the *Bitcoin Wiki*[8] provided very useful information about them in low-level, despite omitting (or not specifying clearly) details like the *push data* opcodes. Several posts from forums, blogs and Q&A answer sites where used to complete the information[9,10]. The most difficult part was to understand how signatures are performed, since a new special transaction (copy of the transaction but removing some fields) has to be created to then sign it and set it into DER format in order to add a signature to a script[11,12].

To view scripts in a more graphical way, the online site *WebBTC* provided a visual script executor given a blockchain transaction[13] and *hashmal* project from *mazaclub* GitHub user provided a GUI to create and test scripts[14].

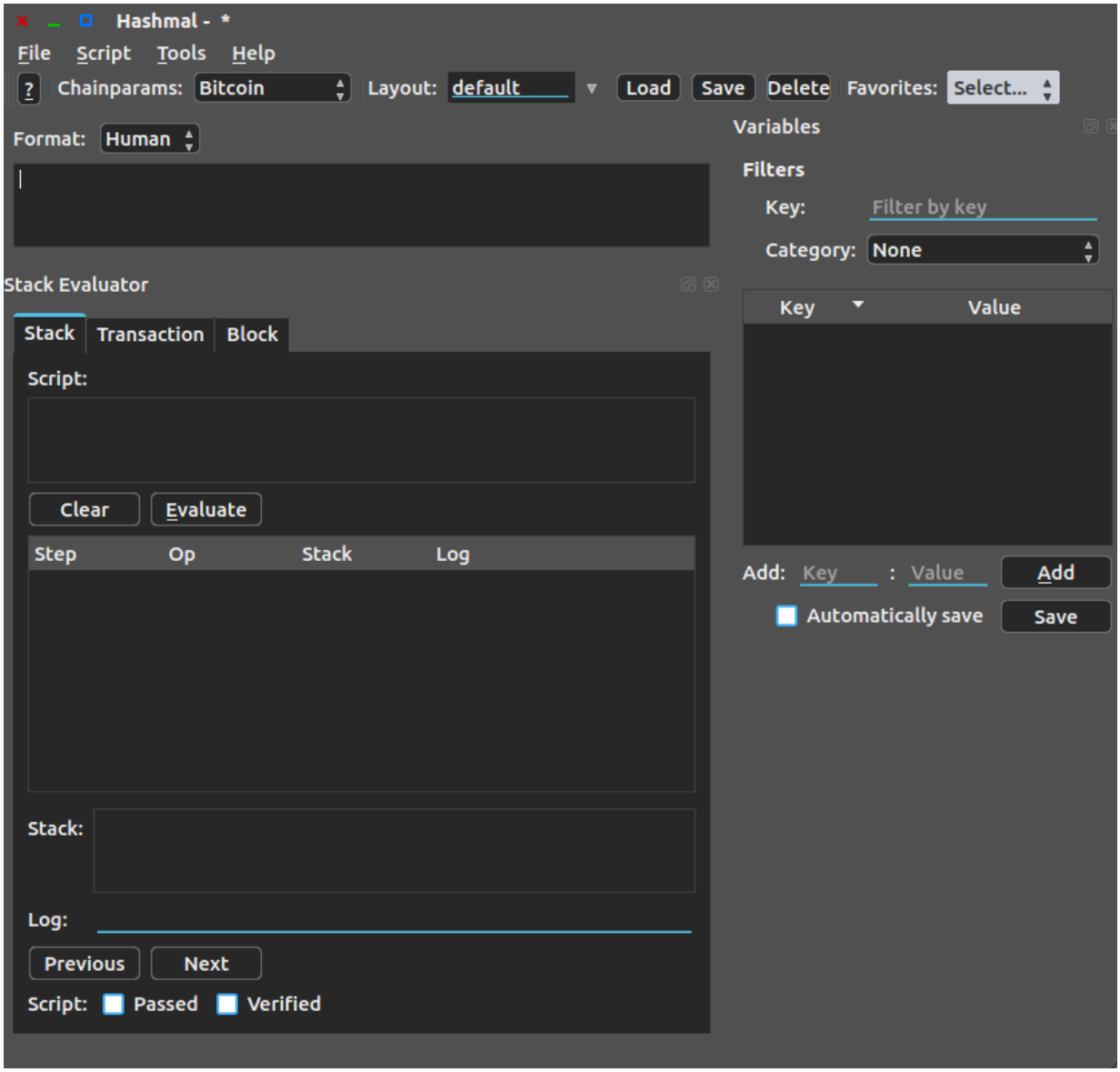


Figure 2.2: *Hashmal* script editor and executor screenshot

2.6 Conclusions about the research

Once the research was concluded and a raw transaction in bytes had acquired meaning for us, being able to decode most of the bytes in it, we concluded several points:

- **The Bitcoin and *smart contracts* potential**
  The fact that Bitcoin allows scripts to set how the funds of a transactions can be spent and scripts to spend those funds providing the conditions specified, has an

enormous potential as allows developing *smart contracts* that ables us to perform financial transactions with a huge set of possible spending conditions just limited to the scripting language limits, where those transactions will be protected cryptographically by a decentralized network of nodes that will check and validate that those conditions are accomplished. These idea and its potential has never seen before the appearance of Bitcoin.

