Açai: a backup protocol for Lightning Network wallets

Master thesis

Margherita Favaretto

Kongens Lyngby 2019

DTU Compute
Department of Applied Mathematics and Computer Science

Master of Science in Engineering
Computer Science and Engineering

Açai: a backup protocol for Lightning Network wallets
Master thesis

Margherita Favaretto
s170065@student.dtu.dk

Advisors:
Nicola Dragoni, Associate Professor at DTU Computer Science Department
Stefano Pepe, CEO Uniquid Inc.

Technical University of Denmark
Department of Applied Mathematics and Computer Science

DTU Compute
Richard Petersens Plads
Building 324
2800 Kgs. Lyngby
DENMARK
Tel. +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Nowadays, Bitcoin is considered the world's most widely used and valuable digital currency. The scope of this technology is allowing anyone to send value without a trusted intermediary or depository, with the purpose to give to each user the complete control of their own money and a higher level of privacy on own use.
As all new technologies, Bitcoin is not perfect immediately, and a long work has to be done to make this technology more solid and a possible candidate to become the main payment system in the next future.
Bitcoin, also known as *Digital Gold*, is a network based on distributed consensus, which means that the nodes on the network has to verify the transactions, and agree with their order and existence on the *Blockchain*. In order to achieve the consensus, there are scalability issues that limit at 7 transactions per second [Cro+16] the current bandwidth. This limit of Bitcoin is called *Scalability problem* and refers to the fact that this technology presents some limits on the number of transactions that can process. For this reason, different technologies have been proposed to solve this issue, including *Lightning Network*.

The focus of this thesis is on this last technology, which is the second layer payment of Bitcoin and permits fast payments transactions inside a network of participants without the use of the Blockchain for each transaction. The main problem of Lightning Network, from the user point of view, is the absence of a recovery mechanism for the unspent bitcoins in case of a wallet failure (e.g. loss of the device, or corrupted transaction data inside the wallet local storage). In fact, the absence of an underlying distributed ledger in Lightning makes impossible the recovery of unspent transactions through the address derivation, which is normally provided by the Bitcoin legacy protocol. Therefore, this issue does not permit to have the portability among different devices and even the possibility to be able to recover their funds if the own device is damaged.

The objective of this research project is to analyze Bitcoin and Lightning technology, their mechanism and their main features, ending up with a technically feasible solution to mitigate the recovery of unspent Bitcoin after a Lightning wallet failure. To this aim, we have analyzed the literature, the most recent upgrade, we have met the most active developers of the Lightning Network open source protocol, validating our possible solution also from an engineering and practical point of view. All the gathered information has been finally used to design a possible solution, called *Açai Protocol*. Thanks to our interaction with the Lightning Network community, we were able to identify the Watchtower functionality and design the Açai Protocol as an extension for it. In more details, the proposed solution is aimed to adopt the mechanism of Watchtowers, already introduced in the Lightning Network to monitor the

channels when a node is offline, as a possible mechanism of backup.

In conclusion, we will provide an implementation of the protocol to demonstrate how it is possible to extend the current code implementing the Watchtower, including the backup functionality introduced in our solution.

Preface

This thesis was prepared at DTU Compute, Department of Applied Mathematics and Computer Science at the Technical University of Denmark, and at Uniquid Inc., placed in San Francisco (CA, United States). This work was done in fulfilment of the requirements for acquiring an M.Sc.in Computer Science and Engineering.

The project has been completed in the period between 28^{th} August 2018 and 27^{th} January 2019 with an assigned workload of 30 ECTS credits.

The thesis was conducted under the supervision of Nicola Dragoni, Associate Professor at DTU Compute, and Stefano Pepe, CEO of Uniquid Inc.

Kgs. Lyngby, 27^{th} January 2019

Margherita Favaretto

Acknowledgements

I would like to express my deep gratitude to Prof. Nicola Dragoni, for helping and guiding me during the development of this work, and allowing me to reach important goals that I would have never expected to achieve.

I am also very thankful to Stefano Pepe, for the opportunity to implement a part of my thesis at his company. Thanks for being so patient to teach me, listening to my doubts and guiding me to this so innovative world of Bitcoin. Thanks for allowing me to discover the inspiring life in San Francisco, for all the professional dialogues and tips: this experience will be a part of me, as well as a source of inspiration for my future.

I would like to thank all the members of the Lightning Network Community, especially Alekos Filini, Peter Todd, Christian Decker, Alex Bosworth and Conner Fromknecht for the important feedbacks and invaluable advice, which were fundamentals for this work.

With this thesis, I will conclude two very important years of my life, so I wish to thank all my friends, my roommates and my colleagues that have helped me to feel at home, even in a country that is not mine. I will take with me all of your life-lessons and thanks for having been my family in these wonderful years.
I wish to thank Riccardo, that despite the distance, has been on my side during these two years. Thanks for being there for me day by day, giving me all the strength to work with perseverance to reach all my goals.
Finally, I am extremely grateful to my Family, that has always supported me to pursue my academics studies. Thank you for allowing me to live this experience, celebrating with me my happiest moments and teaching me to smile in front of every difficulty. This accomplishment would have not been possible without you. Thank you.

Contents

CHAPTER 1
Introduction

Bitcoin can be defined as a collection of concepts and technologies that are the basis of a digital money ecosystem. The term Bitcoin, that is both the name of the technology and the name of the units of currency, is used to store and transmit value among the participants in the Bitcoin network [Ant14].

Research produced by the University of Cambridge estimates that in 2017, there were 2.9 to 5.8 million unique users using a cryptocurrency wallet, most of them using Bitcoin [HR17]. Furthermore, in [SA18] it is possible to observe that the number of Bitcoin transactions in December 2016 were 180 million, the value is steadily increasing, and in December 2018 was 368 million, Figure 1.1.

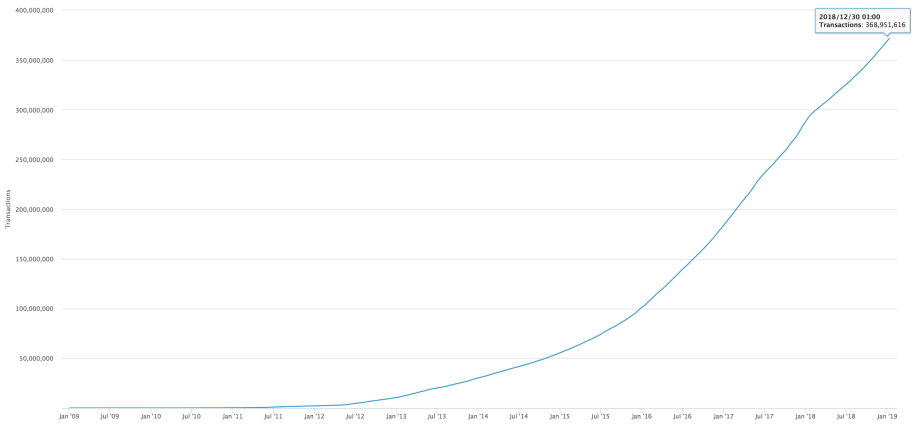

Figure 1.1: Total Number of transactions.[SA18].

Compared to the current payment system based on centralization, the Bitcoin permits to have a *finality condition*, since the probability that a transaction will be reverted is extremely lower over time [CK17]. Furthermore, it permits to have a *privacy environment*, due to the use of pseudonymous, neither transaction or accounts are connected to real-world identities. In fact, the Bitcoins are received on the so-called address, which is not possible to connect to a real-world identity, unless the intermediaries reveal their user-base information like emails and phone numbers under their custody. Unlike the normal payment system, which requires several days to complete a transaction if the payee and the payer are completely in a different physical location, Bitcoin is *fast* and *global* since a transaction is propagated almost immediately in the network and they are completely independent of the physical location. Moreover, it is *permission-*

less, in fact, everybody is able to download the software of Bitcoin for free [Inc16].

Every transaction inside the Bitcoin is stored on a single, distributed public ledger, called *Blockchain*. All confirmed transactions are included in a so-called 'block', that it is broadcast to the Blockchain, so all users are aware of each transaction. This prevents stealing and double-spending, that happens when someone spends the same currency twice [Eco15].

Blockchain has evolved in the last 10 years and has been steadily moving from mathematical concepts to implementations with different network technologies and architectures. The use of this technology permits to achieve the main goals of Bitcoin: transparency, decentralization, finality and anonymity [Put+].

In fact, as a public ledger system, Blockchain records and validates each and every transaction made, which makes the system *transparent*. Despite Bitcoin users make their wallet addresses publicly, it is extremely hard to trace transactions back to their real-life identity. Moreover, even if the wallet addresses was publicized, a new wallet address can be easily generated. This allows to achieve *anonimity* inside the system. All the transactions made are authorized by miners, a network of communicating nodes, which makes the transactions *final* and prevent it from the threat of hacking. Blockchain technology discards the need of any third-party or central authority for peer-to-peer transactions, and this permits to have a *decentralization* of the technology.

On the other hand, Bitcoin presents some limits on the number of transactions that the network is able to process. This is related to the fact that the blocks are limited in size and frequency, and the Bitcoin network is able to handle maximally around 7 transactions per second. Compared to a centralized payment system like Visa, which can handle up to 45,000 transactions per second, this is an obstacle if we want to extend the use of Bitcoin on a large scale [Con18].

This question called also as the *Bitcoin Scalability problem* refers to the discussion concerning the limits on the number of transactions the Bitcoin network can process. This has created a bottleneck in Bitcoin, causing transaction fees and delayed processing of transactions that cannot be fit into a block [Pea16]. Various proposals have emerged on how to scale Bitcoin, and a contentious debate has resulted, that is characterized as an *ideological battle over Bitcoin's future* [WP17].

One of the proposed solutions to the Bitcoin scalability problem is *Lightning Network technology*, introduced in the paper by Poon and Dryja [PD16]. This technology permits the users to have transactions without publishing all of them on the Blockchain, but just the initial and final status. This does not increase the capacity of the Bitcoin system and allows that more transactions can be done.

The Lightning Network consists of interconnected *payment channels*. A payment channel is a one-to-one channel allowing two participants to exchange funds between each other. A network is created by connecting multiple channels

with different users, making transactions possible across multiple channels. For instance, Alice may pay Bob without having a channel directly connecting them, but using intermediary channels [Net18a].

The develop of Lightning Network technology is still in progress, and like all emerging technologies, there are different challenges to face to ensure that the technology is successfully adopted on a large scale. Nevertheless, this technology is already very utilized and on 1st January 2019, the number of Lightning Network Channels were 16,872 [Vis19].

One of the problems of Lightning Network is not offering a mechanism of backup if one of the nodes loses accidentally all transaction data, stored inside the confidential status channels. As a result, if a wrong channel status is sent to another peer, the node could be considered malicious and be punished with a Lightning Network penalty, which imposes the loss of own funds on the channel.

The purpose of this work is analyzing this problem and finding a solution that can permit to have a recovery mechanism avoiding the usage of wrong channel states. This research is fundamental for all who want to make Bitcoin as the main payment system in the next future. The solution, defined as Açai Protocol, has to manage a backup mechanism to guarantee to nodes to recover own status if they accidentally lose all information.

1.1 Thesis Objectives

Given all the information and the context above, we believe that Bitcoin and Lightning Network technology are very interesting fields of study even if in their infancy. The main aim of this thesis is to study the Bitcoin, Lightning Network technology and to propose a solution for a Lightning Network recovery mechanism.

In summary, the thesis presents the following objectives:

- Analysing the Bitcoin technology, Blockchain and the Scalability problem.
- Studying Lightning Network and its main properties.
- Performing a literature survey of the state of the problem that we want to focus on, to get a better overview of what has been done and what still needs to be done.
- Studying how the fields of Game Theory and Cryptography are connected to Bitcoin and how these fields are addressed in this thesis.
- Proposing a possible solution to the recovery mechanism in Lightning Network.
- Implementing the Açai Protocol, to show how the code of the Watchtower can be extended to incorporate the back-up functionality.
- Proposing future works that can be improved further Lightning Network, and also the Bitcoin technology.

1.2 Thesis Outline

The structure of this thesis is as follows:

- **Chapter 1 - Introduction.** It gives an overview of the technologies analyzed in this thesis, the problem we focus on, the thesis objectives and the outline.
- **Chapter 2 - Theoretical Background.** This chapter provides some basics concepts of Bitcoin and Lightning Network. Moreover, it analyzes the Scalability problem in more details.
- **Chapter 3 - Bitcoin technology.** This chapter will analyze Bitcoin, with an analysis of the white paper "Bitcoin: A Peer-to-Peer Electronic Cash System" written by the unknown person or people who developed Bitcoin, called Satoshi Nakamoto, in 2008. [Nak].
- **Chapter 4 - Lightning Network.** This chapter analyzes the Lightning Network mechanism, seeing all the features important for a full comprehension of the next chapters.
- **Chapter 5 - Problem Description and Related Work.** The aim of this chapter is to explain the problem faced in the thesis in more details. This chapter also describes what has been done before in the literature and what is needed.
- **Chapter 6 - Methodology.** This chapter describes how we have reached the goals, which features we have considered and which theoretical aspects we have used.
- **Chapter 7 - Açai Protocol.** This chapter proposes a possible solution to have a recovery mechanism in Lightning Network, using the concept of Watchtowers. All the steps to achieve this solution are demonstrated and explained.
- **Chapter 8 - Implementation.** This chapter aims to provide an implementation of the solution proposed in Chapter 7.
- **Chapter 9 - Evaluation.** This chapter documents which goals we have achieved and if the proposed solution can completely solve the problem.
- **Chapter 10 - Conclusion.** This chapter concludes what we have reached and also discusses hindsight: which aspects were even better or a little worse than we have expected and how this project could be improved by further work.

CHAPTER 2

Theoretical Background

Before facing the main chapters of this project, some theoretical fundamentals are here presented. The aim of this chapter is to give an overview of the main basics concepts connected to Bitcoin and Lightning Network technology.

The Chapter is organized as follows. First, a general overview of the Bitcoin technology is giving, analyzing also its ledger, the Blockchain. Secondly, an analysis of the Scalability problem is presented, on what it consists and why it is so crucial in the progress of technology as future currency. Subsequently, the Lightning Network technology is introduced, analyzing its main aspects.

In the next two chapters, Bitcoin and Lightning Network will be explored in more details, to understand the mechanisms which permit these two technologies to work.

2.1 Bitcoin

With the term Bitcoin, we refer to both a virtual currency, indicated by the symbol (฿), and also to the behind technology. The goal of Bitcoin is to send money in a peer-to-peer network without the need for intermediaries [Net16]. It may be used to buy or sell goods and services as with the other common cryptocurrencies.

Bitcoin users communicate with each other using the Bitcoin protocol mainly via the internet network, even if other transport networks can also be used. The Bitcoin protocol stack is an open source software, that may be run on several computing devices, including laptops and smartphones [Ant14]. In the Figure 2.1, it is shown Mycellium, an Android application that permits to have a Bitcoin wallet, and to send and receive bitcoins using own mobile phone [Dev18].

Through Bitcoin is possible to do all common things that it is possible to do with conventional currencies, for example: buy and sell goods, send money to people or organizations, or extend credit. Bitcoin can be also purchased, sold, and exchanged for other currencies at specialized currency exchanges. For instance, on 22^{nd} January 2019, 1 Bitcoin was worth 3,526.95 USD (United States dollar) [Des18].

As it was already mentioned, Bitcoin is a distributed, peer-to-peer system, so there is no central server or point of control. Bitcoin represents the culmi-

Figure 2.1: Mycelium Bitcoin Wallet application [Dev18].

nation of decades of research in cryptography, game theory and distributed systems. It is the result of a unique and powerful combination of four key innovations brought together, that permits to this technology to work:

1. The Bitcoin Protocol: a decentralized peer-to-peer network
2. The Blockchain: a public ledger containing all the transactions
3. Consensus rules: a set of computational rules for independent transaction validation and currency issuance
4. Proof-of-Work algorithm: a mechanism for reaching global decentralized consensus on the valid Blockchain [Ant14]

The Bitcoin system consists of users with wallets containing keys, transactions that are propagated across the network, and miners who produce the consensus Blockchain, which is the authoritative ledger of all transactions.
As it was mentioned in Chapter 1, Bitcoin was invented in 2008 with the publication of a paper titled "Bitcoin: a Peer-to-Peer Electronic Cash System" [Nak], written under the alias of Satoshi Nakamoto. In this paper, Nakamoto combined different prior inventions such as b-money and hashCashTo, giving life a decentralized electronic cash system that does not rely on a central authority for currency issuance or settlement and validation of transactions.

The system is based on a distributed computation system, called a "Proof-of-Work" algorithm, that aims to conduct a "global" election every 10 minutes, permitting the decentralized network to arrive at a consensus about the state of transactions [Ant14].

Bitcoin is not controlled by any person or organizations, neither Satoshi

Nakamoto nor anyone else experts individual control over the system, that operates based on fully transparent mathematical principles, open source code, and consensus among participants. The invention itself is also revolutionary and has already become a new science in the fields of distributed computing, economics, and econometrics [Ant14].

Bitcoin is also a practical and novel solution to a problem in distributed computing, called *Byzantine General's Problem*. With this term, we define the problem consists of trying to have an agreement on a course of action or state of a system by exchanging information over an unreliable and potentially compromised network. The Proof-of-Work algorithm represents a breakthrough in distributed computing, in fact, this solution can be used to achieve consensus on decentralized networks to prove the fairness of elections, lotteries, asset registries, digital notarization, and more [Ant14].

The Bitcoin network started to be active in 2009, based on an implementation published by Nakamoto and since revised by many other programmers.

In January 2019, the 24-hour average block size was 0.79 MB. The average of confirmed Bitcoin transactions per day was 250,000. The average size of transactions waiting to be confirmed was 267,707 Bytes [SA18].

2.1.1 Blockchain

Blockchain was invented by Satoshi Nakamoto in 2008 [Nak] to serve as a digitized, decentralized, public ledger of Bitcoin. All of the Bitcoin users have a copy of the Blockchain, that may be used to have a unique chronological history about the transactions happened in the network [Nak].

The Blockchain may be seen as a chain of blocks: each block contains a hash, generated using the SHA256 cryptographic hash algorithm of the previous block, (*parent block*), up to the first block of the chain, called *genesis block*.
The *previous block hash* field is inside the block header and affects the current block's hash. The child's own identity changes if the parent's identity changes. When the parent block hash is modified, the parent's hash also changes. As consequence, the parent's changed hash causes a change in the previous block hash pointer of the child. Subsequently, this causes the child's hash to change, which necessitates a change in the pointer of the grandchild, which in turn changes the grandchild, and so on...
Therefore, if a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks. Since the recalculation would require enormous computation power, the existence of a long chain of blocks makes the Blockchain's deep history immutable, which is a characteristic of Bitcoin's security [Ant14].

In [Ant14], the authors compared the Blockchain to layers in a geological formation or glacier core sample. The surface layers could change with the seasons, or even be overthrown before they have time to settle. But as time

goes on, geological layers become more and more stable and if you look a few hundred feet down, you are looking to the past that has remained undisturbed for millions of years. In the Blockchain, the most recent few blocks could be changed if there is a chain recalculation due to a fork. The top six blocks are possible to recalculate. But if you go more deeply into the Blockchain, blocks are less and less likely to change.

The Blockchain is a decentralized system. Having a decentralized system permits to have no third party that can access to information, the transactions are fast and transparent, everyone is able to see the transactions that have taken place on the network. Having a decentralized system brings to three advantages:

- Fault tolerance: decentralized systems are less likely to fail accidentally since they rely on many separate components.
- Attack resistance: decentralized systems are more expensive to attack and destroy or manipulate because they lack sensitive central points that can be attacked at a much lower cost than the economic size of the surrounding system.
- Collusion resistance: it is much harder for participants in decentralized systems to act in ways that benefit them at the expense of other participants [But17].

Figure 2.2 compares the centralized system with the decentralized Blockchain system.

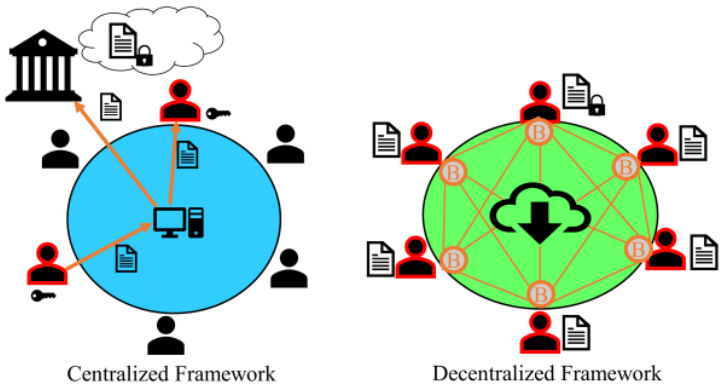

Figure 2.2: Comparison between the (a) centralized system with intermediaries and a (b) decentralized Blockchain system [Put+].