- **Sparsed and few low-level documentation**
  Despite there's quite much information about the Bitcoin concept and high-level operation explanations, there are just few websites that explain how Bitcoin works at a low-level detail, explaining what each byte means inside a transaction. Even best way to understand some details is to look at the Bitcoin Core's C++ implementation[15], the widely used Bitcoin node (with some comments that try to make the code more readable). Developing an informative website or media that collects and explains Bitcoin from the highest to the lowest level would be a really interesting and valuable educational resource that could aim more developers to play with the Bitcoin technology

- **The need for software models and architectures (ie: a framework)**
  Due to the lots of concepts, ideas and dispersed items that play a role in Bitcoin (and just knowing and fully-understanding the transactions and therefore script pieces), defining a model (an object in OOP) for each item that is present in the process of a transaction generation can be a way to allow faster features development by abstracting hard low-level details so new developers don't get afraid to develop new Bitcoin features without spending hours to understand absolutely everything and losing time with technical, technological, programming-language details that don't provide any value or innovation. We'll try to create those models as much as our timing and project scope allows us to create them without delaying the schedule.
  In this sense, the alt-currency Ethereum[16] has made great progress.

The software framework

Once the research was full-filled and (most part of) transactions were understood at a byte low-level, we started developing our software.

3.1 The goals

The main goal for our software was to create simple P2PKH transactions in this first iteration of the development, to then be able to perform more iterations and add more complex transactions, until completing the project goal of creating the transactions needed to create and operate *Bitcoin Payment Channels*.

3.2 Technologies and environment setup

We decided to use *Python 3.6* because of our experience developing applications using this programming language and the ability it provides to focus on adding and implementing features quickly without having to worry much about low-level coding aspects like memory handling, centering the efforts in adding value to the application and therefore, to the project.

3.2.1 Libraries

After deciding the programming language, we searched for libraries that helped us performing complex operations like signatures, encoding operations, ...

We used the following ones:

- **python-bitcoinlib** [17]
  Provides handling of base58 encoding, scripts, public keys and data structures in general of the Bitcoin protocol

- **pybitcointools**[18]
  Library to perform common cryptographic operations in Bitcoin, providing an easy interface that converts the format of the input parameters automatically to avoid tedious format conversions

3.2.2 Version control system

We use Git, as mentioned in the initial report to handle the software versions and work in parallel. The source code of the project is publicly available[5]:

`https://github.com/uab-projects/btc-payment-channels`[6]

3.3 The architecture and design

As we said in the previous research chapter conclusions, there was a need of modelling the Bitcoin items in order to provide a better understanding and an abstraction to allow fast feature development without having to worry for low-level programming-language details. Following this non-functional requirements in aim to provide a better library for Bitcoin Python developers, we modelled all items that were in the scope of the project using OOP:

- **Address**
  An address allows to set how the funds must be spent in a transaction output, and can also contain public or private keys. Our model must provide an easy interface to create them using the items that they require, depending on the address type, that we'll also model. From an address, we have to be able to create automatically an output script if the address type matches (ie: does not contain a public / private key)

- **Script**
  A script must be able to contain a list of fields (opcodes an data fields) to be able to set the spend conditions or to spend some output script. Therefore we can model a basic script, input, output, redeem and payment scripts and use the factory design pattern to easily build the most known types.

- **Input**
  Contains the reference to the previous transaction, number of output, spending script, and sequence field

[5] Checkout `https://www.uab.codes` to stay informed about the latest releases

[6] Ensure to check the `development` branch as we don't spend time launching *releases* very often as the code it's in continuous progress

- **Output**
  Contains the value to spend and the script with the conditions to spend
- **Transaction**
  Contains all the transaction fields, version, inputs, outputs and locktime field

With all these items modelled and handled with easy, defining all formats of inputs and ouputs in an extensive in-code documentation, we provide a very good framework to work with and easily extend it using the items to create new transactions. We just have to have a way to transform them into bytes.

3.3.1 The serializable interface

To allow modularity and the easy combination of all the models, each of it must implement the serializable interface, that basically means that the object has to be able to be converted into an array of bytes (a *bytes* built-in object in Python), with the `serialize()` method. Optionally (as it's not a feature to accomplish our project main goals), also has to implement a `deserialize` method, to create a new object from an array of bytes.

This way, we can create several objects from our models and join them as if we joined arrays of bytes, but with the ease of creation of a built-in Python classes instantiation, friendly for developers.

3.3.2 The libraries' use

If all these models have been created, why have we mentioned the previous libraries? We have just coded the models and joined them together, but the algorithmic parts and complex operations have been delegated to those libraries, such as base58 encoding and decoding for addresses and ECDSA signatures. The rest has been coded by ourselves.

3.4 A basic transaction creation

Once all the models have been coded and its serialization implemented, we have tested our framework by creating a P2PKH basic transaction that transfers funds creating a signature and setting the public key (hash) to pay to.

After several ECDSA signatures format mistakes and misunderstandings, we succesfully

broadcasted a valid and confirmed transaction moving funds between ourselves.[7] The transaction was created before the milestone deadline scheduled, the 20th of March of 2017

[7]The transaction can bee seen in the following *testnet* block explorer using its `txId`: `https://tbtc.blockr.io/tx/info/258fb211724412d6ec6a531973c58233143e6ab355623658adc3164a5c70bd5b`

Unidirectional payment channel

Once we can create transactions and more important, scripts, in an easy way, we are prepared to design more complex scripts to create *smart contracts*. This means in the following iteration of development, we have to provide new models that allow the generation of *multisig* redeem scripts and other special conditions like *time / expire* conditions or *hashlock* contracts.

4.1 Understanding a payment channel

There's quite few information about payment channels over Bitcoin cryptocurrency online. We've used the knowledge our tutors have provided us and an article also referenced by them on the *Bitcoin Magazine* [19].