The size of the Blockchain system is steadily increasing. In August 2014, the dimension of the Blockchain was of 20 Gigabyte, in January 2019 the dimension was of 198 Gigabyte, as it is demonstrated in Figure 2.3.

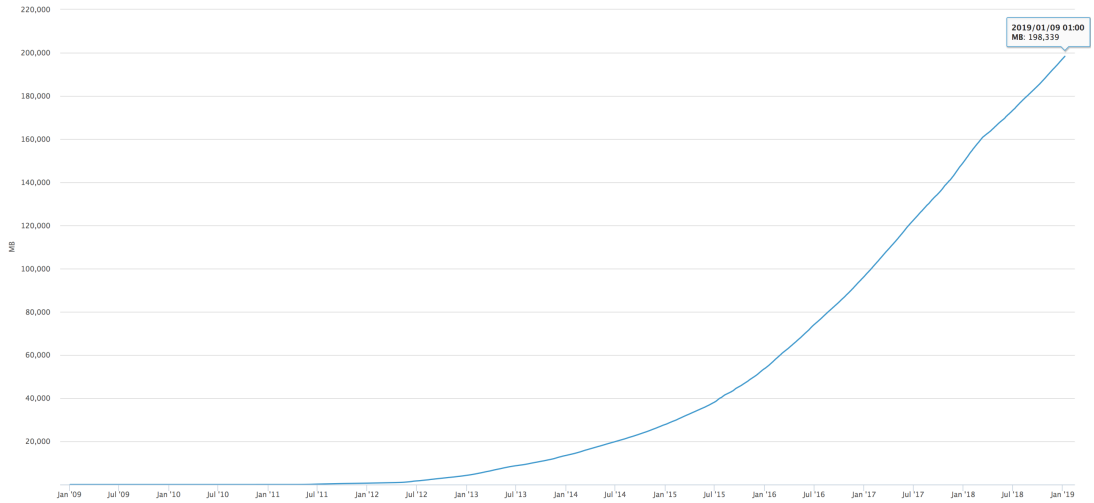

Figure 2.3: The total size of all block headers and transactions. Not including database indexes[Blo18b].

The Figure 2.4 gives a good overview, summarizing all aspects of Bitcoin system that we see in Sections 2.1 and 2.1.1.

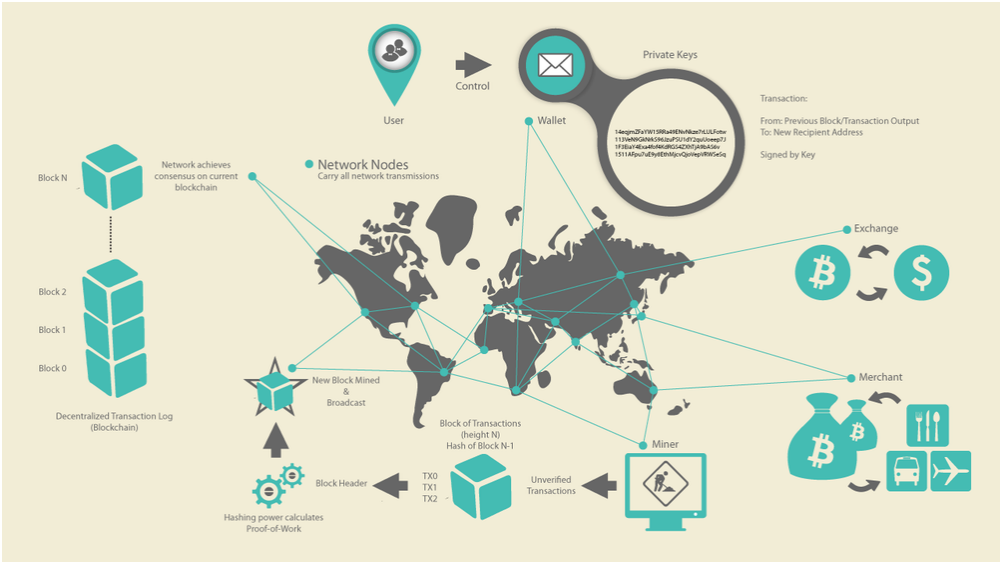

Figure 2.4: Bitcoin overview [Ant14].

2.2 The Bitcoin workflow

The Figure 2.5 summarizes the entire workflow of the Bitcoin, that we mentioned above.

How it is possible to observe, when a new transaction takes place, it is broadcast in the Bitcoin network. There is no gold or government issued money backing this money, it's a completely decentralized form of money. Each node inside the Bitcoin network has an own copy of the transactions in the Blockchain, and they can validate the new transaction based on the data inside their own copy. To validate the transaction a crucial point is the use of cryptography, that permits to have integrity and maintain chronological order. Once the transaction is confirmed, it can be combined with the other new transactions to create a block to add in the Blockchain. If the transaction is included in the ledger, it can be considered confirmed. Including all transactions in the Blockchain allows Bitcoin wallets to calculate their spendable balance in a way that new transactions can be verified and ensuring that they're actually owned by the spender.

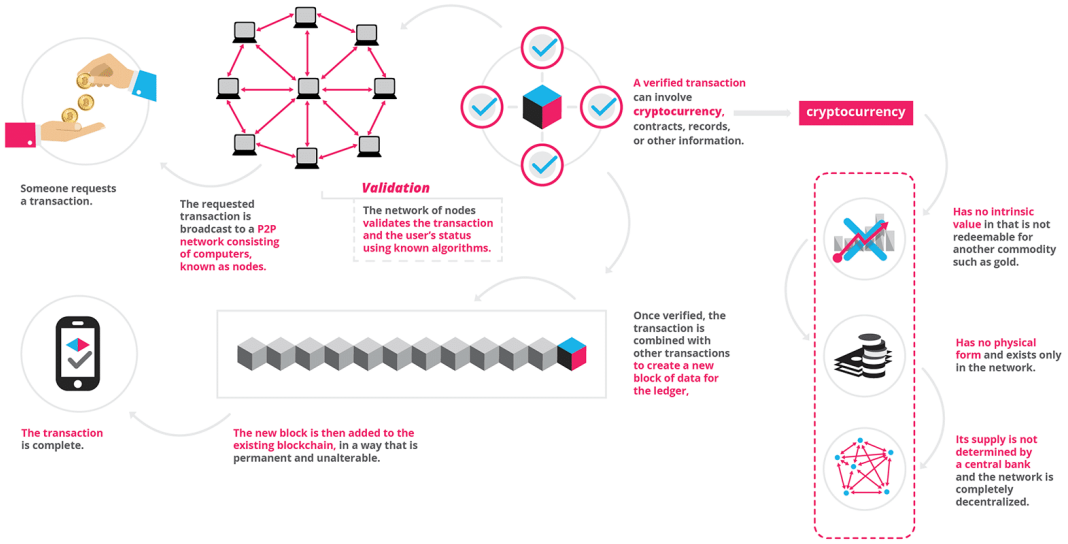

Figure 2.5: Bitcoin workflow [how18].

2.3 Scalability problem

Bitcoin is the world's most widely used and valuable digital currency, but despite this, there are some drawbacks to its decentralized design. In fact, transactions confirmed on the Bitcoin Blockchain take up to one hour before they are final. In fact, as Satoshi Nakamoto shows in [Nak], if an attacker has 0.1% of the hashing power in the network, the minimum number of valid blocks, in

order to have 99,9% probability that he is not able to reverse the transaction, is 6.

The Bitcoin scalability problem refers to the limits on the number of transactions the Bitcoin network can process. Scaling means to facilitate transactions within a given amount of storage space and network capacity, and how to perform more transactions at rates that are economically feasible. The on chain transaction processing capacity of the Bitcoin network is bounded by the average block creation time of 10 minutes and the block size limit of 1 MegaByte. These also restrict the network's throughput. In fact, the transaction processing capacity maximum is estimated between 3.3 and 7 transactions per second [Cro+16]. This block size limit and the exceptional adoption of its infrastructure has created a bottleneck in Bitcoin, causing increased transaction fees and delayed processing of transactions that cannot be fit into a block [Net18b]. As we have mentioned in Section 1, compared to a centralized payment system like Visa, which can handle up to 45,000 transactions per second, this is an obstacle If we want to extend the use of Bitcoin on a large scale [Con18], Figure 2.6.

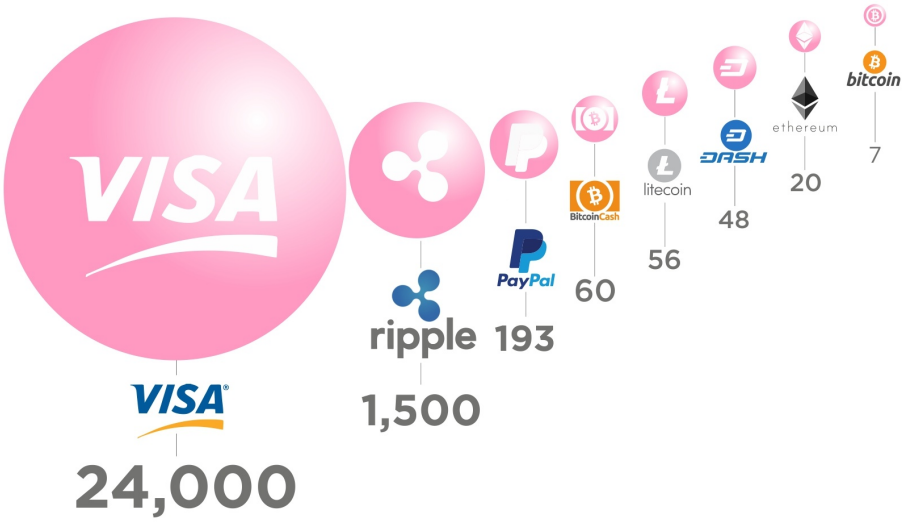

Figure 2.6: Cryptocurrencies Transaction Speeds compared to visa and Paypal[how18].

The problems connected to the Scalability Problem are:

- Limitations: the Blockchain includes each created block to its ledger, but at the same time also each block increases with data as it carries the history of the last blocks.
- Data Size: The Bitcoin Blockchain is limited in size to 1MB, this restricts the maximum throughput to 3.3–7 transactions/sec. [Cro+16]

- Response Time: Each confirmed transaction requires peer-to-peer verification, which could become time-consuming with the increase of the number of blocks involved. Bitcoin is currently verifying, or creating, one block every ten minutes.[Bit18]

2.4 Lightning Network

One of the solutions proposed to solve the scalability problem is Lightning Network. This technology is a decentralized network using smart contract functionality in the Blockchain, to create instant payments across a network of participants.
We can define a smart contract as a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts permit the performance of credible transactions without third parties.

Lightning Network is one of the first application of a multi-party smart contract using Bitcoin's built-in scripting, and is providing a technological development in multiparty financial computations with Bitcoin [Net18a].

It is important to analyze this technology, since it is widespread and more and more people are beginning to use it, as it is shown in Figure 2.7, indicating the number of Lightning Network channels.

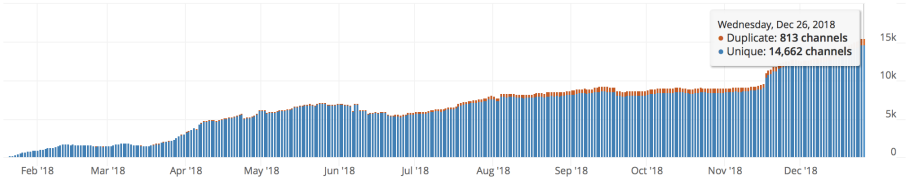

Figure 2.7: Number of channels in Lightning Network between February 2018 and December 2018. [Vis19].

This technology permits to have high volume, low latency digital micropayments without relying on trusted intermediaries [Lar18]. In this way Lightning participants do not give unilateral custody of funds to a third trusted party, reducing transaction costs and counterparty risk. Hence, it achieves instant micropayments via smart contracts. In Lightning Network, through a network of multisignature transactions, any participant is able to pay anyone else within the graph of participants [Net18a].

Payment channel, a local two-party consensus, is the core of the Lightning Network technology. The use of the Lightning Network is based on opening a payment channel by committing a funding transaction to the relevant Blockchain. Once the transaction is broadcast on the chain, the payment channel is open and ready for transactions. When one of the parties wishes to update the balance with a new balance, both parties must consent to the new balance and

generate a new spend from the transaction [PD16]. When one of the users or both of them decide to close the payment channel, they broadcast the final version of the transaction to the ledger [Net18a].

Therefore, the payment channels permit the participants to transfer Bitcoin to each other without having to make all their transactions public on the Blockchain. This is done by using a system to penalize uncooperative participants. In Lightning Network, the penalty works in this way: if one party broadcasts an old transaction state, the counterparty takes all the funds of the channel.

Therefore, Lightning Network can offer a scalable, decentralized micropayments solution on top of the Bitcoin Blockchain, through a network of interconnected payment channels [Net18a].

CHAPTER 3
Bitcoin Technology

In this Chapter, we will analyze in more details Bitcoin technology, explaining the important concepts that we need to know to understand completely the work that will be presented in the next Chapters.

As we have seen in the Chapter 2, Bitcoin is a solution against the double spending problem, it is a decentralized digital currency without a central part that may be sent from user-to-user on the peer-to-peer Bitcoin network without the need for intermediaries [Net16].

Furthermore, the transactions are verified by network nodes through cryptography and recorded in a public distributed ledger, the Blockchain. Bitcoins are created as a reward for a process known as *mining*.

In this Chapter, we will analyze the original white paper published by Satoshi Nakamoto in October 2008 [Nak], extending the work also with the most recent literature.

3.1 Transactions

The electronic payments in Bitcoin are created by generating transactions that transfer bitcoins among users.

As it is shown in Figure 3.1, a transaction is characterized by the following key elements: bitcoin version, a hash of the transaction, a Locktime, which indicates when the transaction may be spent on the blockchain, then one or more inputs, and one or more outputs.

Every input consists of the following information:

- hash pointer to a previous transaction, that is necessary to identify the output that is used as input
- an index to specific unspent previous transaction output (UTXO) that is spent in the current transaction
- unlocking script length
- unlocking script, called scriptSig, which satisfies the conditions associated with the use of UTXO

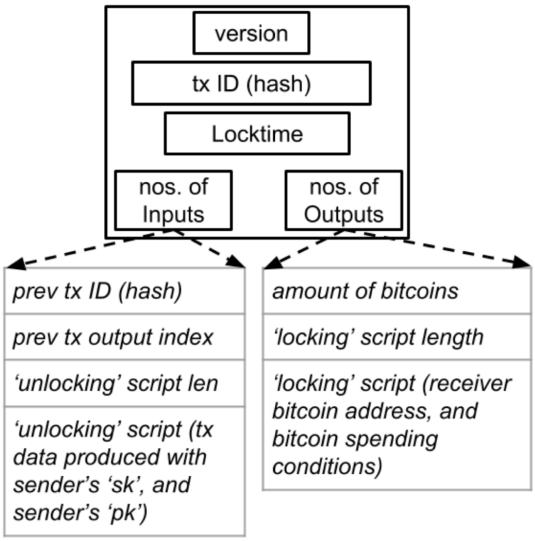

Figure 3.1: Bitcoin transaction [Con+18].

The output consists of the following elements:

- the number of bitcoins that are being transferred
- the locking script length
- the locking script, also known as scriptPubKey, which set a condition that must be met before the UTXO may be spent

In this network, an electronic coin is seen as a chain of digital signatures. So, each transaction processes in this way: the owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. This mechanism is called *Chain of Ownership*, in Figure 3.2.

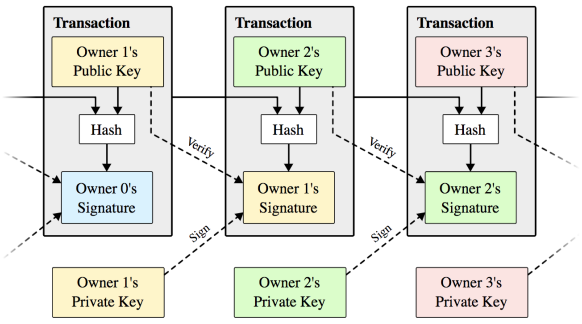

Figure 3.2: Transactions Ownership in Bitcoin [Nak].

This procedure presents a possible issue: the payee is not able to verify that one of the owners did not double spend the coin. For example, a user (e.g. Alice) might simultaneously generate two different transactions, using the same set of coins, to two receivers (e.g. Bob and Charlie). To this aim, the earliest transaction is considered as the one that counts, and the later attempts to double-spend are not considered. Moreover, to accomplish this without the use of intermediaries, transactions have to be publicly announced and the participants must agree on a single history of the transactions order. This is achieved through the use of Blockchain.

3.1.1 Combining and Splitting Value

Although in the transactions, it would be possible to handle coins individually, it would be inconvenient to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain one or more inputs and outputs. Therefore, there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts: one for the payment, and one returning the change back to the sender if the output is higher than what the user wishes to pay.

In this way, the Bitcoin achieves two goals:

- it introduces the idea of change
- it is possible to identify the unspent coins or balance of a user by verifying the outputs from its previous transactions

This mechanism is managed using a Forth-like scripting language which verifies the essential conditions to claim the coins. There are two scripts: the "Pay-to-PubKeyHash" (P2PKH), which requires only one signature from the owner to authorize a payment, and the "Pay-to-ScriptHash" (P2SH), which is typically used as multisignature addresses, but it also used for a variety of transaction types and supports future developments [And12].

3.2 Blockchain

The Bitcoin Blockchain is a public ledger that records Bitcoin transaction history in form of blocks. In each block, the transactions are stored using Markle Tree [Mer87], and a relatively secure timestamp and a hash of the previous block is stored. To guarantee independent verification of the chain of ownership each node stores its own copy of the Blockchain. Then, the nodes of the network may confirm the transactions, add them to their copy of Blockchain and then broadcast these additions to the other nodes. About every 10 minutes, a new block is created, added to the Blockchain, and quickly published to all nodes, without requiring a central party. Bitcoin is based on a *timestamp server* functionality, that works by taking a hash of a block of items to be timestamped and widely publishing the hash.
Moreover, it is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest

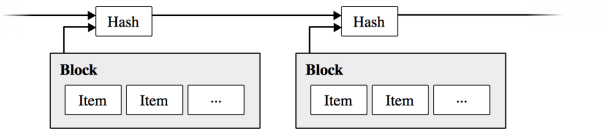

Figure 3.3: Transactions Ownership in bitcoin.[Nak].

proof-of-work chain, which he may obtain by querying network nodes until he verifies to have the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. By linking the transaction to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirms the network has accepted it.

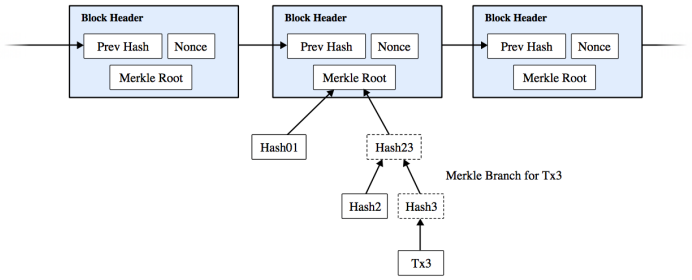

Figure 3.4: Simplified Payment Verification [Nak].

3.3 Proof-of-Work

The Blockchain is kept consistent, complete, and unalterable by miners, which repeatedly group newly broadcast transactions into a block, which is then broadcast to the network and verified by recipient nodes. With the term *mining*, we indicate a record-keeping service done through the use of computer processing power.

Pending transactions that have occurred over Bitcoin network and are therefore awaiting approval by a miner, are held in what is known as the Mempool (Memory pool) [Aso18].

To be accepted by the rest of the network, a new block must contain a Proof-of-work (PoW). The PoW requires miners to find a number called *nonce*, that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The number of zero bits required is provided by the target. The target is a very large number and known to every Bitcoin client. For a block to be accepted, the resulting hash has to be a value less compared to the current target and it has to contain a certain number of leading zero bits to be less than that.

This proof is easy for any node in the network to verify, but extremely time-

consuming to generate, as for a secure cryptographic hash, miners must try many different nonce values before meeting the difficulty target. Once the nonce value is found, the block cannot be modified without redoing the entire calculation again.

As we have mentioned in the Section 2.1.1, make a modification of the block in the ledger is very hard, since if an attacker wants to modify a past block, he has to do the proof-of-work of the block and all blocks after it.

The proof-of-work determines the representation in majority decision making. In Proof-of-work is essentially one-CPU-one-vote and the majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. Suppose that a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains.

The Figure 3.5 summarizes the important steps to validate a transaction in Bitcoin.

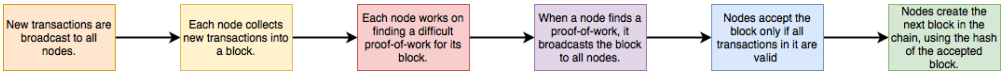

Figure 3.5: The steps to run the network [Nak].

To guarantee that the nodes stay honest, it was introduced incentives. For example, the first transaction in a block is an important transaction that creates a new coin owned by the miner, that created of the block. This adds an incentive for nodes to support the network and create a way to initially distribute coins into circulation since there is no central authority to issue them.

The incentive can also be created with transaction fees, for example, if the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction.

In order to receive the incentive, miners mine in the form of the so-called *mining pools*. This means that all miners that are associated with a pool work cooperatively to mine a particular block under the control of a pool manager. In case of successful mining, the manager distributes the reward among all the associated miners proportional to the resources expended by each miner.

This is important, in case of an attacker inside the network. In fact, if he is able to assemble more CPU power than all the honest nodes (*51% attack*), he would have to choose between using it to defraud people by stealing back his payments or using it to generate new coins. For him, it is more profitable to play by the rules, which favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth [Nak].

3.4 Consensus Protocol

The nodes interact over a network and collaboratively construct the Blockchain without trusting on a central authority. It is necessary to consider the case in which the individual nodes crash, behave maliciously, act against the common goal, or the network communication becomes interrupted. To offer a continuous service, the nodes run a fault-tolerant consensus protocol to ensure that they all agree on the order in which entries are appended to the Blockchain. In order to add a new block in the Blockchain, every miner has to follow a set of rules specified in the consensus protocol. Bitcoin achieves the distributed consensus, through the Proof of Work (PoW) based consensus algorithm.

The distributed consensus protocol is based on the following rules:

- input and output values are rational
- transactions only spend unspent outputs
- all inputs being spent have valid signatures
- no coinbase transaction outputs, which is a unique type of bitcoin transaction only created by a miner, was spent within 100 blocks
- no transactions spend inputs with a lock time before the block in which they are confirmed

Bitcoin is considered as secure and as robust, thanks to its consensus model [Con+18].

3.5 Privacy

Figure 3.6 shows the difference between traditional privacy method and new privacy model in Bitcoin. The traditional banking model reaches a level of privacy by limiting access to information to the parties involved and using a third party as a trusted party. The use of the ledger, that announces all transactions publicly precludes this method, but privacy may still be maintained by keeping public keys anonymous. The other nodes may see that someone is sending an amount to someone else, but without information linking the transaction to anyone.
As an additional firewall protection, a new key pair should be used for each transaction to keep them from being linked to a common owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

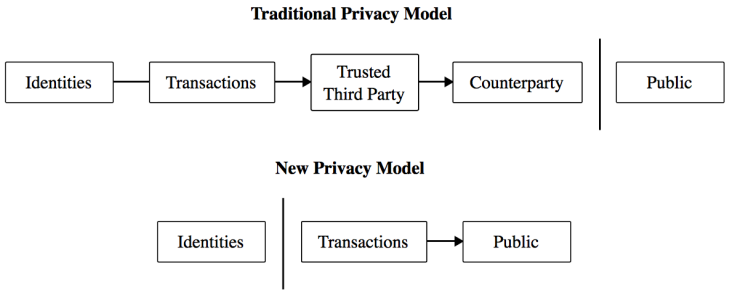

Figure 3.6: Comparison between Traditional Privacy Model and New Privacy Model [Nak].

3.6 Wallets

In Bitcoin, the wallet refers to the data structure used to store and manage a user's keys, necessary to transact bitcoins.

In each transaction, the destination address, called *Bitcoin address*, is generated by performing a series of irreversible cryptographic hashing operations on the user's public key. Moreover, a user may have multiple addresses by creating several public keys, and these addresses might be associated with one or more wallets. The private key of the user is used to spend the owned bitcoins in the form of digitally signed transactions. Thus, using the hash of the public key as receiving address permits to the users a certain degree of anonymity, and as we have seen in the section 3.6, it is recommended the practice to use different Bitcoin address for each receiving transaction [Con+18].

A common misconception is that wallets contain bitcoin, that are recorded in the Blockchain on the Bitcoin network instead. In fact, users control their own coins on the network by signing transactions with the keys in their wallets.

There are two primary types of wallets: non-deterministic wallet and deterministic wallet, distinguished by if the keys they contain are linked to each other or not.

Non-deterministic wallet
In the non-deterministic wallet, each key is independently generated from a random number. In this case, the keys are not related to each other, and is also known as a JBOK wallet from the phrase "Just a Bunch Of Keys."

Deterministic wallet
In the deterministic wallet, all the keys are derived from a single master key, known as the seed. All the keys in this type of wallet are linked to each other and can be generated again if the user has the original seed. There are different key derivation techniques used in deterministic wallets, but the most commonly used derivation method uses a tree-like structure and is known as a hierarchical deterministic or HD wallet.

The use of non-deterministic wallets is discouraged for anything other than simple tests because they are simply too cumbersome to back up and use. In this thesis, we consider a deterministic HD wallet with a mnemonic seed for backup [Ant14].

3.6.1 Deterministic Wallets

Deterministic, or "seeded," wallets contains private keys that are all derived from a common seed, through the use of a one-way hash function. The seed is a randomly generated number that is combined with other data, such as an index number or *chain code* to derive the private keys. In a deterministic wallet, the seed is sufficient to recover all the derived keys, and therefore a single backup at creation time is sufficient. The seed is also sufficient for a wallet export or import, allowing for easy migration of all the user's keys between different wallet implementations.
Figure 3.7 and 3.8 compare the non-deterministic wallet and the deterministic wallet.

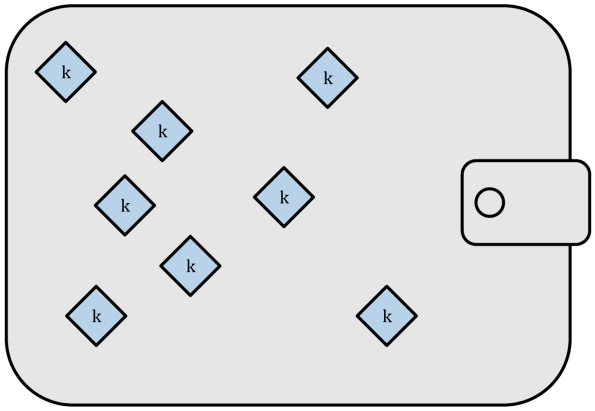

Figure 3.7: Non-deterministic (random) wallet: a collection of randomly generated keys.

3.6.2 HD Wallets- BIP 32/44

As we have mentioned in Section 3.6.1, deterministic wallets were developed to permit to derive many keys from a single *seed*. The HD wallet is also defined as the BIP-32 standard and is the most advanced form of deterministic wallets.

The keys are derived in a tree structure, such that a parent key may derive a sequence of children keys, each of which may derive a sequence of grandchildren keys, and so on, to an infinite depth. This structure is illustrated in Figure 3.9. The structure of HD wallets offers two major advantages over random keys. First, the tree structure may be used to express additional organizational meaning, for instance when a specific branch of subkeys is used to receive incoming payments and a different branch is used to receive change from outgoing

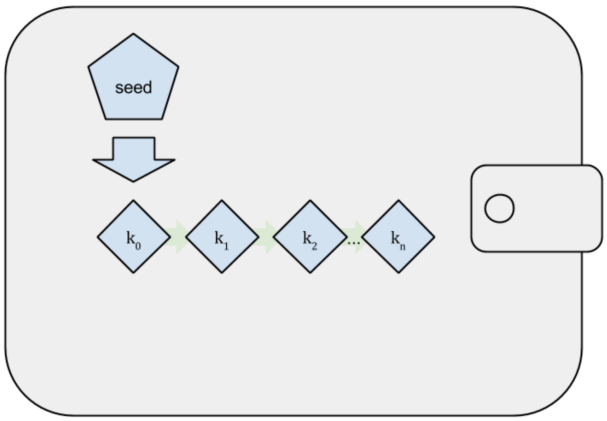

Figure 3.8: Deterministic (seeded) wallet: a deterministic sequence of keys derived from a seed..

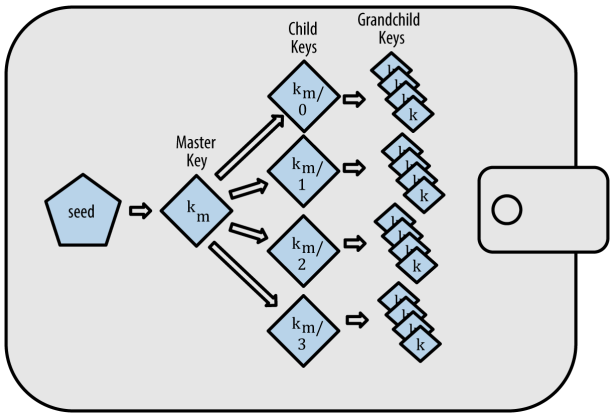

Figure 3.9: Type-2 HD wallet: a tree of keys generated from a single seed.

payments. The second advantage of HD wallets is that users may create a sequence of public keys without having access to the corresponding private keys. This allows HD wallets to be used on an insecure server or in a receive only capacity, issuing a different public key for each transaction [Ant14].

3.6.3 Seeds and Mnemonic Codes (BIP-39)

BIP-39 standard gives the possibility to create seeds from a sequence of English words that are easy to transcribe, export, and import across wallets. In this way the user instead of remembering the seed in hexadecimal, he may recover their data through a sequence of 12 or 24 English words. For example:

Seed in hex: 0C1E24E5917779D297E14D45F14E1A1A
Seed as a sequence of the following words: *army van defence carry jealous true garbage claim echo media make crunch*

3.6.4 Creating an HD Wallet from the Seed

HD wallets are composed by a single root seed, which may be a 128-, 256-, or 512-bit random number. Every key in the HD wallet is derived from this root seed, which gives the possibility to recreate the entire HD wallet from that seed in any compatible HD wallet. This makes it easy to back up, restore, export, and import HD wallets containing thousands or even millions of keys by simply transferring only the mnemonic that the root seed is derived from.

The process of how creating the master keys and master chain code for an HD wallet is shown in Figure 3.10.

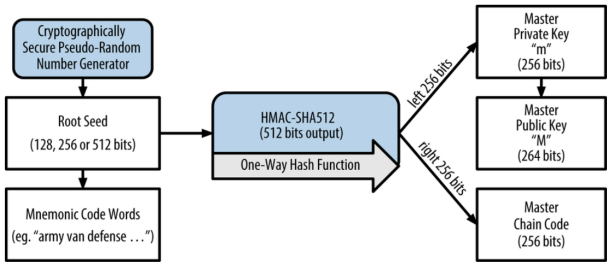

Figure 3.10: Creating master keys and chain code from a root seed [Ant14].

The root seed is input into the HMAC-SHA512 algorithm and the resulting hash is used to create a master private key (m) and a master chain code(c). The master private key (m) then creates a corresponding master public key (M) using the normal elliptic curve multiplication process m * G.
Whereas, the chain code (c) is used to introduce entropy in the function that creates child keys from parent keys.

3.6.5 Private child key derivation

Subsequently, HD wallets use a child key derivation (CKD) function to derive child keys from parent keys. The child key derivation functions are based on a one-way hash function that combines:

- A parent private or public key (ECDSA uncompressed key)
- A seed called a chain code (256 bits)
- An index number (32 bits)

As we have already mentioned in Section 3.6.4, the chain code is used to introduce deterministic random data to the process so that knowing the index and a child key is not sufficient to derive other child keys. The initial chain code seed (at the root of the tree) is made from the seed, while subsequent child chain codes are derived from each parent chain code.

The children keys are generated as follows:

1. The parent public key, chain code, and the index number are combined and hashed with the HMAC-SHA512 algorithm to produce a 512-bit hash.
2. This 512-bit hash is divided into two 256-bit halves.
3. The right-half 256 bits of the hash output become the chain code for the child.
4. The left-half 256 bits of the hash and the index number are added to the parent private key to produce the child private key.

In Figure 3.11, we see this illustrated with the index set to 0 to produce the "zero" (first by index) child of the parent. Modifying the index allows us to

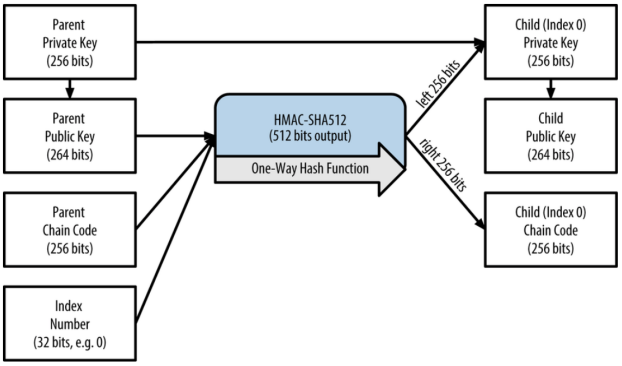

Figure 3.11: Extending a parent private key to create a child private key [Ant14].

extend the parent and create the other children in the sequence, e.g., Child 0, Child 1, Child 2, etc. Each parent key may have 2^{31} children (2^{31} is half of the entire 2^{32} range available because the other half is reserved for a special type of derivation). Repeating the process one level down the tree, each child may, in turn, become a parent and create its own children, in an infinite number of generations.

3.6.6 Using derived child keys

Child private keys are indistinguishable from non-deterministic keys because the derivation function is a one-way function, in fact, the child key may not be used to recover the parent key. Moreover, the child key also may not be used to find any siblings, in fact, if you have the n^{th} child, you are not able to find its siblings, such as the n–1 child or the n+1 child, or any other children that are part of the sequence. Only the parent key and chain code may derive all the children. It needs both the child private key and the child chain code to start a new branch and derive grandchildren. So, the child private key may be used to make a public key and a bitcoin address. Then, it may be used to sign transactions to spend anything paid to that address.

3.6.7 Extended keys

The key derivation function may be used to create children at any level of the tree, based on the three inputs: a key, a chain code, and the index of the desired child. The two essential ingredients are the key and chain code and combined these are called an *extended key*. The term extended key might also be though as extensible key because such a key may be used to derive children.

Extended keys are stored and represented simply as the concatenation of the 256-bit public key and 256-bit chain code into a 512-bit sequence. There are two types of extended key: the extended private key, which is the combination of a private key and chain code and may be used to derive child private keys. And the extended public key, that is a public key and chain code, which may be used to create child public keys.

So, it is important to highlight that through the extended private key may create a complete branch, whereas through the extended public key may only create a branch of public keys. This gives us two ways to derive a child public key: either from the child private key or directly from the parent public key.

Thus, this technique permits to produce an infinite number of public keys and bitcoin addresses, but cannot spend any of the money sent to those addresses. The Figure 3.12 illustrates the mechanism for extending a parent public key to derive child public keys.

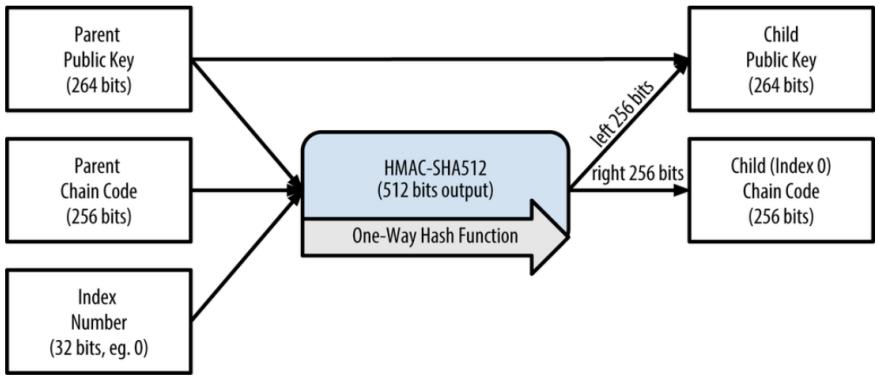

Figure 3.12: Extending a parent public key to create a child public key [Ant14].

In this solution, there is a potential risk. Since, an extended public key contains the chain code, if a child private key is known, it may be used with the chain code to derive all the other child private keys.

To solve this risk, HD wallets use an alternative derivation function called *hardened derivation*. The hardened derivation function uses the parent private

key to derive the child chain code, instead of utilizing the parent public key. This creates a "firewall" in the parent/child sequence, with a chain code that may not be used to compromise a parent or sibling private key. The difference compares to the normal child private key derivation, is that the parent private key is used as input to the hash function, instead of the parent public key, as shown in the diagram in 3.13. The result of using the hardened private

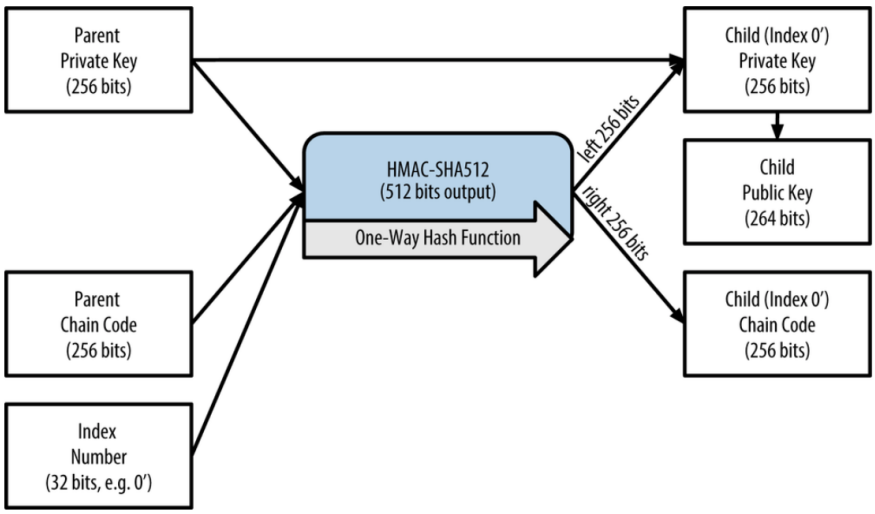

Figure 3.13: Hardened derivation of a child key; omits the parent public key [Ant14].

derivation function is completely different from using normal derivation. The resulting "branch" of keys may be used to produce extended public keys that are not vulnerable because the chain code they contain may not be exploited to reveal any private keys.

Thus, for using the convenience of an expand public key to derive branches of public keys, without having the risk of a leaked chain code, it is necessary to derive it from a hardened parent, rather than a normal parent. As a best practice, the level-1 children of the master keys are always derived through the hardened derivation, to prevent compromise of the master keys.

3.6.8 Index numbers for normal and hardened derivation

The index number utilized in the derivation function is a 32-bit integer. In order, to distinguish between keys derived through the normal derivation function versus keys derived through hardened derivation, this index number is split into two ranges.

- Index numbers between 0 and 2^{31}–1 (0x0 to 0x7FFFFFFF) are used only for normal derivation.

- Index numbers between 2^{31} and 2^{32}–1 (0x80000000 to 0xFFFFFFFF) are used only for hardened derivation.

To make the index number easier to utilize, the index number for hardened children is displayed starting from zero, but with a prime symbol. The first normal child key is therefore displayed as 0, whereas the first hardened child (index 0x80000000) is displayed as 0'. In sequence then, the second hardened key would have index 0x80000001 and would be displayed as 1', and so on. When you see an HD wallet index i', that means 2^{31}+i.

3.6.9 HD wallet key identifier (path)

To simplify the writing of keys in an HD wallet, a convention is used, where each level of the tree is separated by a slash (/) character (see Figure 3.14). Private keys derived from the master private key start with "m". Whereas, public keys derived from the master public key start with "M". Thus, the first child private key of the master private key is m/0. The first child public key is M/0. The second grandchild of the first child is m/0/1, and so on.
For instance, the path m/x/y/z describes the key that is the z-th child of key m/x/y, which is the y-th child of key m/x, which is the x-th child of m.