The basics of a payment channel is that there has to be a *commitment* or *funding* transaction to fund the channel and then make payments using those funds to generate transactions that share out the funds to give more funds to a party. After the channel operation, the channel must be closed with a *closing* transaction.

The idea is that those transactions that share out the funds can end in the blockchain (maybe after some time) and be valid, so that both nodes trust the transaction and can verify them using their nodes. But they don't get broadcasted to the network until the close transaction sets the final funds distribution, relieving the blockchain from storing those transactions and allowing to increase the speed of transactions between these parties as they can be generated and validated in much less time than the time it takes a regular transaction to appear on the blockchain and confirmed. At the time they exchange the payment transaction (that distributes the channel funds) the payment has been done as they know, can validate theirselves and trust that if the transaction ends in the blockchain would be valid.

The key is how to create those transactions in order that no attacks are possible (even DoS) to the channel by any of the parties by using cryptography and game theory

4.2 Modelling a unidirectional payment channel

Knowing that a payment channel implies the creation of scripts that ensure no attacks on the channel are possible and the parties play fair, the effort and problem is to create those scripts to be functional and secure at the same time. To speed up development, we will evade the malleability issues and assume the same scripts can be coded using `SegWit`, but without implementing them using `SegWit` to save time.

4.2.1 Schedule problems

At the time of writing this document, the schedule said we should have implemented those three scripts and transactions to allow *commitment*, *payment* and channel *closure*. Due to the lack of awareness and excess of optimism, we couldn't design all three scripts and we just modelled the first one. In the rescheduling chapter, more details are given about how the schedule has changed.

4.2.2 Commitment

The commitment for a unidirectional channel can be done in two ways: a transaction that funds the channel and expires at a certain time, returning the funds to the funder(s) to prevent the funds from being locked out if a party doesn't collaborate or create a simple funding transaction and after that a refund transaction that is valid after a certain time. The easiest is two create two transactions, but triggers problems of who signs the transaction first, trust problems in the timelapse the first transaction is created and signed and the refund is signed too, ... The first option requires a complex script but avoids such trust problems

Single-transaction commitment script

This transaction funding transaction must spend inputs owned by the parties of the channel and set the output conditions to a *multisig* output that requires both parties signatures to create transactions on, therefore creating the channel, or to return the funds after a certain time in order to prevent fund-locking if a party doesn't collaborate.

Creating a *multisig* script requires the comprehension of Bitcoin P2SH scripts, that as many low-level aspects of the Bitcoin subject, few information can be found over the Internet. Fortunately, a blogger and developer posted a really detailed and clear explanation about the subject[20]

Our `scriptSig` proposal for the payment transaction (whose partial hash, just of the redeemScript piece, would define the first commitment transaction using a P2SH).

```
OP_0 <sigA> <sigB> | OP_2 <pubKeyA> <pubKeyB> OP_2 OP_NOTIF <time>
    OP_CHECKLOCKTIMEVERIFY OP_DROP <PubKeyFunder> OP_CHECKSIG
```

Where `<PubKeyFunder>` would be the public key of the channel funder (we assume an easy case where just one of the parties fund the channel). In the case the channel expires, the `scriptSig` would change to spend the funds from `OP_0 <sigA> <sigB>` to `OP_0 <sigFunder> OP_DUP OP_OVER` so the *multisig* fails but doesn't triggers and error and then the time is verified and after that the signature is checked agains the funder public key. This script has been revised by one of our tutors and requires its implementation and further testing

Goals revision

As appeared on the initial report, the final goal is to implement a bidirectional payment channel based on the idea and protocol explained on the Christian Decker's paper [21]. The software that implements the payment channel must be able to create a payment channel asking the user for parameters that can be understandable for a person with basic Bitcoin knowledge about its operation.

Depending on the timing, that has changed due to optimistic estimations, we'll achieve a certain level of automation and ease to create the channel. We don't know how many iterations of software development (transaction model, script model, implementation and test) will take until the payment channel gets implemented and its operational. After that, the rest of cycles or iterations will be dedicated to automate and enforce the channel security.

5.1 Conclusion

Despite the new and advanced low-level knowledge of Bitcoin that has made us reschedule the timing due to optimistic timings, we still think that can achieve the main goal of creating an operational bidirectional payment channel using the Christian Decker's protocol. What will change is that the automation and ease of use of the channel maybe won't be the same as the expected in the initial report.

Methodology revision

We thought at the initial report to work using a CVS for our code that we'll keep using as provides extended control and integrity over our code and iterative development as provides features in short periods of time, but taking an adequate time to design, implement and test (from now on transactions for the payment channels) and reducing the iteration times to one week or less strictly to see results soon. What we've learned is that design (or transaction and script model or creation in these cases) is really important as if a mistake is made in that stage and then everything gets implemented, we'll realize the mistake while testing, having to change the design and losing much of the time of a work that in most cases won't be profitable. Therefore in these further iterations of the software development we'll increase time in design and testing and try to reduce it in implementation as the prior objective is to create operational transactions that create a payment channel.

6.1 Conclusions

We'll follow the same methodologies as have been useful to keep and maintain a rhythm to be on time in most of the time of the project. We'll just invest a few more time in design stage to prevent design errors to be discovered on the test phase and losing implementation work time, while also adding more time to the test phase as implementing scripts is not easy as far as we have seen. Also reducing the iterations to one week or less will try to provide more features in less time. We'll check in next report how this methodology has worked.

Schedule revision

As we said before, and once our knowledge of Bitcoin arrived at a low-level detail, we realized the time we estimated for script and transaction modelling was too optimistic so changes appeared in the planning from that task forward.

7.1 Changes motivation

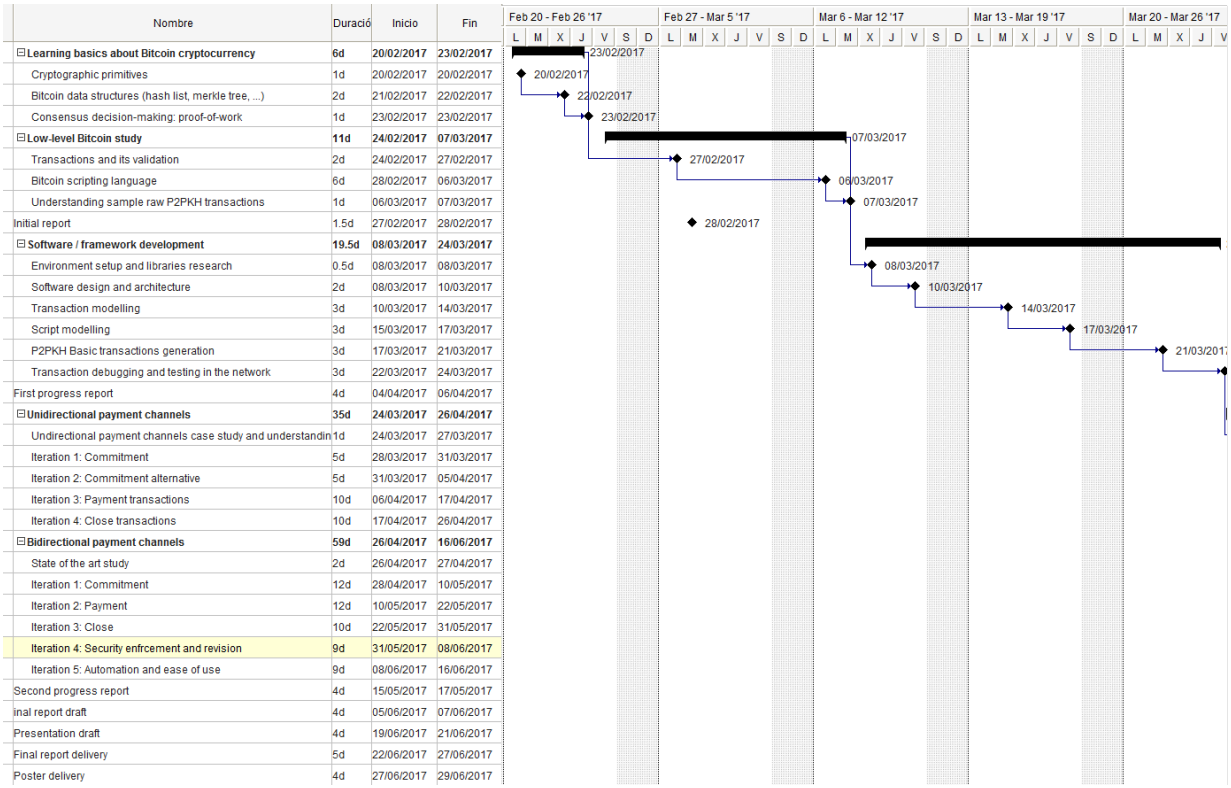
As appears in the previous methodology chapter, we'll dedicate more time to transaction modelling and script creation and to its testing and try to reduce the implementation time as with the framework created it will suppose to be simple and not require much lines of code. This way we want to achieve the goal by redistributing the time assigned to its task.

7.2 Other minor changes

In order to make the Gantt diagram simpler, we've reduced the iteration to design, implement, test and launch that mean to create, implement, test and broadcast transactions respectively. Therefore it's understood that in each iteration cycle, a different transaction will be developed, requiring 3 transactions modelling per channel (and that the unidirectional channel has to be implemented). Also several subtasks were reduced to a main task to reduce the resulting Gantt diagram size and iterations were defined to fullfill all diagram time lapse.