HD path	Key described
m/0	The first(0) child private key from the master private key(m)
m/0/0	The first grandchild private key from the first child (m/0)
m/0'/0	The first normal grandchild of the first hardened child (m/0')
m/1/0	The first grandchild private key of the second child (m/1)
M/23/17/0/0	The first great-great-grandchild public key of the first great-grandchild of the 18th grandchild of the 24th child

Figure 3.14: HD wallet path examples [Ant14].

3.6.10 Navigating the HD wallet tree structure

The HD wallet allows having tremendous flexibility in the creation of the tree structure. In fact, every parent extended key may have 4 billion children: 2 billion normal children and 2 billion hardened children. Each of those children may have further 4 billion children, and so on.

On the other hand, it is quite difficult to navigate this infinite tree. Two BIPs, Bitcoin Improvement Proposals, offer a solution to deceive this complexity by creating some proposed standards for the structure of HD wallet trees.

- **BIP-43:** it proposes to use the first hardened child index as a special identifier that means the "purpose" of the tree structure. For example, an HD wallet using the only branch m/i'/ is intended to signify a specific purpose and that purpose is identified by index number "i."
- **BIP-44:** it extends the previous proposal and offers a multi-account structure as "purpose" number 44' under BIP-43.

BIP-44 specifies the structure as consisting of five predefined tree levels:

```
m / purpose' / coin_type' / account' / change / address_index
```

Purpose'
In the case of BIP-44 is always set to 44'.

Coin_type'
The second-level "coin_type" specifies the type of cryptocurrency coin, allowing for multicurrency HD wallets where each currency has its own subtree under the second level. There are three currencies defined for now: Bitcoin is m/44'/0', Bitcoin Testnet is m/44'/1', and Litecoin is m/44'/2'.

Account
The third level of the tree is "account", which allows users to subdivide their wallets into separate logical subaccounts, for accounting or organizational purposes. For instance, an HD wallet might contain two bitcoin "accounts": m/44'/0'/0' and m/44'/0'/1'. Each account is the root of its own subtree.

Change
On the fourth level, "change", an HD wallet has two subtrees, one for creating receiving addresses and one for creating change addresses.

Address_index
Usable addresses are derived by the HD wallet as children of the fourth level, making the fifth level of the tree the "address_index". For example, the third receiving address for bitcoin payments in the primary account would be M/44'/0'/0'/0/2 [Ant14].

The Figure 3.15 shows more examples.

HD path	Key described
M/44 ' /0 ' /0 ' /0/2	The third receiving public key for the primary bitcoin account
M/44 ' /0 ' /3 ' /1/14	The fifteenth change-address public key for the fourth bitcoin account
m/44 ' /2 ' /0 ' /0/1	The second private key in the Litecoin main account, for signing transactions

Figure 3.15: BIP-44 HD wallet structure examples [Ant14].

3.7 Building Blocks Primitive of Bitcoin

The Bitcoin Technology offers some guarantees, that has been used as building blocks to create several other applications.

This section summarizes all these guarantees:

- **No-Double Spend:** one of the main important aspects of Bitcoin is that ensures that no UTXO (Unspent Transaction Output) can be spent twice.
- **Immutability:** As we've seen in the previous chapter, once a transaction is recorded in the blockchain and sufficient work has been added with subsequent blocks, the transaction's data becomes immutable. In fact once deeply recorded, the computation and energy required to change make change practically infeasible.
- **Neutrality:** Anyone can create a valid transaction and trust they will be able to transmit that transaction and have it included in the blockchain at any time. So, the decentralized Bitcoin network propagates valid transactions regardless of the origin or content of those transactions.
- **Secure timestamping:** The consensus rules do not accept any block whose timestamp is too far in the past or future. In this way, this ensures that timestamps on blocks can be trusted.
- **Authorization:** Digital signatures, validated in a decentralized network, provide authorization guarantees. Scripts that ask a requirement for a digital signature are not executed without authorization by the holder of the private key contained in the script.
- **Auditability:** All transactions are public and can be audited. While blocks may be linked back in an unbroken chain to the genesis block.
- **Accounting:** It is not possible to destroy or create bitcoin in a transaction. Since, in any transaction, the value of inputs is equal to the value of outputs plus fees, whereas the outputs cannot exceed the inputs.
- **Nonexpiration:** If a transaction is valid today, it will be valid also in the near future, as long as the inputs remain unspent and the consensus rules do not change.

- **Integrity:** Each bitcoin transaction signed with SIGHASH_ALL or parts of a transaction signed by another SIGHASH type cannot be modified without invalidating the signature, and so invalidating the transaction itself.
- **Transaction Atomicity:** Bitcoin transactions are atomic. They are either valid and confirmed (mined) or not. In case of partial transactions, they cannot be mined and there is no interim state for a transaction.
- **Discrete (Indivisible) units of Value:** Transaction outputs are discrete and indivisible units of value, so they cannot be divided or partially spent.
- **Quorum of Control:** Multisignature constraints in scripts requires a quorum of authorization, predefined in the multisignature scheme. The M-of-N requirement is imposed by the consensus rules.
- **Timelock/Aging:** A script clause containing a relative or absolute timelock can only be executed after its age exceeds the time specified.
- **Replication:** The decentralized storage of the blockchain ensures that when a transaction is confirmed a sufficient number of times, it is replicated across the network and becomes durable and resilient to power loss, data loss, etc.
- **Forgery Protection:** A transaction can only spend validated, existing outputs.
- **Consistency:** In the absence of miner divisions, blocks that are stored in the ledger are subject of reorganization or disagreement with exponentially decreasing likelihood, based on the depth at which they are recorded.
- **Recording External State:** A transaction can commit a data value, utilizing OP_RETURN, representing a state transition in an external state machine.
- **Predictable Issuance:** Less than 21 million bitcoin will be issued, at a predictable rate [Ant14].

These building blocks offered by Bitcoin may be the basis to compose a trust application, that is utilized to create applications. One example of these applications is Payment Channels, that it will be analyzed in the next Chapter.

The payment Channels uses mainly these building blocks of Bitcoin:

1. Quorum Of Control
2. Time Lock
3. No double Spend
4. Nonexpiration
5. Censorship Resistance
6. Authorization

CHAPTER 4
Lightning Network

In this Chapter, we will analyze in more details Lightning technology, explaining the important characteristics that it is necessary to know in order to understand completely the work that will be presented in the next Chapters.

As we have seen in the Chapter 2, Lightning Network is a decentralized network using smart contract functionality in the Blockchain, in order to create instant payments across a network of participants.
Moreover, this technology aims to solve the scalability problem of Bitcoin, without publishing all transactions on the Blockchain, but just the initial and final status.

In this Chapter, we will analyze first the Payment Channels, then Lightning Network and subsequently its benefits.

4.1 Payment Channel

Payment channels, also known as Micropayment Channel, is a class of trustless techniques designated to exchange bitcoin between two parties, outside of the Bitcoin Blockchain [she18]. In fact, these transactions are held off-chain, as a sort of notes for eventual batch settlement. Since the transactions are not settled, they may be exchanged without the usual settlement latency, allowing extremely high transaction throughput, low (sub-milliseconds) latency, and fine (satoshi-level) granularity.

We define the concept of state channel as a virtual construct represented by the exchange of state between two parties, off-chain. The payment channel is a state channel, where the state channel being altered is the balance of virtual currency [Ant14].

4.2 Basic concepts and terminology

In this paragraph, we will define the main concepts and terminology to know, to understand completely the mechanism behind Lightning Network. The concepts that we analyze are funding, commitment and settlement transaction.

Funding transaction
The Funding transaction must be transmitted to the network and mined to establish the channel. In the example of a payment channel, the locked state is the initial balance of the channel.

Commitment transactions
The commitment transactions are signed transactions exchanged by the parties, that alter the initial state. These transactions are held-off chain by each party pending the channel closure.

Settlement transaction
The channel may be closed either cooperatively, by submitting a final settlement transaction to the ledger, or unilaterally, by either party submitting the last commitment transaction to the Blockchain. The settlement transaction represents the final state of the channel and is settled on the ledger.

During the entire lifetime of the channel, just two transactions are broadcast to the Blockchain: the funding and the settlement transactions. Between these two states, the two parties can exchange any number of commitment transactions that are never submitted to the Blockchain, as it is shown in Figure 4.1.

4.3 Simple Payment Channel

In order to explain how the payment channels work, we consider two participants: Alice and Bob. Suppose that Alice has to pay a service offering by Bob, and they have a micropayment channel. Both Alice and Bob use special software that handles the payment channel, for example, Alice is running the software in her browser, Bob is running it on a server. In more details, the software is composed of basic bitcoin wallet functionality and may create and sign Bitcoin transactions.

Funding transaction
To set up the payment channel, Alice and Bob establish a 2-of-2 multisignature address, with each of them holding one of the keys. Alice submits a deposit and found an address. The first transactions by Alice, paying to the multisignature address, is the funding and anchor transaction for the payment channel. The funding transaction establishes the maximum amount, called deposit, that may be transmitted in this channel, so setting the channel capacity. Once the anchor transaction is funding, Alice may use the service.

Commitment transactions
Alice's software creates and signs a commitment transaction that changes the channel balance to credit some money to Bob's address and refunds the money to Alice. The transaction is signed by Alice, and it requires two signatures (2-of-2), but only has Alice's signature.
When Bob's server receives this transaction, it adds the second signature and returns it to Alice with the service. Now both parties have a fully signed

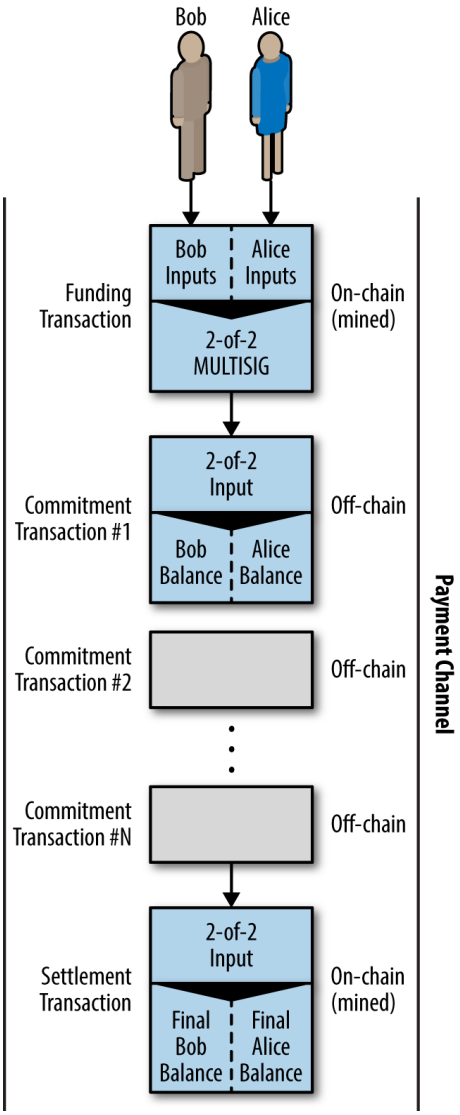

Figure 4.1: A payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions [Ant14].

commitment transaction that either can redeem, representing the correct up-to-date balance of the channel. Neither party broadcasts this transaction to the network.

Suppose that Alice wants to use the service a second time. In this case, Alice's software creates and signs another commitment transaction that consumes the same 2-of-2 output from the funding transaction. The second commitment transaction allocates one output to Bob's address and another one to Alice's address. Bob's software signs and returns the second commitment transaction, together with the service.

Settlement transaction If Alice wants to stop to use the service, either Bob or Alice may transmit the final state of the transaction for settlement. This last transaction is the settlement transaction and pays Bob for the service that Alice utilized, refunding the remainder of the funding transaction to Alice.

4.4 Timelocks

Let us consider again Alice and Bob with a micropayment channel, there are two possible scenarios when the state channels might fail:

- Once the funding transactions happen, Alice needs Bob's signature to get any money back. If Bob disappears, Alice's funds are locked in a 2-of-2 and effectively lost. This channel, as constructed, leads to loss of funds if one of the parties disconnects before there is at least one commitment transaction signed by both parties.
- While the channel is running, Alice may take any of the commitment transactions Bob has countersigned and transmitted one to the blockchain. So, the channel fails because Alice may cheat by broadcasting a prior commitment that is in favour.

A proposed solution to solve the problem above is using *timelocks* (nLockTime).

Refund Transaction
As we have seen, Alice cannot risk funding a 2-of-2 multisig unless she has a guaranteed refund. To solve this issue, Alice constructs with the funding transaction, a refund transaction at the same time. She signs the funding transaction but without transmitting it to Bob. Alice transmits only the refund transaction to Bob, in order to obtain his signature.

The refund transaction acts as the first commitment transaction and it is linked to a timelock, that establishes the upper bound for the channel's life. In this case, Alice might set the nLocktime to 30 days or 4320 blocks into the future. All subsequent commitment transactions have to have a shorter timelock, in this way they can be redeemed before the refund transaction.

Once Alice has a fully signed refund transaction, she may transmit the signed funding transactions knowing that she may eventually after the timelock expires, redeem the refund transaction even if Bob disappears.

Each commitment transaction that the parties exchange during the life of the channel will be time-locked into the future. But the delay will be slightly shorter for each commitment, in this way the most recent commitment may be redeemed before the prior commitment it invalidates. Using the nLockTime, neither party can successfully broadcast any of the commitment transactions until their timelock expires. If all goes well, they will cooperate and close the channel cooperatively with a settlement transaction, making it unnecessary to transmit an intermediate commitment transaction. Otherwise, the most recent commitment transaction may be propagated to settle the account and invalidate all prior commitment transactions.

The ability to broadcast a commitment earlier ensures it will be able to spend the funding output and preclude any other commitment transaction from being redeemed by spending the output. The guarantees offered by the Bitcoin Blockchain, preventing double-spends and enforcing timelocks, effectively allow each commitment transaction to invalidate its predecessors.

Thus, state channels use timelocks to enforce smart contracts across a time dimension. Despite all the advantages, Timelocks presents two drawbacks:

- By establishing a maximum timelock when the channel is first opened, they limit the lifetime of the channel.
- The second problem is that since each subsequent commitment transaction must decrement the timelock, there is an explicit limit on the number of commitment transactions that may be exchanged between the parties.

4.5 Asymmetric Revocable Commitments

In order to punish the other party if they try to cheat, it is possible to use a revocation key. To explain revocation keys, we will construct again a payment channel between Alice and Bob. Alice and Bob start the channel by collaboratively constructing a funding transaction, which locks the channel state in a 2-of-2 multisig, each funding the channel with 5 bitcoin.

In this scenario, instead of creating a single commitment transaction that both parties sign, Alice and Bob create two different commitment transactions that are asymmetric.

Alice has a commitment transaction with two outputs. The first output pays Bob the 5 bitcoin he is owed immediately. The second output pays Alice the 5 BTC she is owed, but only after a timelock of 1000 blocks. In this way each party has a commitment transaction, spending the 2-of-2 funding output.
This input is signed by the other party. At any time the party holding the transaction may also sign and broadcast. If they broadcast the commitment transaction, it pays the other party immediately whereas they have to wait for a short timelock to expire. We refer to this mechanism of penalty as Lightning Network Penalty.

Therefore, the revocation key permits the wronged party to punish the cheater by taking the entire balance of the channel. The revocation key is composed of two secrets, each half generated independently by each channel participant. This is quite similar to a 2-of-2 multisig but constructed utilizing elliptic curve arithmetic, so that both parties know the revocation public key but each party knows only half the revocation secret key.

For each transaction, both parties reveal their half of the revocation secret to each other, this means to request the penalty output if this revoked transaction is ever broadcast.

Each of the commitment transactions has a delayed output. The redemption script for that output allows one party to redeem it after a total number of blocks, or the other party to redeem it if they have a revocation key, penalizing transmission of a revoked commitment.

4.6 Hash Time Lock Contracts

An important feature is possible to add to payment channels, to extend them with a special type of smart contract that allows the participants to commit funds to a redeemable secret, with an expiration time. This feature is called Hash Time Lock Contract, HTLC, and may be used in both bidirectional and routed payment channels.

To create an HTLC, the intended recipient of the payment must create a secret R and then they create the hash of the secret H:

H= Hash(R)

This produces a hash H that may be included in an output's looking script. This technique expects whoever knows the secret may use it to redeem the output. The secret R is also seen as a preimage to the hash function, and is just the data that is used as input to a hash function.

The second part of an HTLC is the time lock component. In fact, if the secret is not revealed, the payer of an HTLC may get a refund after some time. This result is achieved with the use of the CHECKLOCKTIMEVERIFY.

4.7 Lightning Network

The Lightning Network is a routed network composed of bidirectional payment channels connected end-to-end. This network permits the participant to route a payment from channel to channel without trusting any of the intermediaries. The Lightning network was first introduced by Joseph Poon and Thadeus Dryja in February 2015 [PD16], and it was based on the concept of payment channels.

To explain Lightning Network, suppose to have a network with five participants: Alice, Bob, Carol, Diana and Eric. These five participants have opened payment channels with each other, in pairs. Alice has a payment channel with Bob, Bob is connected to Carol, Carol to Diana, and Diana to Eric. For simplicity, we assume that each channel is funded with 2 bitcoin by each participant, for a total capacity of 4 bitcoin in each channel, Figure 4.2.

Consider the case, in which Alice wants to pay Eric 1 bitcoin. The scenario is shown in Figure 4.3.

Step 1: Alice is using a Lightning Network node that is storing her payment channel to Bob and is able to discover routes between payment channels. Alice's LN node is able to connect to Eric's LN node through the Internet.

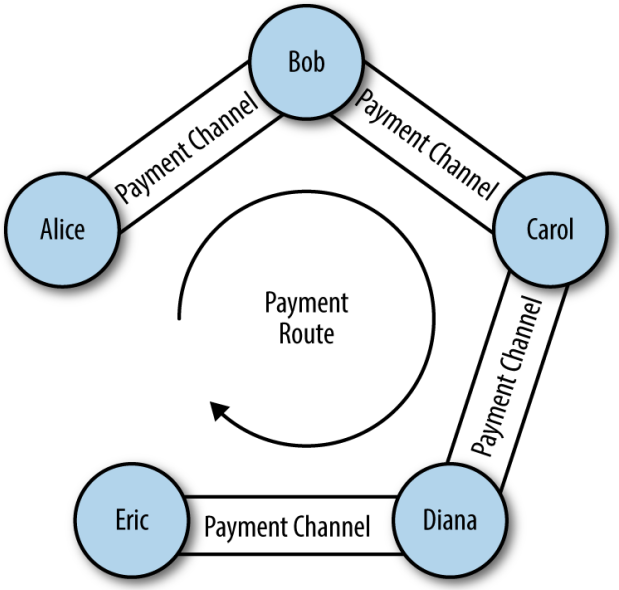

Figure 4.2: A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric [Ant14].

In order to receive the payment from Alice, Eric's Lightning Network node creates a secret R utilizing a random number generator, he calculates a hash H of the secret R and sends this hash to Alice's.

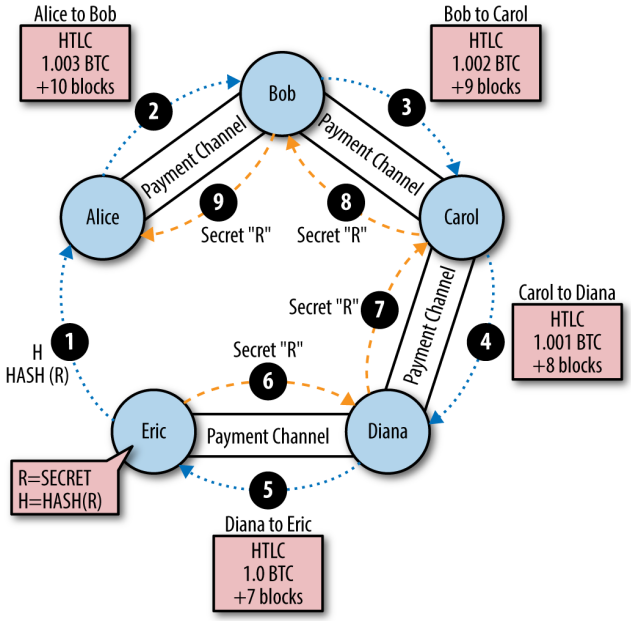

Figure 4.3: Step by step payment routing through a Lightning Network [Ant14].