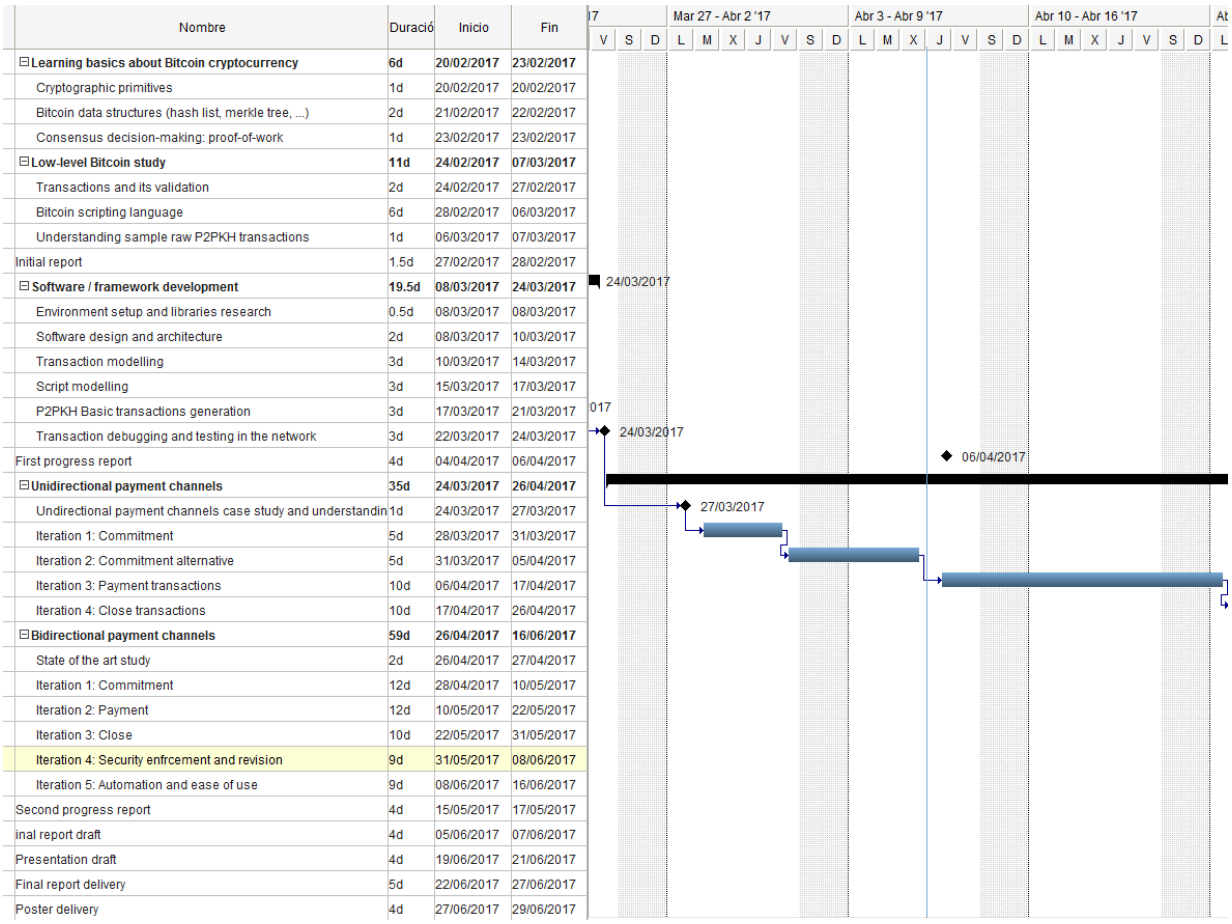
7.3 Resulting Gantt diagram

After performing the previously mentioned changes, we obtain the following new Gantt diagram:



Nombre	Duració	Inicio	Fin
Learning basics about Bitcoin cryptocurrency	**6d**	**20/02/2017**	**23/02/2017**
Cryptographic primitives	1d	20/02/2017	20/02/2017
Bitcoin data structures (hash list, merkle tree, ...)	2d	21/02/2017	22/02/2017
Consensus decision-making: proof-of-work	1d	23/02/2017	23/02/2017
Low-level Bitcoin study	**11d**	**24/02/2017**	**07/03/2017**
Transactions and its validation	2d	24/02/2017	27/02/2017
Bitcoin scripting language	6d	28/02/2017	06/03/2017
Understanding sample raw P2PKH transactions	1d	06/03/2017	07/03/2017
Initial report	1.5d	27/02/2017	28/02/2017
Software / framework development	**19.5d**	**08/03/2017**	**24/03/2017**
Environment setup and libraries research	0.5d	08/03/2017	08/03/2017
Software design and architecture	2d	08/03/2017	10/03/2017
Transaction modelling	3d	10/03/2017	14/03/2017
Script modelling	3d	15/03/2017	17/03/2017
P2PKH Basic transactions generation	3d	17/03/2017	21/03/2017
Transaction debugging and testing in the network	3d	22/03/2017	24/03/2017
First progress report	4d	04/04/2017	06/04/2017
Unidirectional payment channels	**35d**	**24/03/2017**	**26/04/2017**
Undirectional payment channels case study and understandin	1d	24/03/2017	27/03/2017
Iteration 1: Commitment	5d	28/03/2017	31/03/2017
Iteration 2: Commitment alternative	5d	31/03/2017	05/04/2017
Iteration 3: Payment transactions	10d	06/04/2017	17/04/2017
Iteration 4: Close transactions	10d	17/04/2017	26/04/2017
Bidirectional payment channels	**59d**	**26/04/2017**	**16/06/2017**
State of the art study	2d	26/04/2017	27/04/2017
Iteration 1: Commitment	12d	28/04/2017	10/05/2017
Iteration 2: Payment	12d	10/05/2017	22/05/2017
Iteration 3: Close	10d	22/05/2017	31/05/2017
Iteration 4: Security enfrcement and revision	9d	31/05/2017	08/06/2017
Iteration 5: Automation and ease of use	9d	08/06/2017	16/06/2017
Second progress report	4d	15/05/2017	17/05/2017
inal report draft	4d	05/06/2017	07/06/2017
Presentation draft	4d	19/06/2017	21/06/2017
Final report delivery	5d	22/06/2017	27/06/2017
Poster delivery	4d	27/06/2017	29/06/2017

Figure 7.3: Gantt diagram of the new schedule [February-March]