Step 2: Alice's node constructs an HTLC, with a 10-block refund timeout, for an amount of 1.003 bitcoin. The extra 0.003 will be utilized to compensate the intermediate nodes for their participation in this payment route. Alice offers this HTLC to Bob, deducting 1.003 bitcoin from her channel balance with Bob and committing it to the HTLC. The channel balance between Alice and Bob is now expressed by commitment transactions with three outputs: 2 bitcoin balance to Bob, 0.997 bitcoin balance to Alice, 1.003 bitcoin committed in Alice's HTLC.

Step 3: If Bob is able to have the secret R within the next 10 blocks, he may claim the 1.003 locked by Alice. Thus, Bob's node constructs an HTLC on his payment channel with Carol. Bob's HTLC commits 1.002 bitcoin to hash H for 9 blocks, which Carol may redeem if she has secret R. The channel balance between Bob and Carol is now: 2 to Carol, 0.998 to Bob, 1.002 committed by Bob to the HTLC.

Step 4: If Carol gets R within the next nine blocks, she may claim 1.002 bitcoin locked by Bob. Thus, she may make an HTLC commitment on her channel with Diana. She commits an HTLC of 1.001 bitcoin to hash H, for eight blocks, which Diana may redeem if she has secret R. The channel balance between Carol and Diana is now: 2 to Diana, 0.999 to Carol, 1.001 committed by Carol to the HTLC.

Step 5: Now, Diana is able to offer an HTLC to Eric, committing 1 bitcoin for seven blocks to hash H. The channel balance between Diana and Eric is now: 2 to Eric, 1 to Diana, 1 committed by Diana to the HTLC.

Step 6: Eric has secret R. He may claim the HTLC offered by Diana. He sends R to Diana and claims the 1 bitcoin, adding it to his channel balance. The channel balance is now: 1 to Diana, 3 to Eric.

Step 7: Then, Diana has secret R. Therefore, she may now claim the HTLC from Carol. Diana transmits R to Carol and adds the 1.001 bitcoin to her channel balance. Now the channel balance between Carol and Diana is 0.999 to Carol, 3.001 to Diana. Note that Diana has "earned" 0.001 for participating in this payment route.

Step 8: The secret R allows each participant to claim the outstanding HTLCs. Carol claims 1.002 from Bob, setting the balance on their channel to 0.998 to Bob, 3.002 to Carol.

Step 9: Finally, Bob claims the HTLC from Alice. Their channel balance is updated as 0.997 to Alice, 3.003 to Bob.

Alice has paid Eric 1 bitcoin without opening a channel to Eric. None of the intermediate parties in the payment route had to trust each other. For the short-term commitment of their funds in the channel, they are able to earn a small fee, with the only risk being a small delay in a refund if the channel was closed or the routed payment failed.

4.8 Transport and Routing

All communication between the nodes in Lightning Network are encrypted, and every node has a long-term public key that they use as an identifier and to authenticate each other.

The path between Alice and Charlie, presented in the previous section is known just to Alice's node. All other nodes in the payment route know only the adjacent nodes.

This feature is very crucial in Lightning Network since it ensures the privacy of payments and makes it very difficult to apply surveillance.

This is possible because Lightning Network implements an onion-routed protocol based on a scheme called Sphinx. This routing protocol ensures that a payment sender can construct and communicate a path through the Lightning Network such that:

- Intermediate nodes may decrypt just their portion of route information and find the next hop.
- Intermediates nodes may not learn about any other nodes that are part of the path, except the previous and the next hops
- Intermediates nodes may not know the length of the payment path or their own position in that path.
- Each part of the path is encrypted in a way that a network level attacker may not associate the packets from different parts of the path to each other.
- Unlike Tor, that is an onion-routed anonymization protocol on the Internet, there are no "exit nodes" that can be placed under surveillance. The payments do not need to be transmitted to the Bitcoin Blockchain; the nodes just update channel balances. [Ant14]

4.9 Lighnting Network benefits

As Lightning Network may adopt benefits from Bitcoin, also the vice-versa is true. In fact, the Bitcoin network may gain a significant increase in capacity, privacy, granularity, and speed, without sacrificing the principles of trustless operation without intermediaries:

- **Privacy** While the payments on the Bitcoin Blockchain are public, the Lightning Network payments instead are much more private. The participants in a route know payments propagated across their channels, they do not see the sender or recipient.
- **Fungibility** A Lightning Network makes it much more difficult to apply surveillance on bitcoin, in a way to increase the fungibility of the currency.

- **Speed** Bitcoin transactions using Lightning Network are settled in milliseconds, while HTLCs are cleared without committing transactions to a block.
- **Granularity** A Lightning Network permits payments at least as small as the Bitcoin "dust" limit, maybe even smaller.
- **Capacity** A Lightning Network increases the capacity of the Bitcoin system by different orders of magnitude. It is not possible to have an upper bound to the number of payments per second that are possible to be routed over a Lightning Network since it depends on the capacity and speed of each node.
- **Trustless Operation** A Lightning Network utilizes Bitcoin transactions between nodes that operate as peers without trusting each other. Therefore, a Lightning Network preserves the principles of the bitcoin system [Ant14].

CHAPTER 5
Problem Description and Related Work

In this Chapter, we will analyze the problem description and the related work in more details.

The problem that this thesis aims to analyze is the recovery of the unspent bitcoins stuck inside the Lightning Network after a wallet failure (e.g. lost or corrupted transaction data put into the wallet storage).
A user with Bitcoin wallet may use the traditional cryptographic seed and the BIP32 address derivation, presented in Section 3.6.2, to recover the UTXO on the Blockchain. On the opposite, the Lightning Network does not use any distributed ledger, so the loss of all own data implies the loss of all own channel funds.

This problem has been faced in literature, and there are also different applications that try to solve it. In the next Sections, we will analyze these proposed solutions, and then we will reformulate our goals, based on what has been already done and what is needed.

The solutions that we will analyze in the next sections are:

- Eltoo
- Electrum Wallet
- Lightning Wallet based on Olympus server
- Piln

5.1 Related Work

5.1.1 Eltoo

Eltoo is a new protocol for payment channels and was proposed by Christian Decker, Rusty Russel and Olaoluwa Osuntokun in [DRO]. This solution offers some starting points in order to solve the proposed problem.

Eltoo is a very recent improvement on Poon-Dryja channels, which aimed to improve the efficiency of the network, by applying the following principles:

- The two sides of a channel shares the same commitment
- Using of a new *Opcode* (operation code) on the Blockchain

Eltoo does not only make the setup of payment channels more efficient but removes some of the complexities in maintaining a Lightning Network node as an end user [Ant].
The problem that Eltoo tries to solve is how to guarantee that an old state may not be broadcast to the Blockchain, once it has been replaced. In the Lightning Network, all updates happen off-chain, so it is necessary for Blockchain to have a way to evaluate all sides of the transactions before making a final confirmation.

In order to solve this issue, in Eltoo every state is composed of two transactions:

- An *update transaction*, that spends the previous contract's output to generate a new output
- A *settlement transaction*, that splits the funds based on the agreed upon distribution

Both the two types of transactions are shown in Figure **??**. In Eltoo, the new update transaction may be attached to a configurable timeout. In this way, the participants have to agree on an update before the timeout expires, otherwise, they will create a new update transaction, spending the previous output and double-spending the corresponding settlement, so effectively invalidating it.

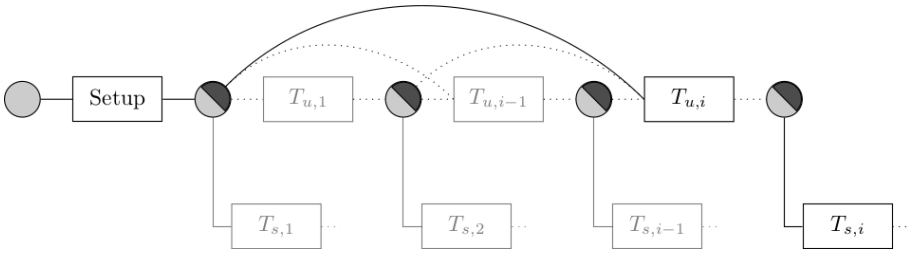

Figure 5.1: In Eltoo protocol the intermediate states can be skipped by rebinding a later update transaction to an earlier one, or directly to the setup transaction.[**decker2018Eltoo**].

The repeated invalidation of prior state to agree on a new state builds a long chain of update transactions, which terminates by the latest settlement transaction. To solve this issue, Eltoo permits to skip intermediate updates, basically connecting the final update transaction to the contract creation.

To this aim, the authors propose a new SIGHASH flag, SIGHASH_NOINPUT, which permits a transaction input to be bound to any transaction output with a matching script. In fact, all output scripts of prior update transaction outputs match later input scripts, in this way it is possible to bind a later update to any prior update, to skip any number of intermediate updates. Thus, to implement Eltoo, it is necessary to do a minor change to Bitcoin, adding the SIGHASH_NOINPUT flag for signatures.

Compare to the Lightning paper [PD16], that uses LN-penalty to punish a misbehaving party, Eltoo simply enforces the latest agreed-upon state of the off-chain contract. In this way, it is possible to eliminate all transactions belonging to outdated states. Therefore Eltoo reduces the amount of data stored by the nodes and the footprint on the network.

As the development of our solution, the main advantage the Eltoo brings is the shared status between the two parties. In this way, if one of the two sides loses all own status channel data, it may verify on the Blockchain the nodes with which it has an open channel, and then request to the counterparts the common status.
This scenario assumes that the counterpart is cooperative and help the unlucky node to recover own data. But, *what might it happen if the counterpart is malicious?*

Scenario: Let suppose that two users Alice and Bob, are using Eltoo protocol. Alice loses all her status channels. Eltoo is based on common share transaction between two nodes, so Alice simply asks Bob to recover own previous status and not risk to have an LN-penalty.

Problem: This might be a very useful solution, but, unfortunately, presents key vulnerabilities. The biggest one comes from the case that an adversarial node, in this case, Bob does not respond with the latest channel status to Alice, providing instead a previous one. The LN-Penalty fee, involuntary triggered by Alice, represents an economic incentive for Bob to act adversarially. In fact, if Alice broadcasts the wrong transaction (sent by Bob) to the Blockchain, she simply acts as a malicious node and Bob may obtain all the funds of channels.

5.1.2 Electrum Wallet

The problem to offer a mechanism of back up in Lightning Network was analyzed also by Electrum Wallet. Electrum is a software that permits to havea created wallet to be used in Bitcoin network, Figure 5.2. In 2017, it has begun to develop a wallet dedicated to Lightning Network, available for all users [Kon18]. The aim of this wallet was to solve three current problems connected to Lightning Network wallet:

- Backup of the entire Lightning Wallet (the problem addressed in this thesis)
- The Lightning nodes have to have the entire Blockchain, to be aware of each transaction happened on-chain in the Bitcoin network
- The Lightning Network wallet has to be online often, to monitor the open channels to avoid frauds

To solve the backup problem, Electrum Lightning Network wallet considered to send all data to a third party which offers an *on-cloud service*. All user data stored were encrypted. In this case, if the device is damaged or is lost, the user might recover all own data of the wallet including the open wallet, without the

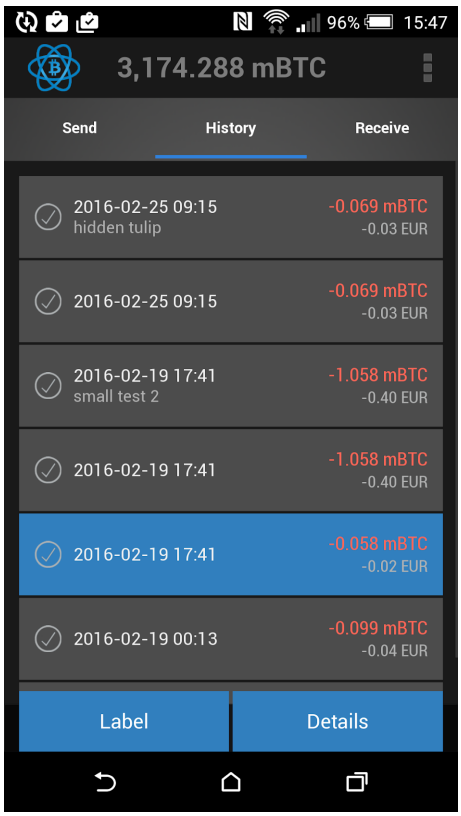

Figure 5.2: Electrum Bitcoin Wallet [Gmb].

service on-cloud known anything about the user information.

Problem: As presented in Electrum Lightning Network wallet, the backup problem may be solved using an on-cloud service. This solution causes a centralization of the network, against the decentralized nature of Bitcoin. Moreover, a number of security threats are associated with cloud data services:

- Traditional security threats, such as network eavesdropping, illegal invasion and denial of service attacks
- Specific cloud computing threats, such as side-channel attacks, virtualization vulnerabilities and abuse of cloud services.

Thus, this solution would not guarantee a secure service and also it would make the system centralized [Lui18].

5.1.3 Lightning Wallet: An Android phone application

The Lightning wallet is an android application, that permits to use Lightning Network wallet on own phone and relies on a server, called Olympus. The Lightning wallet does not depend on or ever shares any personally identifying information with Olympus and is designed to remain functional even when Olympus server goes offline.

One of the services that Olympus server permits to have is storing encrypted payment channel backups which may be used to reimburse locked channel balance if the user loses access to the own phone.

The application uses a mechanism, called *double backup strategy*: two copies of encrypted channel data are saved on both Olympus server and on user's Google Drive. The user has to use the own mnemonic phrase as an encryption key, in order to obtain and decrypt own channel backups.

This service is not free and a Lightning payment is automatically made by the Lightning wallet in exchange when the first payment channel is created. A requirement is that the peer, on the other side of the lost channel, has to be online and well-behaving to receive the recovery request, and initiate a remotely forced channel closure. In fact, both the peers have the option to forcefully close an opened payment channel, without the other's permission [Wal].

Problem : As the solution presented in Section 5.1.2, the Lightning wallet application foresee a centralization of Bitcoin, using external services as Olympus and Google Drive. On the other hand, it is interesting how this application uses the mnemonic seed, presented in Section 3.6.3, in order to access the service.

5.1.4 Piln

A further solution is proposed by Piln, a prepaid IPFS pinning service. The InterPlanetary File System (IPFS) is a protocol and network designed to create a content addressable, a peer-to-peer method of storing and sharing hashed hypermedia files in a distributed file system [Fin16].

Even if the service, offering by Piln, was available until January 8, 2019, this technology may offer some ideas for our solution. The use of Piln was anonymously and instantaneously, and it was created for the Bitcoin payments over the Lightning Network. Piln established to upload a file to IPFS through an API or a website like `https://globalupload.io/` and then visit `https://piln.xyz/` to pay for hosting. Therefore, the user might utilize this service to memorize all the status channel for a backup purpose. This service provided a service to store any data for any period of time, given the amount that the client pays. The list of pinned objects was public and everyone might extend the pinning time of any object.

5.2 Reformulate the problem

Analyzing the literature, it is possible to affirm that there is not a solution that solves completely the issue and respects the decentralized nature of Bitcoin technology.

I have seen an opportunity within the new mechanism Eltoo, which expects the two nodes of an open channel to share the same state (or commitments) for their unspent bitcoins. By leveraging Eltoo, one of the two nodes might try to recover missing data from the counterpart, since its unspent bitcoins are stored inside the same commitment. This might be a very useful solution, but, unfortunately, as we have seen in Section 5.1.1, it presents key vulnerabilities.

Moreover, our idea of leveraging Eltoo to mitigate the issue is still missing remediation comparable to the mentioned seed + BIP32 recovery which is provided today by any Bitcoin deterministic wallet.

5.3 Goals

Once analyzed the literature and the existing solutions to the addressed problem, we have defined the goals that our solution has to satisfy:

- It has to mantain the decentralization nature of Bitcoin, so not relying on on-cloud or another external service
- It has to guarantee the anonymity of each single user
- It has to guarantee the integrity and the confidentiality of data. Thus, it has to prevent unauthorized access (*confidentiality*) and modification (*integrity*).
- It has to be easy to implement and to rely on concepts and mechanisms already presented in Lightning Network
- It has to guarantee an immediate recovery service

CHAPTER 6
Methodology

In this Chapter, we will describe the methodology that has been followed during the development of this thesis. First, we will explain the methodology for the literature survey and secondly how we have reached our proposed solution.

6.1 Literature Survey

To reach the first goal, that is the comprehensive literature survey, we have proceeded in more steps.
First of all, we have analyzed through the Literature, the concepts of Self Sovereign identity and Protocols for Decentralized Networks, which gave us an introduction of the use of Blockchain in the companies and the decentralized system.
Then, we concentrated our focus on the following two main papers:

- *"Bitcoin: A Peer-to-Peer Electronic Cash System"* by Satoshi Nakamoto (October 31, 2008)
- *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments* by Joseph Poon Thaddeus Dryja (January 14, 2016)

which are crucial to analyze Lightning and Bitcoin in more details. During our analysis, *"Mastering Bitcoin"* by Andreas M. Antonopoulos [Ant14] has been our reference book.
Subsequently, we performed an online search using the following search engines for publications and articles:

- DTU electronic library
- IEEE Xplore
- Google Scholar
- Google
- Nakamoto Institute

Since Bitcoin and Lightning are quite new fields and their development is still on progress, the number of publications on this topic is considerably limited. For this reason, we have based our bibliography on the most recent articles and on the official specifics guide of Lightning Protocol.

6.2 Açai Protocol Design

Bitcoin is the result of the combination of three enormous fields: Game Theory, Economic and Cryptography. To completely understand Bitcoin and Lightning and to develop our solution, it was crucial to learn how to think in terms of game theory, in terms of economic and in terms of cryptography.

In our research, we have used mainly game theory and cryptography concepts, leaving aside the economic aspect not confronted in our work.

6.2.1 Game Theory

The game theory is a field of applied mathematics, that offers tools for studying situations in which parties, called players, make decisions that are interdependent [Mye13]. The key pioneers of game theory were mathematicians John von Neumann and John Nash in 1944, as well as economist Oskar Morgenstern [Inv18]. The aim of this science is analyzing the interdependence strategies to analyze the other player's possible decisions.

One of the most important concepts in game theory is so-called the *Nash Equilibrium*. The aim is giving a solution for a non-cooperative game involving two or more players in which each player is supposed to know the equilibrium strategies of other players, and no player has anything to gain by changing only their own strategy [OR94].

Nash equilibrium

We can define the Nash equilibrium, as follow:

> If each player has chosen a strategy, and no player can benefit by changing strategies while the other players keep theirs unchanged, then the current set of strategy choices and their corresponding payoffs constitutes a Nash equilibrium [Lev15].

To simplify, suppose to have two players, Alice and Bob, we may affirm that Alice and Bob are in Nash equilibrium if Alice is making the best decision she can, taking into account Bob's decision while Bob's decision remains unchanged, and Bob is making the best decision he may, taking into account Alice's decision while Alice's decision remains unchanged.
Therefore, a group of players are in Nash equilibrium if each one is making the best possibledecision, taking into account the decisions of the others in the game as long as the other parties' decisions remain unchanged.

The Nash equilibrium is used to analyze the outcome of the strategic interaction of several players, that means what will happen if several people are making decisions at the same time and if the outcome for each of them depends on the decisions of the others. John Nash's idea may show that one may not predict the result of the choices of multiple decision makers if one analyzes those decisions in isolation.

Game theory and cryptocurrency

Game theory in cryptocurrency has to do with incentives within the system: if you break a rule within the algorithm's system, there has to be a consequence that far outweighs the benefit. This is called *asymmetrical design* and aims to disincentivise bad actors to work against a system.

In this way, attacks inside Bitcoin such as double spend attack, made by a miner or the 51% attack, by multiple miners might be avoided on the Bitcoin network. If a minor attempts to do this, they will devalue their mining investment. In fact, they lose their mined Bitcoins and are not rewarded for completing transactions.

Therefore, miners are incentivised to be good actors on the network and this is the essence of game-theory science [Can18].

6.2.2 Cryptography

In Bitcoin, several techniques of cryptography are implemented, which make up the essence of this cryptocurrency.

Public key cryptography
When a user sends bitcoin to his/her counterpart, he/she creates a message (transaction), attaching the new owner's public key to the number of coins, and sign it with own private key. When the transaction is broadcast to the bitcoin network, each node may know that the new owner of these coins are the owner of the new key. The signature verifies for everyone that the message is authentic. In more details, the Private key is used for the encryption of transactions, while the Public key is used for the decryption of transaction [Blo18a].