Nombre	Duració	Inicio	Fin
Learning basics about Bitcoin cryptocurrency	**6d**	**20/02/2017**	**23/02/2017**
Cryptographic primitives	1d	20/02/2017	20/02/2017
Bitcoin data structures (hash list, merkle tree, ...)	2d	21/02/2017	22/02/2017
Consensus decision-making: proof-of-work	1d	23/02/2017	23/02/2017
Low-level Bitcoin study	**11d**	**24/02/2017**	**07/03/2017**
Transactions and its validation	2d	24/02/2017	27/02/2017
Bitcoin scripting language	6d	28/02/2017	06/03/2017
Understanding sample raw P2PKH transactions	1d	06/03/2017	07/03/2017
Initial report	1.5d	27/02/2017	28/02/2017
Software / framework development	**19.5d**	**08/03/2017**	**24/03/2017**
Environment setup and libraries research	0.5d	08/03/2017	08/03/2017
Software design and architecture	2d	08/03/2017	10/03/2017
Transaction modelling	3d	10/03/2017	14/03/2017
Script modelling	3d	15/03/2017	17/03/2017
P2PKH Basic transactions generation	3d	17/03/2017	21/03/2017
Transaction debugging and testing in the network	3d	22/03/2017	24/03/2017
First progress report	4d	04/04/2017	06/04/2017
Unidirectional payment channels	**35d**	**24/03/2017**	**26/04/2017**
Undirectional payment channels case study and understandin	1d	24/03/2017	27/03/2017
Iteration 1: Commitment	5d	28/03/2017	31/03/2017
Iteration 2: Commitment alternative	5d	31/03/2017	05/04/2017
Iteration 3: Payment transactions	10d	06/04/2017	17/04/2017
Iteration 4: Close transactions	10d	17/04/2017	26/04/2017
Bidirectional payment channels	**59d**	**26/04/2017**	**16/06/2017**
State of the art study	2d	26/04/2017	27/04/2017
Iteration 1: Commitment	12d	28/04/2017	10/05/2017
Iteration 2: Payment	12d	10/05/2017	22/05/2017
Iteration 3: Close	10d	22/05/2017	31/05/2017
Iteration 4: Security enfrcement and revision	9d	31/05/2017	08/06/2017
Iteration 5: Automation and ease of use	9d	08/06/2017	16/06/2017
Second progress report	4d	15/05/2017	17/05/2017
inal report draft	4d	05/06/2017	07/06/2017
Presentation draft	4d	19/06/2017	21/06/2017
Final report delivery	5d	22/06/2017	27/06/2017
Poster delivery	4d	27/06/2017	29/06/2017

Figure 7.4: Gantt diagram of the new schedule [March - April]

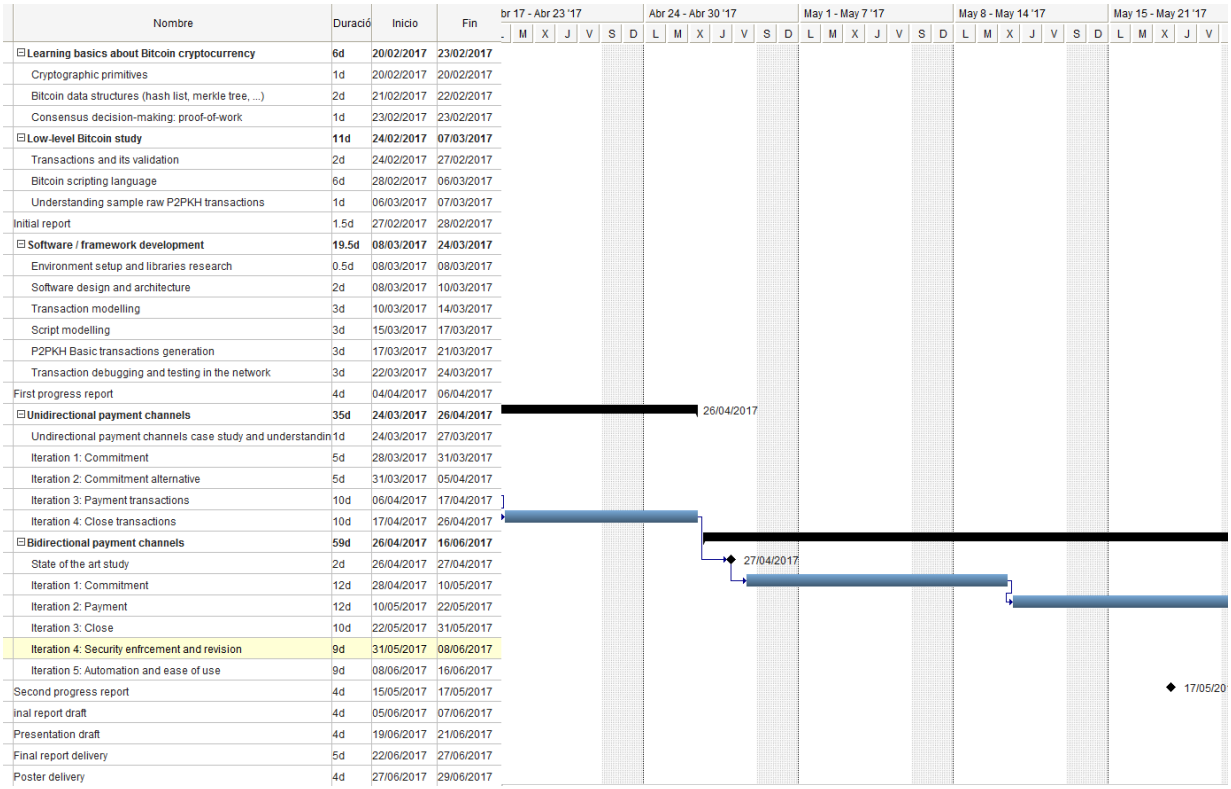


Figure 7.5: Gantt diagram of the new schedule [April - May]