Block-chaining
In order to preserve the integrity of the Blockchain, every block in the chain confirms the integrity of the previous one. Since the record insertion is costly because each block must have specific requirements, this makes it difficult to generate a valid block.

Hashcash
Hashcash is a cryptographic hash-based proof-of-work algorithm that requires a selectable amount of work to compute, it is the first secure efficiently verifiable cost-function or proof-of-work function.
The advantage of hashcash is that is non-interactive and there is no a central authority that manages secret keys.
In Bitcoin, the SHA256 is used as the underlying cryptographic hash function. It takes as input data and transforms it, in an effectively-impossible to reverse or to predict way, into a relatively compact string. In this way, nobody may obtain the same hash value if the inputs are different. Bitcoin blocks may be identified by their hash, which serves the dual purpose of identification as well as integrity verification.

6.2.3 Lightning Network Community

During this project, I could receive feedbacks and suggestions, writing to the mailing list of the Lightning Network Community (`https://lists.linuxfoundation.org/mailman/listinfo/lightning-dev`). In this way, I could ensure that the project accurately reflects the requirements and needs of the community.
This work is the result of a constant correspondence, that permits me to increase my knowledge of the technology, keep up with the newest upgrades and improve continuously my solution. I had also the possibility to participate to the *Lightning Hackday in New York* on 27^{th}- 28^{th} October 2018, where I could talk with the most expert people of the field about my thesis and it was crucial for receiving feedbacks on my research.

CHAPTER 7
The solution: Açai Protocol

This chapter aims to present the Açai Protocol, our proposed solution to guarantee a backup service in *Lightning Network* (Chapter 4).

The Chapter is organized to show all the steps that conducted us to finalize the solution. In the next sections, we will present three solutions that we have developed.
The first solution considers connected nodes as lockbox of backup, whereas the second solution uses the *Watchtowers* to guarantee a recovery mechanism. We will explore these two solutions through different use cases, explaining how they work, their advantages and what makes impracticable their implementations. In order to solve these drawbacks, we will develop a third and final solution, called Açai protocol, that we may consider as an improvement of the two first proposals. We will analyze the main aspects and demonstrate why this last solution is able to solve the problem presented in Chapter 5.

In the next sections, we will consider the network represented in Figure 7.1, where the node of Alice is connected by payment channels to Bob, Charlie and Eve nodes.

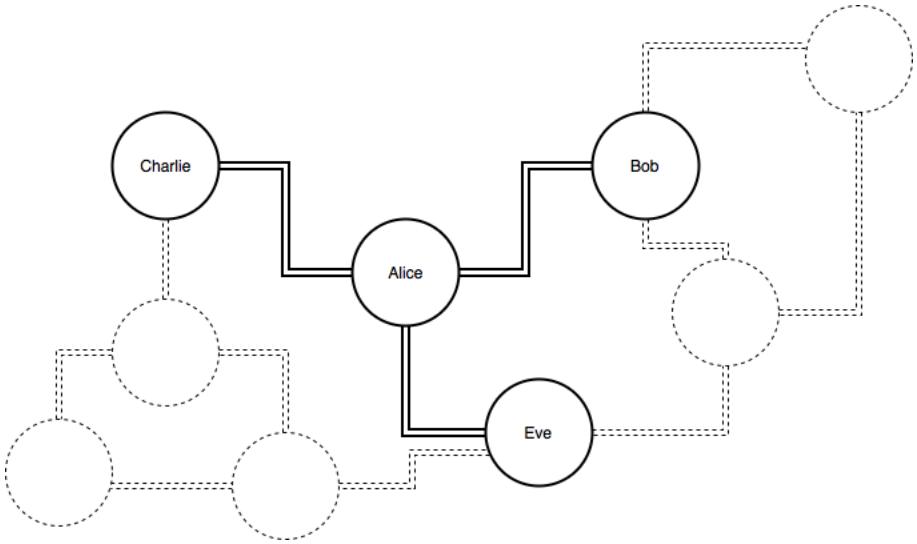

Figure 7.1: Network Features.

7.1 What is missing in Eltoo

Let consider Eltoo, the protocol presented in Section 5.1.1. As we have seen, Eltoo allows sharing the same information between the two counterparts of a payment channel. In this section, we aim to demonstrate why Eltoo is not sufficient to recover the lost wallets. Suppose to have a payment channel between Alice and Bob and the software running Alice's wallet fails, losing all her status channel information. The following detailed use case documents this possible scenario.

Use Case: Backup through Eltoo protocol

Name: Backup through Eltoo protocol

Summary: Use Eltoo protocol to recover the channel status in case of wallet failure

Actors: Alice

Preconditions:

- Alice has an open payment channel with Bob, using Eltoo protocol.
- Bob must be online and available to send the latest channel status to Alice.
- Alice has lost the most recent status channels, due to a wallet failure.

Main Scenario:

1. Alice asks to one of the counterparts her last status channel.
2. The counterpart receives the request and is incapable to provide the requested data.
3. The counterpart sends to Alice the data about their status channel.
4. Alice recovers the status channel thank to the selected counterpart.

Alternative Scenarios:

1. The counterpart sends to Alice an obsolete status which was previously valid, to trigger the protocol penalty (Section 4.5) and steal all the funds in the channel.
    - Alice has no way to verify if data provided from the counterpart is correct since the channel data is not written anywhere else.

As it is demonstrated in the Alternative Scenario and also in 5.1.1, Eltoo does not solve the case in which Bob is not cooperative and sends a wrong status to Alice. If this happens, Alice might trigger the protocol penalty without knowing it.

Thus, Eltoo alone may not solve the problem completely, even if the possibility to have a shared common state between the two nodes might be useful in order to have the same status for both the nodes.

To mitigate this problem, we have developed three possible solutions.

7.2 Nodes as lockbox of Backup

The first idea that we have developed was to use the other connected nodes as a lockbox of back-up of own recent status.

Let consider that a node, called Alice is connected through payment channels to nodes Bob, Charlie and Eve (Figure 7.1).
We may formalize the idea as follows:

> *If a node Alice is connected to Bob, Charlie and Eve, for each transaction between Alice and Bob, Alice sends encrypted information, regarding the new backup of all her status channel, to Charlie and Eve. For each commitment transaction with Charlie, Alice sends encrypted information, backup of all her status channel, to Bob and Eve. Finally, for each commitment transaction with Eve, Alice sends encrypted information, regarding the last backup of all her status channel, to Bob and Charlie.*

Thus, Alice uses the connected nodes as a lockbox for backup. Moreover, the data are encrypted with the public key of Alice, so just Alice is able to decrypt the information, Figure 7.2. It is important to note that, as it is possible to observe in the Figure below, Alice sends to Charlie her status with Bob and Eve, and not the status with him. This is further security protection since Charlie has not the possibility to send to Alice a previous backup, different from the last one, more favourable for him. In this way, if Alice sends her back up to Charlie, this one memorizes her data, without reading or modifying it.
Therefore, for each transaction, a new backup is created, encrypted with the private key of the owner and sent to the connected nodes.
In this way, if Alice loses her wallets, she may ask the information to the other connected nodes and update her status. To guarantee the service, for each request of information Alice has to pay a small fee to her connected nodes. This solution reduces the probability of successful attacks by Eve because other nodes will provide additional proof on the channel status between Alice and Eve.

7.2.1 Privacy and costs

This solution has two main disadvantages:

- Privacy
- Expensive costs

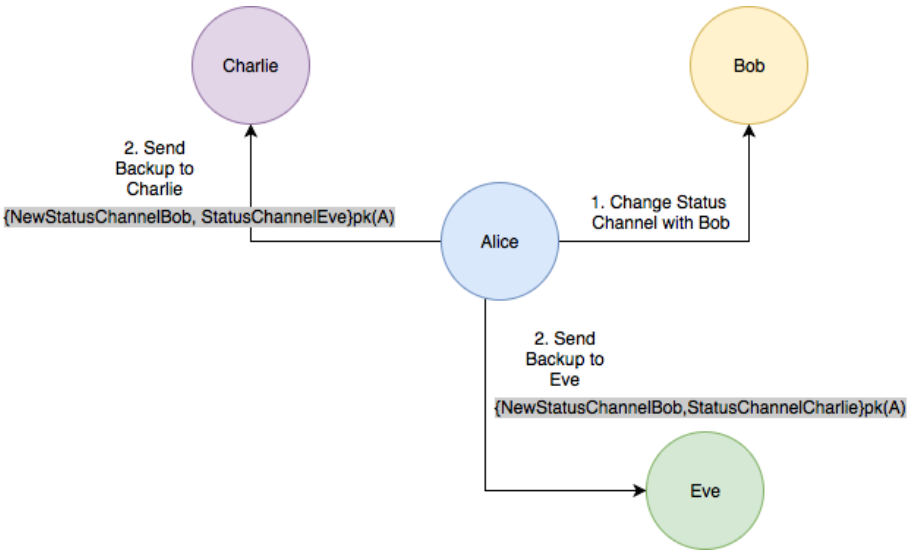

Figure 7.2: Nodes as lockbox of Backup. Alice and Bob change their status channel, Alice sends to Charlie and Eve the new backup.

Privacy
First, this solution may be considered against privacy. In fact, the other nodes may know when the "unlucky" node has lost all status channels data and use this information at their advantage, providing a previous version of the encrypted backup. To solve this lack of privacy, Alice might ask regularly to the other nodes a backup of her status, for example, every time she connects to Lightning Network. In this way, she may put to test her counterparts and verify that the data are correct. In the negative case, she may identify her counterpart as not trusted and closes the channel with him.

Expensive Costs
As regarding the second disadvantage, to decrease the costs, it is not necessary to ask information to all nodes every time. For example, consider the network in Figure 7.2.

- Each transaction with Bob, Alice sends the backup to Charlie and Eve.
- Each transaction with Charlie, Alice sends the backup to Bob and Eve
- Each transaction with Eve, Alice sends the backup to Bob and Charlie

Therefore, if Alice needs to recover the status with Charlie, she may simply ask to Bob or Eve, and not to both of them. For instance, she might ask Bob the status with Charlie, to Charlie the status with Eve, to Eve the status with Bob, changing every time this order. On the other hand, it is necessary to send information every time to all nodes, so if one of the channels will be closed, there is no risk to lose data.

7.2.2 Assumptions

To develop this solution, we have made the following assumptions:

- A node has to be connected to at least two nodes. For example, Alice has to be connected to Bob and Charlie: this allows that for each transaction with Charlie she may send the backup to Bob, and vice-versa.

- We presume that Alice maintains mnemonic seed (Section 3.6.3) in a safe place, in order to never lose it. Only in this way, Alice is able to connect to the Blockchain, to know her connected nodes and to ask to them the backup.

7.2.3 Use cases

In this Section, we will present a possible scenario, demonstrating which problems this solution might present.

Use Case: Backup through connected nodes

Name: Backup through connected nodes

Summary: Recovery the channel status with Bob in case of wallet failure through connected node Charlie.

Actors: Alice

Preconditions:

- Alice has a payment channel, with Bob, Charlie and Eve.
- The counterpart has to be online and receive the request of Alice.
- Alice has lost the most recent status channels, due to a wallet failure.

Main Scenario:

1. Alice asks Charlie her last status channel.
2. The counterpart receives the request.
3. The counterpart sends to Alice the data about their status channel.
4. Alice recovers the status channel with the selected counterpart.

Alternative Scenarios:

1. The counterpart sends to Alice, not the most recent status, but instead it sends a previously shared status more favourable for him/her.
    - Alice believes that the status sent by her counterpart is correct, even if it is not favourable for her.

7.2.4 Why we discarded this solution

As it is shown in the use case, a possible problem of this solution is that the node, in this case *Alice*, has to be connected to at least two nodes and, moreover, the connected nodes might be offline, so they might not guarantee an immediate backup service. To solve this problem, we have decided to maintain the same design, but instead of using normally connected nodes, we move our focus on full node always online. For this reason, we have seen a possibility on the mechanism of Watchtowers.

7.3 Introducing Watchtowers as a mechanism of backup

Hence, to solve the problem of the previous solution, we have decided to consider Watchtowers, full nodes that provide the storage of encrypted Eltoo channel state, without being able to access them. In Lightning Network, Watchtowers are designed to broadcast, on the Blockchain, the latest channel status if an adversarial node tries to spend an old commitment, exploiting the offline state of its counterpart.

As Tadge Dryja said in [Dry], the price of scalability is having an eternal vigilance inside the network, so this solution provides a form of vigilance inside the Lightning Network when a node is not online. This concept brings to create the idea of the Watchtower, as a service which is always online and watching for channel breaches even at times when your wallet is offline. Further exploring upcoming Lightning Network features, the Watchtowers, as have been outlined so far, will provide a protection service to offline nodes, by storing their encrypted commitments.

In this solution, we will adopt two new concepts of Lightning Network:

- Eltoo
- Watchtowers

We will leverage the availability of Watchtowers to store further data, that will be used to verify the integrity of Eve and other counterparts. In our solution, we will add another function to the Watchtowers, that is a mechanism of backup. To simplify, let consider a Watchtower simply as a full node that is constantly online, not considering the mechanism of monitoring status channel, in order to focus just on the solution of our problem.

7.3.1 Design

In this solution, we introduce two new concepts using to send and request the backup:

- nonce-time Tn, as the current value of nonce-time (sequential integer number that defines the order of the backups)
- payload P, {zip(T1), T1, Alice} pk(Alice), that consists of

1. a zip of all status channels of Alice at a specific time T1
2. the nonce-time correspond to the time T1 of status contained in the zip
3. the owner of the backup, e.g. Alice

This payload is encrypted by the public key of Alice, so the Watchtowers are not able to know the status channel of Alice.

To explain the design, let consider the Figure 7.1, where Alice has Eltoo channels with Bob, Charlie, Eve, and three Watchtowers W0, W1 and W2.

The idea is not sent all data to all Watchtowers, but instead just send the actual nonce-time and the actual payload to one of the Watchtower, and send just the new nonce-time to the others. In this way, we may split the data into the three Watchtowers.

How to send data to the Watchtowers

In this section we will list all the steps that Alice does every time that she changes a status channel with one of her connected nodes (Figure 7.1):

1. Let suppose that Alice and Bob change their status channel.
2. Alice sends the new status to the Watchtowers W0 and shares the current nonce-time with W1 and W2.
3. When Alice sends her information to the three Watchtowers, these memorize the node, current nonce-time, payload:

```
W0:  Alice    T0     P0
W1:  Alice    T0     -
W2:  Alice    T0     -
```

4. Alice and Charlie change the status of their channel.
5. So, Alice sends the new status to W1 and sends the new nonce-time to the others, which substitute the previous current nonce-time in the information of Alice:

```
W0:  Alice    T1     P0
W1:  Alice    T1     P1
W2:  Alice    T1     -
```

6. Alice and Eve change the status of their channel.

7. Alice sends the new status to W2 and sends the new nonce-time to the others:

```
W0:  Alice    T2     P0
W1:  Alice    T2     P1
W2:  Alice    T2     P2
```

8. Alice and Charlie change again the status of the channel.

9. Alice sends the new status to W1 and upgrades the nonce in the others:

```
W0:  Alice    T3     P3
W1:  Alice    T3     P1
W2:  Alice    T3     P2
```

We assume that each Watchtower may be used by more than one nodes, as result we might represent the data stored by watchtower, e.g. W0, as in Figure 7.3.

Node	Nonce	Payload
Alice	T5	P4={zip(T4), T4, Alice} pk(Alice)
Eve	T6	P3={zip(T3), T3, Eve} pk(Eve)
Bob	T3	P3= {zip(T3), T3, Bob} pk(Bob)
Charlie	T4	P1= {zip(T1), T1, Charlie} pk(Charlie)
Diana	T2	P2= {zip(T2), T2, Diana} pk(Diana)

Figure 7.3: Data stored by Watchtower W0.

How to request back up to the Watchtowers

In this Section, we will list all the steps that Alice has to do every time that she needs to request back up to the connected Watchtowers.

1. When Alice needs to have the backup of all her data, she has to ask all her connected Watchtowers the information related to her node. For example, taking the last example above:

```
W0 sends  {Alice, T3, P3} to Alice
W1 sends  {Alice, T3, P1} to Alice
W2 sends  {Alice, T3, P2} to Alice
```

2. Alice analyzes the information sent by W2. She notices that the payload contains the nonce T2, whereas the new nonce-time is T3.

3. Then, she notices that the payload sent by W1 corresponds to T1, but the nonce is T3.

4. When Alice analyzes the information sent by W0, she analyzes that the Payload corresponds to T3, that is also the last nonce-time.

5. Alice knows her last status.

7.3.2 Security

Since all the Watchtowers stores the current time-nonce and the payload is encrypted with the public key of Alice, we may avoid the following situations:

- A Watchtower sends an older payload instead of the last one. The payload has to contain the current time-nonce to be considered the last one, each payload with a nonce time older are not considered.

- If the Watchtower W0 with the last payload P3, sends a previous backup P0 and the current nonce time is T3, it is possible to discover the malicious activity. If W1 has the payload P1, the W2 has the payload P2 and the current nonce-time is T3, we could conclude that W0 contains P3 and is cheating.

- If one of the Watchtowers, W3 decides to change the nonce-time, for example from T3 to T5 and send to Alice T5 with the payload. Alice can think that the actual time-nonce is T5, but no one of the Watchtowers contains a payload corresponding to the state T5. This sort of misbehaviour is solved from the majority, e.g. if 2/3 nodes confirm that the actual nonce-time is T3, Alice considers this last nonce-time as the last one and not T5. The attack may happen just if the 51% of the Watchtower agree to cheat and send to Alice another nonce-time.

7.3.3 Fee

Every time that the node Alice requests data to the Watchtower for the backup, she sends it a small fee through the Lightning channel. This micropayment encourages the Watchtowers to guarantee the service every time, as the nodes in the solution presented in Section 7.2.

7.3.4 Why we discarded this solution

Despite the advantages, this solution has a crucial drawback. In fact, since we use a public key to encrypt the data, another node can personify Alice and send a forged payload to a Watchtower(this problem was analyzed in Section 3.6). Hence, all an attacker has to do is to corrupt the backup data, and modify it with data that is favourable for him.
With corrupted backup data, the operation of Alice is doomed and irrecoverable, especially if private keys or even just derivation paths are part of the backed-up data.

In order to solve this disadvantage, we will apply the use of Watchtowers and use them for a backup mechanism, without the use of a public key.

7.4 Açai Protocol

This solution, named Açai Protocol after the controversial berry fruit, as the solution analyzed in Section 7.3, aims to use the Watchtowers not just for monitoring the channels, but also as a backup service. It applies some minor modifications to the standard protocol, therefore more efficient than the previous solution.
From the problems presented in Section 7.3.4, we have decided to improve the solution and eliminate the use of a public key that may identify the node using the service. We have preferred trying to use the same mechanism, that the Watchtowers utilize for the monitoring of the channel status, to guarantee a service of backup.

For this reason, in the following Section, we will analyze in more details the mechanism of Watchtowers and then subsequently the design of the Açai Protocol.

7.4.1 The Watchtower blob recovery method

To analyze the mechanism of Watchtowers, suppose to have a circumstance where Alice has open channels with Bob, Charlie, Eve (Figure 7.1) and is using 3 Watchtowers: W0, W1 and W2.

Eve, exploiting the situation of Alice being offline or unable to broadcast her *commitments* might try to broadcast an older channel state to the Blockchain, essentially stealing the last payment done to Alice. In this event, a Watchtower is capable to identify this malicious transaction from the mempool (Section 3.3), and intervene in defence of Alice when is offline, sending the correct and most recent channel state that invalidates Eve's attack.

Alice, in order to keep the Watchtowers aware of the latest channel status, has to send a payload made by *hint* and *blob* after every update to her commitments with Bob, Charlie or Eve. The hint and the blob are stored by the Watchtower, and they are utilized for monitoring the channels.

For example, the sequence of events after a Lightning payment between Alice and Bob is:

- Change of the channel status between Alice and Bob (a new Eltoo commitment is signed).
- Alice calculates the variable 'hint' by truncating the commitment hash (which is also the txid): hint = tixd[:16]
- Alice calculates the payload 'blob' by encrypting the commitment itself, using the hash of txid as the symmetric key: blob = Enc(data, txid[16:])
- Alice sends (hint, blob) to one of the Watchtowers W0, W1 or W2.