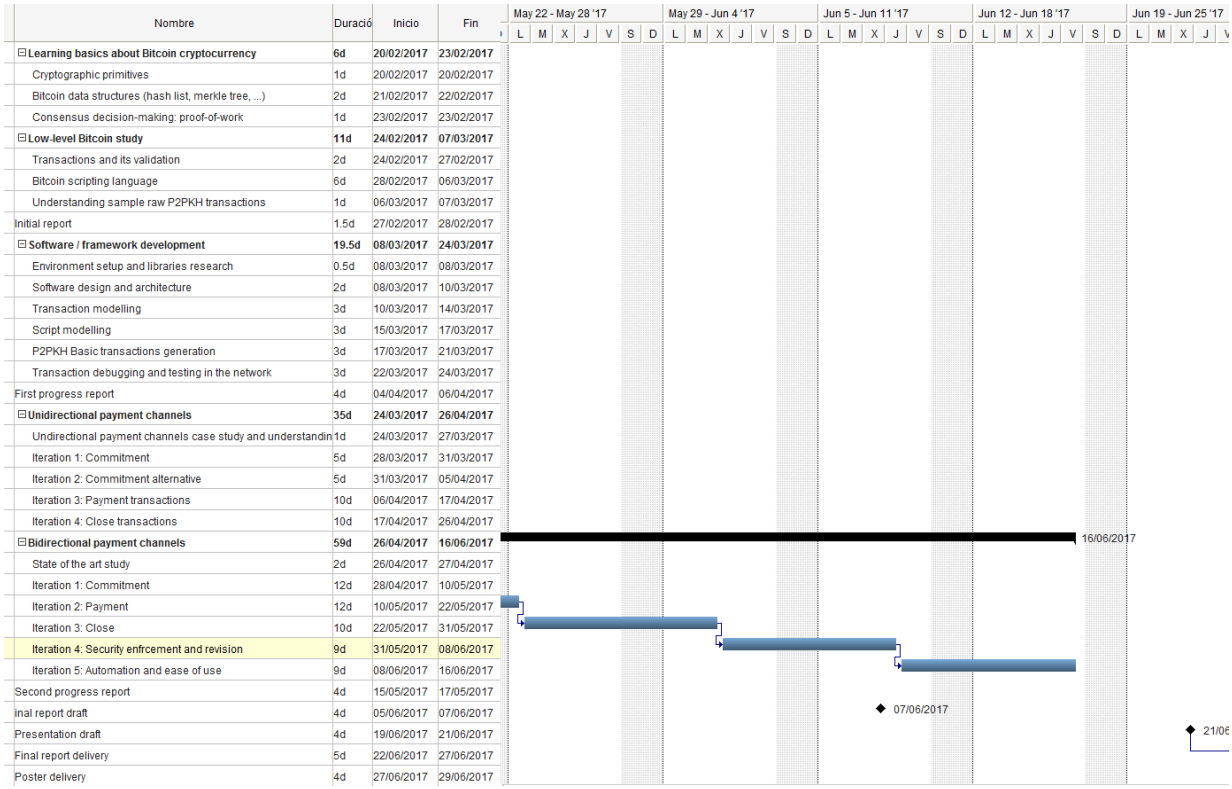


Figure 7.6: Gantt diagram of the new schedule [May - June]

Nombre	Duració	Inicio	Fin
Learning basics about Bitcoin cryptocurrency	**6d**	**20/02/2017**	**23/02/2017**
Cryptographic primitives	1d	20/02/2017	20/02/2017
Bitcoin data structures (hash list, merkle tree, ...)	2d	21/02/2017	22/02/2017
Consensus decision-making: proof-of-work	1d	23/02/2017	23/02/2017
Low-level Bitcoin study	**11d**	**24/02/2017**	**07/03/2017**
Transactions and its validation	2d	24/02/2017	27/02/2017
Bitcoin scripting language	6d	28/02/2017	06/03/2017
Understanding sample raw P2PKH transactions	1d	06/03/2017	07/03/2017
Initial report	1.5d	27/02/2017	28/02/2017
Software / framework development	**19.5d**	**08/03/2017**	**24/03/2017**
Environment setup and libraries research	0.5d	08/03/2017	08/03/2017
Software design and architecture	2d	08/03/2017	10/03/2017
Transaction modelling	3d	10/03/2017	14/03/2017
Script modelling	3d	15/03/2017	17/03/2017
P2PKH Basic transactions generation	3d	17/03/2017	21/03/2017
Transaction debugging and testing in the network	3d	22/03/2017	24/03/2017
First progress report	4d	04/04/2017	06/04/2017
Unidirectional payment channels	**35d**	**24/03/2017**	**26/04/2017**
Undirectional payment channels case study and understandin	1d	24/03/2017	27/03/2017
Iteration 1: Commitment	5d	28/03/2017	31/03/2017
Iteration 2: Commitment alternative	5d	31/03/2017	05/04/2017
Iteration 3: Payment transactions	10d	06/04/2017	17/04/2017
Iteration 4: Close transactions	10d	17/04/2017	26/04/2017
Bidirectional payment channels	**59d**	**26/04/2017**	**16/06/2017**
State of the art study	2d	26/04/2017	27/04/2017
Iteration 1: Commitment	12d	28/04/2017	10/05/2017
Iteration 2: Payment	12d	10/05/2017	22/05/2017
Iteration 3: Close	10d	22/05/2017	31/05/2017
Iteration 4: Security enfrcement and revision	9d	31/05/2017	08/06/2017
Iteration 5: Automation and ease of use	9d	08/06/2017	16/06/2017
Second progress report	4d	15/05/2017	17/05/2017
inal report draft	4d	05/06/2017	07/06/2017
Presentation draft	4d	19/06/2017	21/06/2017
Final report delivery	5d	22/06/2017	27/06/2017
Poster delivery	4d	27/06/2017	29/06/2017

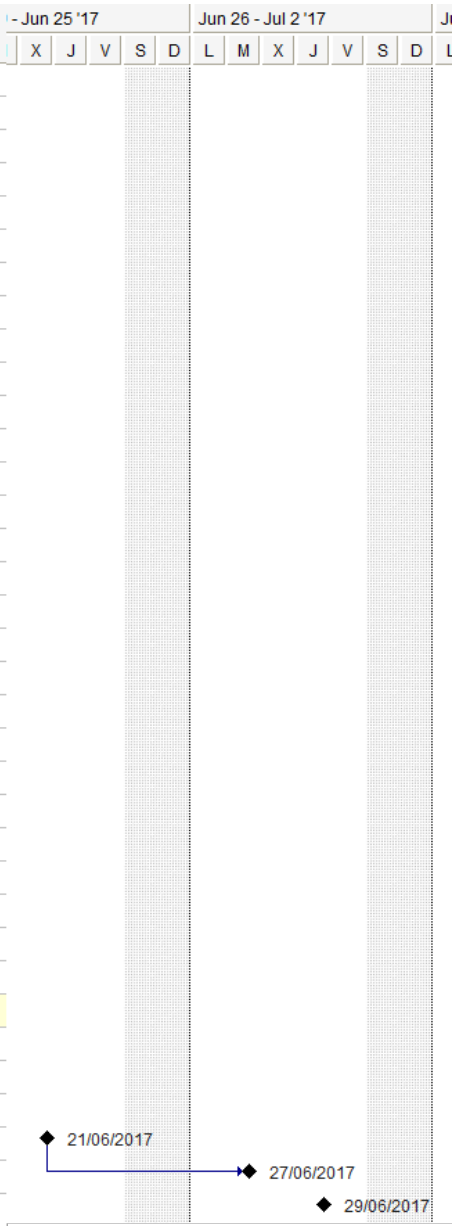


Figure 7.7: Gantt diagram of the new schedule [June - end]

Bibliography

[1] "File:txbinarymap.png - bitcoin wiki." https://en.bitcoin.it/wiki/File:TxBinaryMap.png. (Accessed on 04/04/2017).

[2] "Testnet btc charts | blockr.io." https://tbtc.blockr.io/charts. (Accessed on 04/04/2017).

[3] "Bitcoin and cryptocurrency technologies - princeton university | coursera." https://www.coursera.org/learn/cryptocurrency/. (Accessed on 04/04/2017).

[4] "Bitcoin wiki." https://en.bitcoin.it/wiki/Main_Page. (Accessed on 04/04/2017).

[5] "Bitcoin blockbrowser." https://webbtc.com. (Accessed on 04/04/2017).

[6] "Blockchain explorer price charts | blockr.io." https://btc.blockr.io/. (Accessed on 04/04/2017).

[7] "Transaction - bitcoin wiki." https://en.bitcoin.it/wiki/Transaction. (Accessed on 04/04/2017).

[8] "Script - bitcoin wiki." https://en.bitcoin.it/wiki/Script. (Accessed on 04/04/2017).

[9] "Deconstructing bitcoin transactions part 1 - siliconian." https://www.siliconian.com/blog/16-bitcoin-blockchain/22-deconstructing-bitcoin-transactions. (Accessed on 04/04/2017).

[10] "transactions - how to redeem a basic tx? - bitcoin stack exchange." http://bitcoin.stackexchange.com/questions/3374/how-to-redeem-a-basic-tx. (Accessed on 04/04/2017).

[11] "Op_checksig - bitcoin wiki." https://en.bitcoin.it/wiki/OP_CHECKSIG. (Accessed on 04/05/2017).

[12] "bitcoin core - why the signature is always 65 (1+32+32) bytes long? - bitcoin stack exchange." http://bitcoin.stackexchange.com/questions/12554/why-the-signature-is-always-65-13232-bytes-long. (Accessed on 04/05/2017).

[13] "Bitcoin blockbrowser - debug script execution." https://webbtc.com/script/ce97c4ebc6a058dbb775cf8d7d95aa06a8728673c482a6a721911185b39c332a:0. (Accessed on 04/05/2017).

[14] "mazaclub/hashmal: Hashmal." https://github.com/mazaclub/hashmal. (Accessed on 04/05/2017).

[15] "bitcoin/bitcoin: Bitcoin core integration/staging tree." https://github.com/bitcoin/bitcoin. (Accessed on 04/05/2017).

[16] "Ethereum project." https://www.ethereum.org/. (Accessed on 04/05/2017).

[17] "petertodd/python-bitcoinlib: Python2/3 library providing an easy interface to the bitcoin data structures and protocol.." https://github.com/petertodd/python-bitcoinlib. (Accessed on 04/06/2017).

[18] "vbuterin/pybitcointools: Simple, common-sense bitcoin-themed python ecc library." https://github.com/vbuterin/pybitcointools. (Accessed on 04/06/2017).

[19] "Understanding the lightning network, part 1: Building a bidirectional bitcoin payment channel | bitcoin magazine." https://bitcoinmagazine.com/articles/understanding-the-lightning-network-part-building-a-bidirectional-payment-channel (Accessed on 04/06/2017).

[20] "Bitcoin multisig the hard way: Understanding raw p2sh multisig transactions." http://www.soroushjp.com/2014/12/20/bitcoin-multisig-the-hard-way-understanding-raw-multisignature-bitcoin-transactio (Accessed on 04/06/2017).

[21] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*, pp. 3–18, Springer, 2015.