Therefore, some of the Watchtowers are storing the information that Alice needs in case of failure, without the adversarial risks from asking this data to Bob, Charlie or (worse) Eve as proposed in Section 7.2.

7.4.2 Açai Protocol design

In this Section, we will present the Açai Protocol. This solution aims to use the Watchtowers not just for monitoring the channels, but also as a backup service, without letting the Watchtower itself knowing that the stored Blob is effectively the Açai payload or a normal commitment.

This protocol adopts the same concepts of txid, hint and blob. In order to distinguish the two types of data payload, we will use a different notation: txidç, hintç and blobç.

Therefore:

- txid, hint and blob represent the standard format used by Alice to interact with the Watchtowers and retrieve the normal blobs for verification
- txidç, hintç and blobç represent the format used by Alice's wallet to store the data of all the txid that she is using inside the Lightning Network. This data will be the backup in case of failure.

Technically, Açai Protocol leverages the same endpoints:

- hintç = txidç[:16]
- blobç = Enc(dataç, txidç[16:])

In the Açai protocol, dataç contains information of all txid: [txid_Bob, txid_Charlie, txid_Eve], where txid_Bob is the hash of the Eltoo commitment between Alice and Bob.
Therefore, once Alice retrieved and decoded her Açai blob, she is able to recover the list of the regular blobs previously stored inside the Watchtowers. Every time that a channel changes its state, Alice sends to one of the three Watchtowers W0, W1 or W2 (randomly chosen) two different payloads: the channel state information (hint, blob), and the Açai blob (hintç, blobç) containing the updated list of txid. Note that the Watchtowers will store both the standard blobs (hint, blob) and the Açai blob (hintç, blobç), without being able

to distinguish them. We can imply that an Açai's blobç has length and size comparable to standard Watchtowers blobs. Therefore, Açai Protocol stores the back up inside a properly crafted blob, using hints as unique data pointers and the Watchtowers for the backup itself.

The problem is not completely solved: we need a technique for the wallet to recover the txidç, and so the hintç correspond to the last backup, to be used to receive the blobç containing dataç.
Taking inspiration from *Lightning Wallet: An Android phone application* (Section 5.1.3), we have decided to use the BIP39 mnemonic key (Section 3.6.3) that allow restoring the Bitcoin wallet. We define txidç as the obtained public key through BIP32 function, calculating with the node seed and the following derivation path (Section 3.6.2):

Derivation Path= m'/108'/0(mainnet)/(account number)'/0/Current_Blockheight

Where 108 is an arbitrary purpose number that we have chosen to indicate the Açai Protocol, and the account number is needed to match the wallet subject to restore the process.

In the next two Sections, we will analyze all the steps how Alice sends data to the Watchtowers and how she requests her back up.

7.4.3 Scenario 1: Send the backup to the Watchtower

Scenario: After payment to Bob, the channel state changed, so Alice has to send the new channel information (hint, blob) and the new backup (hintç, blobç) to Watchtower W0 (randomly chosen).

Steps:

1. To generate txidç, which represent the key to identify and decrypt blobç, Alice uses its deterministic wallet address function, applying the current Block height to generate the derivation Path (Section 3.6.2).

   Derivation Path= m'/108'/0(mainnet)/(account number)'/0/Current_Blockheight

2. Calculate the txidç_n.

   Since it is possible for Alice to perform n payments in the same Blockheight, we must enumerate each new state of the Açai backup inside the same Block as: txidç_0, txidç_1, txidç_2, txidç_3...txidç_n

   Therefore, we may assume that we need to change the txidç while keeping the same Blockheight in the BIP32 derivation function, thus we have decided to use the following technique:

   txidç_0= 2SHA256(pub-key)

   As soon as Alice needs to update her Açai backup, but the Blockheight is not yet changed, we can apply the following rule: the txidç_n is calculated

as the hash function of the previous txidç_n-1.

For example:

```
txidç_1= SHA256(txidç_0)
txidç_2= SHA256(txidç_1)
txidç_3= SHA256(txidç_2)
.....
```

3. Truncate the txidç_n into hintç_n= txidç_n[:16] and encrypt blobç_n= Enc(dataç_n,txidç_n[16:])

4. Send hintç_n and the blobç_n to Watchtower W0, with the information of the new channel status (hint,blob).

The Figure 7.4 shows the activity diagram for this scenario. Therefore, after a change of a channel status, Alice has to calculate a public key, through the Derivation Path and her seed. Then she calculates the txidç, through the function SHA256 on the public key. Through the found value, she calculates the hintç and blobç. Blobç cointaind the data regarding the channels status of Alice, encrypted with txidç[16:] as Secret Key, and she sends the values to one of the Watchtowers, that stores them.

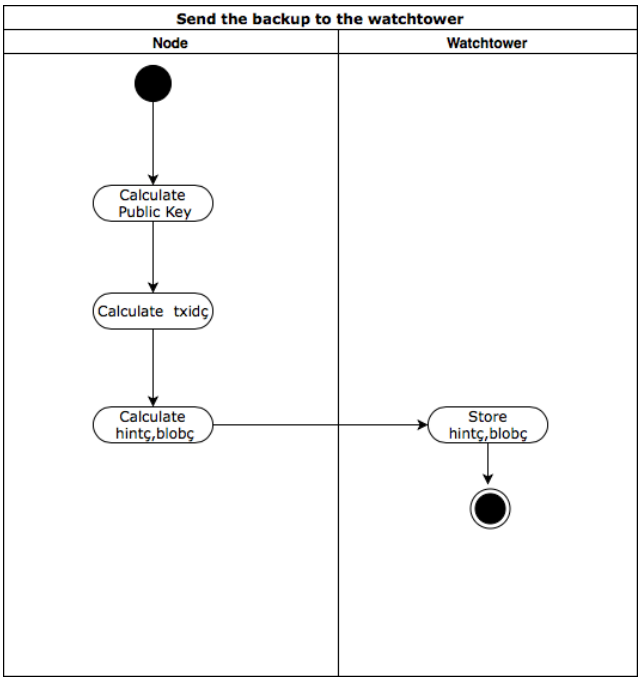

Figure 7.4: Activity Diagram of Send Data to the Watchtower.

7.4.4 Scenario 2: How to request backup to the Watchtowers

Scenario: Alice may request the backup when she has lost all her data accidentally or otherwise she requests the backup to the Watchtower every time she connects to her Lightning node. In this way, she is able to check that the Watchtowers are storing real data and they are providing the promised service.

Steps:

1. Alice asks to the connected nodes the current Blockheight. Note that the nodes cannot cheat, otherwise Alice might decide to close the channel with them, since they are not trusted.

2. Use the Blockheight and the seed to calculate the deterministic pub-key:

   Derivation Path= m'/108'/0(mainnet)/(account number)'/0/Current_Blockheight

3. Calculate txidç_0 = 2SHA256(pub-key) and the hintç_0=txidç_0[:16]

4. Ask to W0, W1 and W2 Watchtowers to retrieve hintç_0.

   - **case a:** If one of the Watchtowers contains the hintç_0, it could also contain hintç_1 (case of more than one transaction in the same Block). Therefore, Alice's wallet calculates txidç_1=SHA256(txidç_0) and asks the Watchtowers if one of them contains the hintç_1. If one has the hintç_1, calculate txidç_2 and hintç_2 and so on. If no one has hintç_2, we can assume that txidç_1 contain the latest channel state so Alice can decrypt blolbç_1 and extract the list of txid.
   - **case b:** If no one of the Watchtowers provides hintç_0, we can assume that Alice doesn't have any transactions at the current Block height. In this case, Alice's wallet generates a new pub-key, decreasing the block height by one: pub-key = m/108'/0(mainnet)'/(account number)'/0/(Current_Blockheight-1). Once generated, Alice's wallet calculates hintç_0 and proceed as in the case a.

5. When Alice's wallet finds her hintç_n, she requests to the Watchtower to send the correspond blobç_n. The blobç is composed by dataç and txidç[:16], where dataç=[txid_Bob, txid_Charlie, txid_Eve]

6. From dataç, Alice may recover the txid_Blob. then she may calculate the hint_Bob=txid_Blob[:16] and ask the Watchtowers which one has that value and requests the corresponding blob_Bob. Since Alice knows txid_Blob[16:], she is able to get the data inside blob_Bob and recover the status channel with Bob. The same happens with the recovery of the status channel with Charlie and Eve.

7. At this point, Alice is able to check that her data match the ones stored in the Watchtowers, or otherwise recovery her data.

The Figure 7.5, shows the activity diagram for this second scenario.

7.4.5 Assumptions

To achieve this solution, we assumed :

- Fees: Watchtowers model still needs a form of trustless payment to be economically viable, e.g. to cover the extra storage and bandwidth for the service (both for the normal service and the Açai backup)
- Memory: The Açai data stored in the watchtowers are not deleted or substituted/tampered.
- Açai Protocol assumes that Alice knows the Watchtowers she used, in order to recover her status.

7.4.6 Formalization of the definition of Watchtower

In Literature, there is not an official formalization of Watchtower, since it is a quite new concept. For this reason, we wish to give a definition of the Watchtower, which includes the mechanism of backup.

We define Watchtower as a full-node, always online, that can watch for channel breaches even at times when your wallet is offline, by leveraging Açai Protocol provide a backup service if your Lightning-enabled Bitcoin Wallet needs to be recovered.

7.5 Comparison among the three solutions

In this Section, we will compare the tree solutions analyzed in 7.2, 7.3 and 7.4, based on the goals specified in Section 5.3.

As it is possible to observe in Figure 7.6, all of the three solutions respect the decentralized nature of Bitcoin, in fact, they do not use any on-cloud or any other external service for back up.
As regarding the anonymity, only the Açai Procol guarantees this feature, whereas the first two solutions are based on the use of the public key, and as it is explained in Section 7.3.4, it might identify the node owner of backup.
All of the three solutions maintain the integrity and confidentiality of the stored data. In fact, in the first two solutions, since the backup and the payload are encrypted with the public key of Alice, no one may modify it or read it. For the Açai protocol, the txidç is the result of SHA256 hash function of the public key and dataç is encoding with the value of txid[16:]. Since the hash function is a one-way operation, it is impossible to compute the input from a particular output, so the possibility for an attacker to modify dataç is quite minimum (the unique way to obtain dataç is through a *rute force attack*).
For the implementation, the first two solutions are not simple to implement: the first one requests a modification on Lightning Network channels to support backup mechanism, and the second requests a new structure for the Watchtower to implement the mechanism of Backup. For the Açai Protocol, the implementation is not difficult because the mechanism utilizes the same structure that the Watchtower use for monitoring the channels.
The use of Watchtowers compared to normal Lightning nodes of the first solution

may offer an immediate recovery service, that we obtained in the second and in the third solution.
In conclusion, Açai Protocol is the unique solution that is able to satisfy all the settled goals proposed in Section 5.3.

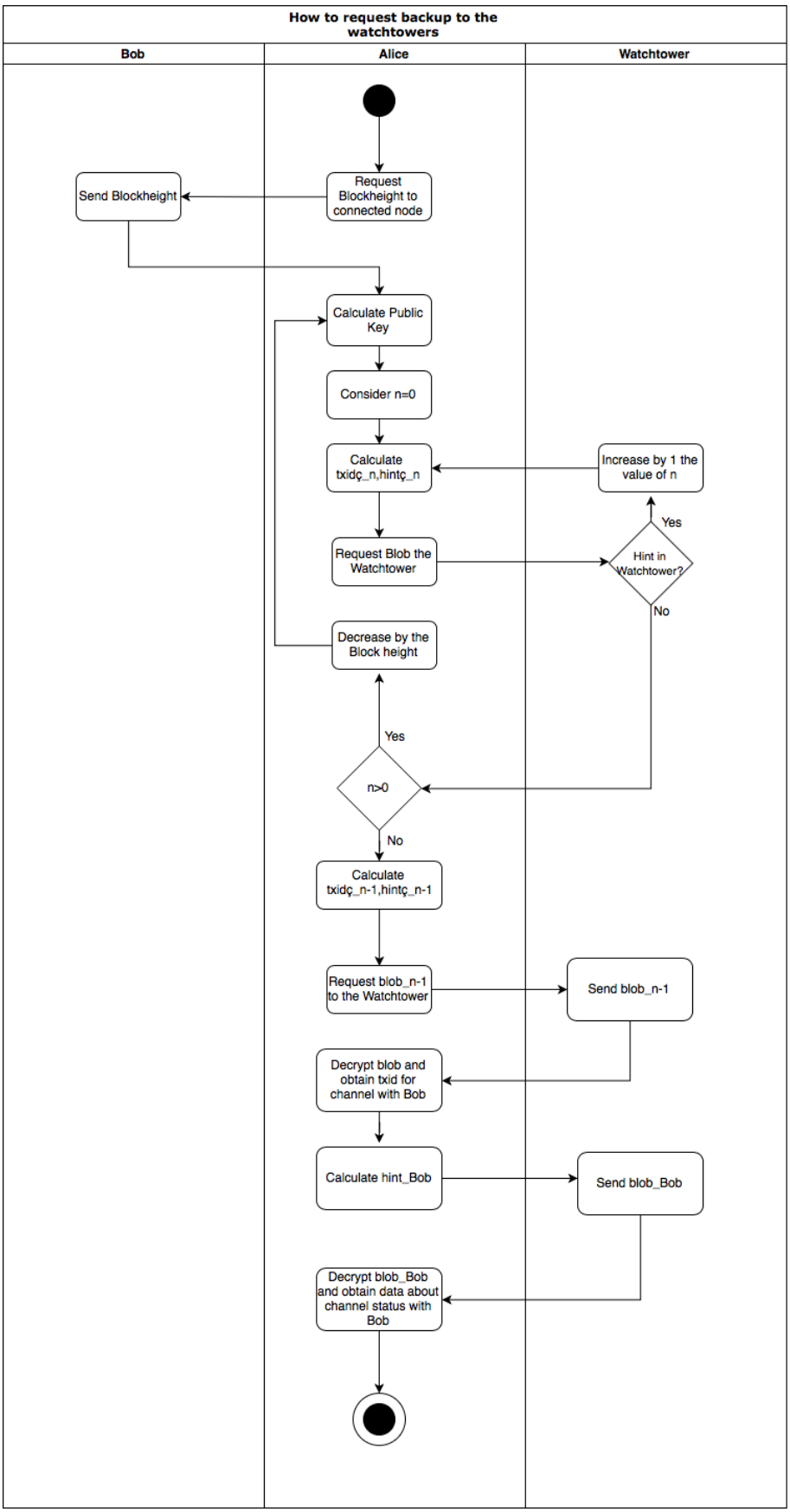

Figure 7.5: Activity Diagram that shows how Alice can request Data to the Watchtower.

	Nodes as lockbox of Backup	Watchtowers as a mechanism of backup	Açai Protocol
Decentralization	✓	✓	✓
Anonymity	✗	✗	✓
Integrity and Confidentiality	✓	✓	✓
Simple to implement	✗	✗	✓
Immediate Recovery Service	✗	✓	✓

Figure 7.6: Comparison among the three solution analyzed in Sections 7.2, 7.3 and 7.4.

CHAPTER 8
Implementation

In this Chapter, we will report a possible implementation for the Açai Protocol, presented in Chapter 7.4.

Since neither Eltoo nor Watchtowers are completely implemented in Lightning Network[1], we will create a Java RMI project, to simulate the communication between a Lightning node, called Alice, and a Watchtower[2].

The JAVA RMI project is a technology which permits to distributed system to communicate through a network. It is generally compromised of two programs: a server and a client. The former creates the remote objects and makes references to these objects accessible, and successively waits for the client to invoke methods on these objects. The latter obtain a remote reference to remote objects and invokes methods on them. In our implementation, the Watchtower is the server, whereas Alice is the client.

In this Chapter, we will focus on how to implement the Açai Protocol, more specifically: how to create txidç, hintç, blobç, how to send data to the watchtowers, how to request the backup, and we will demonstrate how utilizing this information,it is possible to recover the lost status channels.

8.1 Design

Figure 8.1 shows the entire mechanism that we will consider in this project, where the node Alice uses the Watchtower to memorize own data.

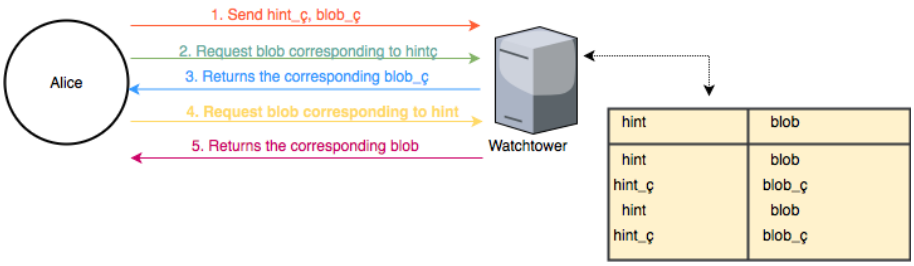

Figure 8.1: Entire system considered in the project..

[1]The development is proceeding in this period

[2]To simplify the RMI project, we will consider a unique watchtower that stores back up data for Alice

To solve the problem of hint and blob storage we will utilize a file, where all data are memorized. Moreover, the solution guarantees the integrity and the confidentiality, since just Alice is able to decrypt her blob/blobç, through her seed and the Derivation path (Section 7.4). The communication between the node and the watchtower is through Lightning Network. For this reason, in the implementation, we consider that the communication between the Watchtower and Lightning node is secure, and the other nodes are not able to steal the information.

The next part of this Section gives an introduction to RMI Architecture and presents the java classes implemented in our project.

8.1.1 Architecture of an RMI Application

The RMI (Remote Method Invocation) is used to build distributed applications, providing remote communication between Java programs, through the package java.rmi.
The RMI is generally compromised of two programs: a server and a client. The server program creates a remote object and makes available a reference of that object for the client, through the use of the registry. The client program requests the remote objects on the server and invokes its methods. The diagram in Figure 8.2 shows the architecture of an RMI application.

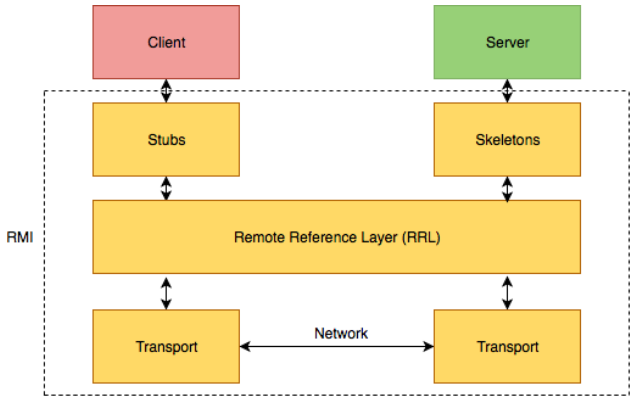

Figure 8.2: Architecture of an RMI Application..

In the Figure, the *Transport layer* connects the client and the server, managing both the existing connection and also the new ones. The *Stub* is a representation (proxy) of the remote object at client program. The *Skeleton* is the object which resides on the server side. The *RRL(Remote Reference Layer)* is the layer which manages the references made by the client to the remote object.

When the client invokes the remote object, it is received by the stub which passes this request to the RRL. When the RRL of the client program receives the request, it invokes a method called *invoke()* of the remote object and it passes the request to the RRL on the server side, through the Transport layer.

Then, the RRL on the server side passes the request to the Skeleton which finally invokes the required object on the server. The result is passed back to the client.

RMI Registry

The RMI registry is a named repository of all server objects, registered using a unique name known as bind name. Every time the server creates an object, it registers this object with the RMI registry (using *bind()* or *reBind()* methods). To invoke a remote object, the client needs first a reference of that object and then it may fetch the object from the registry using its bind name (*lookup()* method) [Tut].
The Figure 8.3 explains the entire process.

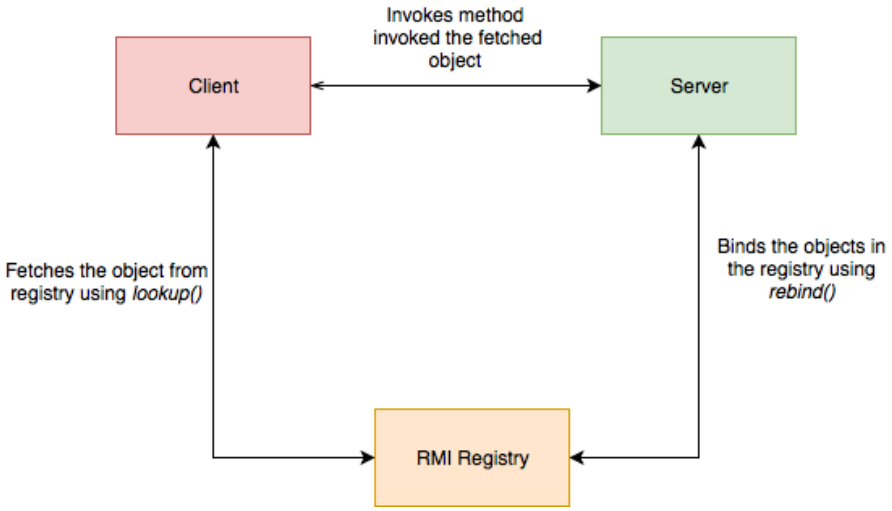

Figure 8.3: RMI Entire process.

We have decided to use JAVA RMI to implement our solution for the following considerations:

- It minimizes the complexity of the application.
- It provides more advanced packages for security.
- It minimizes the difference between working with local and remote objects.

8.1.2 Classes

To write an RMI Java application we need the following classes:

1. The remote interface, that provides the description of all the methods of the remote object.

2. The implementation class (remote object), that implements the remote interface

3. The server program, that extends the implementation class.

4. The client program, that fetches the remote object and invokes the remote method

Figure 8.4 illustrates all the parts of our RMI application.

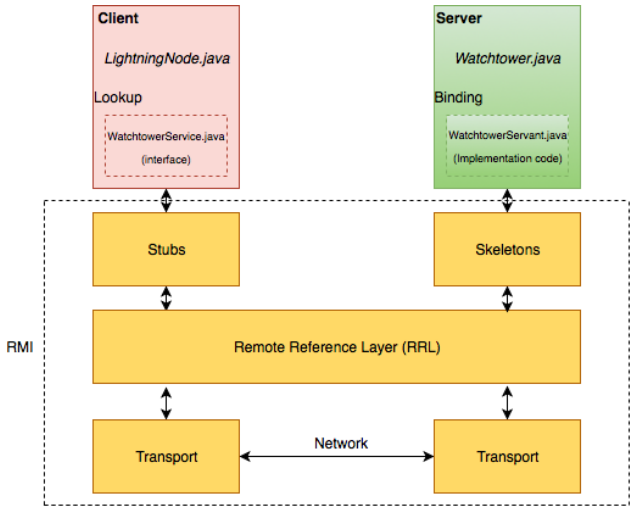

Figure 8.4: Parts of our RMI Application..

In our implementation we have a remote interface, called WatchtowerService.java, a server called Watchtower.java, an implementation class, WatchtowerServant.java and the client LightningNode.java.
First of all, when Alice (client) wants to communicate with the Watchtower (server), she has to create a remote connection with the WatchtowerService. Then, she is able to send the hint, blob, hintç and blobç to the Watchtower, which memorizes them in a file. When Alice loses the information of all her status channels, it sends to the Watchtower the hintç, requesting the correspond blobç. If the Watchtower contains the hintç, it sends to her the corresponding blobç, which can be used to recover all information connected to her channels.

8.2 Watchtower Server

In our implementation, the server code is composed by an interface, WatchtowerService.java and two classes, WatchtowerServant.java and Wacthtower.java. The Interface defines what is possible to be invoked from the client. Essentially, the interface defines the client's view of the remote object. The classes instead provide the implementation, the connection between the client and the server and implement a remote object. They also provide the rest of the code that composed the server program, including the main method that creates an instance of the watchtower remote object and registers it through the RI registry.

8.2.1 Watchtower Server

An RMI server program needs to create the initial remote objects and export them to the RMI runtime, which makes them available to receive incoming remote invocations. The aim of the Watchtower Server is starting the WatchtowerServant and, therefore needs to do the necessary initialization and housekeeping to prepare the server to accept calls from the lightning Node client, Alice.

Before a client can invoke a method on a remote object, it must first obtain a reference to the remote object. The system provides the RMI registry, for finding references to other remote objects. The java.rmi.registry.Registry remote interface is the API for binding remote objects in the registry. The java.rmi.registry.LocateRegistry class provides static methods for synthesizing a remote reference to a registry at a particular network address (host and port). The Watchtower server class creates a name for the object: *watchtower* and adds this name to the RMI registry. This rebind invocation makes a remote call to the RMI registry on the local host. The main method creates an instance of WatchtowerServant and exports it to the RMI runtime (Algorithm 1).

Algorithm 1 WatchtowerServer.java class

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.security.NoSuchAlgorithmException;

public class Watchtower
{
        public static void main(String [] args) throws
        RemoteException, NoSuchAlgorithmException
        {
                Registry reg = LocateRegistry.createRegistry(5099);
                reg.rebind("watchtower", new WatchtowerServant());
                System.out.println("Server is ready");
        }
}
```

8.2.2 The remote Interface WatchtowerService

The WatchtowerService supports the operations *memorize* and getBlob, Algorithm 2. The method *memorize* is utilized, to stores hint, blob, hintç and bobç

Algorithm 2 WatchtowerService.java interface

```
import java.rmi.Remote;

import java.rmi.RemoteException;

public interface WatchtowerService extends Remote {
    void memorize(String s, byte[] blob_ç) throws
    RemoteException, Throwable;
    byte[] GetBlob(String s) throws RemoteException,
    Throwable;
}
```

sent by the client node, in the file. While, the method *getBlob* is used by the client node, Alice, to request to the Watchtower the blobç corresponding to the given hintç.

By extending the interface java.rmi.Remote, the WatchtowerService interface identifies itself as an interface whose methods may be implements by a remote object. As a member of a remote interface, the methods are remote. Therefore, these methods must be defined as being capable of throwing a java.rmi.RemoteException. This exception is thrown by the RMI system from a remote method invocation to indicate that either a communication failure or a protocol error has occurred.

8.2.3 The implementation class WatchtowerServant

The class WatchtowerServant implements the remote interface WatchtowerService and defines the constructor for the remote object.

The implementation class for the interface is declared as follows:

```
public class WatchtowerServant extends UnicastRemoteObject
        implements WatchtowerService
```

This class implements the methods: *memorize* and *getBlob*.

Memorize method
This method aims to store in a file, called *storedFile*, the pair (hint, blob) and (hintç, blobç) sent by Alice, Algorithm 3.
It calls the method *write(String f, String hint, byte[] blob)*, that aims to insert the hint and blob or hintç and blobç, in the *storedFile*. The file *storedFile* is represented in Figure 8.5, where the first line is divided in hint and blob, whereas the second line is divided into hintç and blobç. As it is possible to see,

Algorithm 3 Method memorize()

```
public void memorize(String hint , byte[] blob) throws
RemoteException, Throwable
{
        try {
                write(storedFile, hint, blob);
        }

        catch (IOException e){
                e.printStackTrace();
    }
}
```

for the watchtower is not possible to distinguish between the pair hint, blob and the pair hintç, blobç.

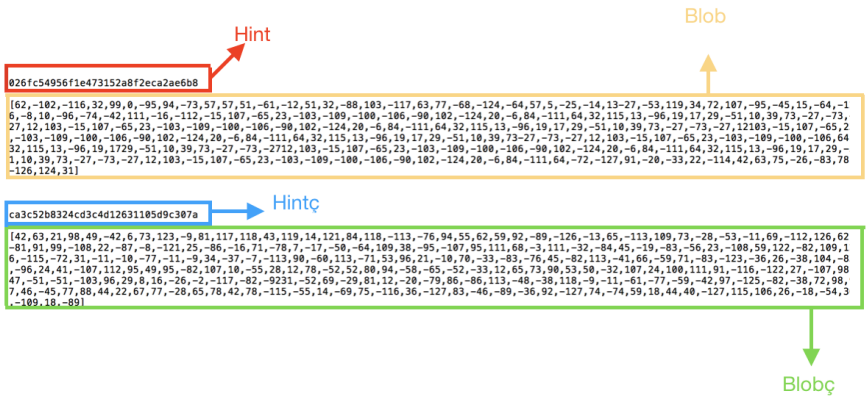

Figure 8.5: *Stored File* how it is possible to see, it is not visible the difference between the (hint, blob) and (hint_ç, blob_ç).

Getblob method

GetBlob calls the method *isContainBlob(string hintToFind)*, that reads all the lines of *storedFile*, and verifies if cointains the hint/hintç requested by Alice. In the positive case, it returns the requested blob/blobç to her (Algorithm 4).

Algorithm 4 Method getBlob()

```
public byte[] getBlob(String hinttoFind)
{
        byte[] blob=null;
        try{
                blob = isContainBlob(hintToFind);
        }
        catch(Exception e)
        {}
        return blob;
}
```

8.3 Client Node

The client, Alice, to look up the Watchtower remote Object, uses the same name that was utilized by WatchtowerServant to register its remote object. In the following part of the code, she invokes several methods on the remote Object. In the next Sections, we will develop the following functionalities:

- Create txidç method
- Calculate hintç and blobç
- Send hintç and blobç
- Request to the watchtowers if it stores the hint/hintç and requests the corresponding blob/blobç

Firstly, Alice creates a connection with the Watchtower. After the connection the node might use one of the methods implementing by the WatchtowerServant, to send the actual status to the Watchtower server.

```
WatchtowerService service = (WatchtowerService)
                Naming.lookup("rmi://localhost:5099/watchtower;
```

Then, Alice initializes the value of txid_Bob, txid_Charlie and txid_Eve:

```
String txid_Bob= "026fc54956f1e473152a8f2eca2ae6b8c
            386704f1a46c900434836f697eea53e";
String txid_Charlie= "026fc54956f1e473152a8f2eca2ae6b8c
            386704f1a46c900434836f697eea5e8";
String txid_Eve= "026fc54956f1e473152a8f2eca2ae6b8c
            386704f1a46c900434836f697eea5e6";
```

Now let suppose, that the channel status between Alice and Bob changes, this implies that also txid_Bob has to be modified:

```
txid_Bob="026fc54956f1e473152a8f2eca2ae6b8
            c386704f1a46c900434836f697eea53e";
            //New channel status between Alice and Bob
```

Create txid_ç method
When a channel status changes, the node must calculate the txidç value. As we mentioned in Section 7.4, the txidç is the result of the SHA256 function on the public key.

To calculate the public key, we have utilized the website: *http://bip32.org*, in Figure 8.6, utilizing our seed and the Derivation Path.

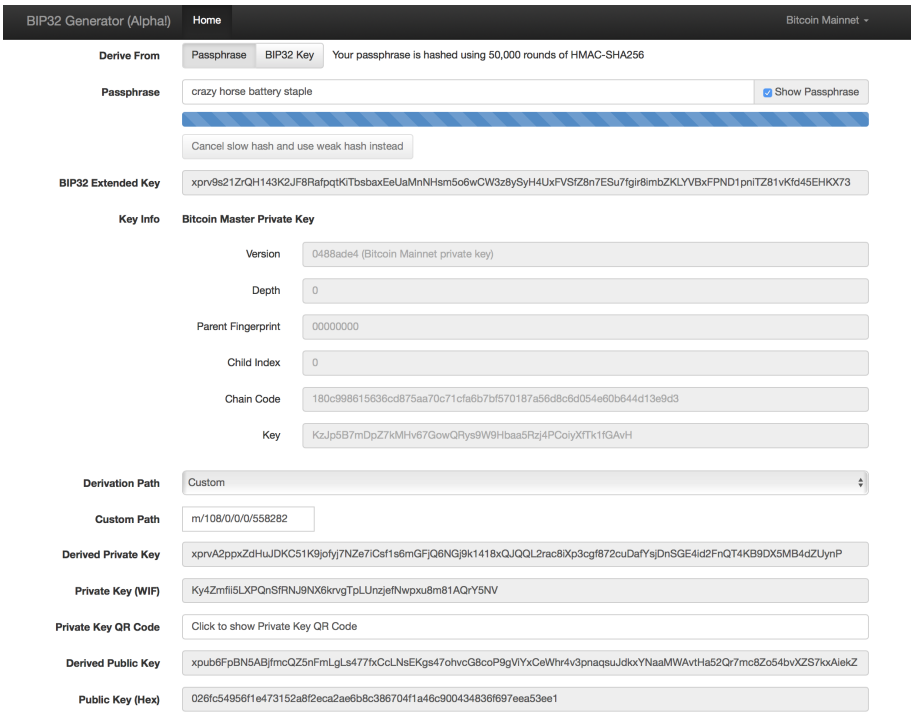

Figure 8.6: BIP32 Deterministic Key Generator.

As Derivation path, we have used m/108/0/0/0/558282, where 558282 is a casual block_height and 108 indicates the purpose to use the Açai Protocol (Section 7.4).
In the algorithm, we simplify the case in which the transaction is the first in the block, so the txid is equal to two times the SHA256 of the public key.

Once obtaining the public key, Alice calculates the txid with the method *calculate2SHA256(pubKey)*, that accepts as input the public key obtained through the BIP32 function (Section 3.6.2):

```
String pubKey="026fc54956f1e473152a8f2eca
                    2ae6b8c386704f1a46c900434836f697eea53ee1";
String txid_ç= calculate2SHA256(pubKey);
```

Calculate hint_ç and blob_ç

The algorithms to calculate the hintç and the blobç, use the function *Truncate* to split in two equals parts the txidç, obtaining the txidç[:16] and txidç[16:]. In the case of the creation of the blobç, also the parameter of dataç are using, in order to encrypt with the txidç[16:]. For the encryption, we have decided to use the AES Encryption Standard. The Algorithm 5 and the Algorithm 6, show how to create hintç and blobç respectively.

Algorithm 5 Create hintç(the same method can be used to create hint)

```
public static String CreateHint(String s)
{
        String[] truncated= Truncate(s);
        return truncated[0];
}
```

Algorithm 6 Create blobç(the same method can be used to create blob)

```
public static byte[] CreateBlob(String txid_ç, String data)
{
        String[] truncated= Truncate(txid_ç);
        String txid= truncated[1];
        Key aesKey = new SecretKeySpec(txid.getBytes(), "AES");
        byte[] blob= null;
        try
        {
                blob= encryptAES(aesKey, data);
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
        return  blob;
}
```

Send hint_ç and blob_ç
After creating the hint, blob, hintç and blobç, Alice sends them to the Watchtower utilizing the remote method *memorize*, Algorithm 7.

Algorithm 7 Send hint, blob, hintç and blobç to the Watchtower through the remote method *memorize*

```
try
{
        service.memorize(hint, blob);
        service.memorize(hint_ç, blob_ç);
}
catch (Throwable e)
{
        e.printStackTrace();
}
```

The class WatchtowerService implements the function memorize, which aims to store in the *stroredFile* the hint, blob hintç and blobç received from the Lightning Node.

Verify if the Watchtower has the hint_ç
Suppose that Alice has lost all own data of her status channels. In order, to recover this information using the Açai Protocol, she has to request the most recent BlockHeigh value to the other nodes and then utilize the BIP32 Generator key to recover the Public Key, using her seed and the Derivation Path with the purpose code 108. After calculating the hintç, she may request to the Watchtower, calling the method *getBlob*, if it contains the blobç corresponds to the sent hintç.

```
blob_ç= service.GetBlob(hint_ç);
```

If the Watchtower contains the corresponded blobç, it returns the value to Alice.

As we have mentioned at the beginning of this Chapter, we have assumed that the lightning Node has a unique transaction inside the last Block height. Thus, the txidç is equal to 2SHA256(Public key) (see Section 7.4).

Recover data
After obtaining the blobç, Alice needs to decrypt it in order to get the dataç inside. She decrypts the blobç with the txidç[16:], using the method *getData*, and she obtains the dataç. In this way, she may know the txid with Bob (txid_Bob), with Charlie (txid_Charlie) and with Eve (txid_Eve)
To recover all information connected to each channel status, Alice has to calculate the hint and then use again the method *getBlob* to obtain each data for each channel.

```
data_ç= getData(txid_ç, blob_ç);
Scanner in= new Scanner(data);
while(in.hasNextLine())
{
        String str=in.next();
        String name=in.next();
        if(name.compareTo("Bob")==0)
            txid_Bob=str;
        else if(name.compareTo("Charlie")==0)
            txid_Charlie=str;
        else
            txid_Eve=str;
}
```

Where dataç is:

```
data_ç
026fc54956f1e473152a8f2eca2ae6b8c
386704f1a46c900434836f697eea53e(txid_Bob)


026fc54956f1e473152a8f2eca2ae6b8c
386704f1a46c900434836f697eea5e8 (txid_Charlie)


026fc54956f1e473152a8f2eca2ae6b8c
386704f1a46c900434836f697eea5e6 (txid_Eve)
```

Thus, Alice is able to recover the txid for each channel status and derive the current data.
For example, if Alice recovers the channel status with Bob:

```
hint= CreateHint(txid_Bob);
blob= service.GetBlob(hint);
data= getData(txid_Bob, blob);
```

she obtains this data with Bob:

```
data with Bob Status Channel: Bob=0.8 BTC Alice=0.1 BTC
```

Conclusion
With this project, we have demonstrated how Alice may recover all status channel, only having the public key, derived from the derivation path, and her seed. Thus, through the seed, Alice is able to recover not just her Bitcoin wallet funds (Section 3.6.3), but also her Lightning wallet funds.

CHAPTER 9
Evaluation

This Section will attempt to give an assessment to the suggested Açai protocol, analyzing the main results, how it might affect the Lightning user and a possible drawback.

In Section 7.4, we have studied Açai protocol, its mechanism and its main features, that permit this solution to work. Furthermore, we have seen a JAVA RMI implementation in Chapter 8, that aims to give a demonstration of the communication between the Lightning node and the Watchtower.

Main results
The Açai Protocol allows having a mechanism to backup data for a Lightning Network wallet. It satisfies all the proposed goals of Section 5.3. First of all, it maintains the decentralized nature of Bitcoin: in fact, it does not rely on any on-cloud or any other external service. It safeguards the anonymity since its mechanism is not based on the use of public keys, which might identify the user as we have seen in 3.6. Through the use of the SHA256 hash function, it maintains the integrity and confidentiality of the stored data. Furthermore, through the Watchtowers, it is able to guarantee an immediate recovery solution. Regarding the implementation, as demonstrated in Chapter 8, it is not complex to develop it since the mechanism uses the same structure that the Watchtowers use for monitoring the channels. Moreover, thanks to this solution, the Lightning nodes are able to recover all funds both in their Bitcoin Wallet and in their Lightning Wallet, through their own seed (Section 3.6.3).

User experience
From the user experience, the user has not to do anything to execute the Açai protocol: this may work without the active intervention by the user, that will be able to recover the own status channel automatically through its own seed.

Game Theory, adversarial thinking and economic principles in the solution
This solution also respects the Nash equilibrium, in fact, we do not have a cryptographic solution to mitigate the case in which the Watchtower is not cooperative with the user, for example affirming to not contain a specific hint. On the other hand, game theory principles may be reasonably used to expect the Watchtowers service to work.
Watchtowers are paid to offer the backup service, through micro-fee every time that a channel is closed, so they are economically incentivized to continue to guarantee the service. Moreover, the watchtowers may not distinguish the different users, so they are not able to ban Alice and the Açai Protocol since it is indistinguishable from normal transactions.

Açai Protocol: for a frequent or sporadic user?
One possible criticism of our protocol is that it is more indicated for frequent users, with high use of Lightning network, for example, minimum one or more transactions for every one-two blocks.
In fact, for a normal daily user is high power consumption to calculate the *txid_ç* using the value of the block height. Since there is a new block every 10 minutes, it is quite unlikely that a user has his/her last transaction every time in the most recent block. On the other hand, in case of a big company, for example, an e-commerce company, the probability to have transactions in the last block is higher.
To customize the proposed protocol for sporadic users, it would be useful using as the last parameter in the derivation Path an integer, with the functionality of a *counter*. The incrementation of this counter permits to have a binary research counting and on a channel that lives more than 32 blocks, this solution permits to do fewer steps.

```
Derivation Path= m'/108'/0(mainnet)/(account number)'/0/counter

                                        where counter=0,1,2...n
```

CHAPTER 10
Conclusion

This Chapter concludes the thesis. It reports the conclusions of the whole study, describing its main contributions and discussing results and possible future work.

10.1 Contributions

The aim of the thesis was to analyze Bitcoin and Lightning technology, their mechanism and their main features, ending up with a technically feasible solution to mitigate the recovery of unspent Bitcoin after a Lightning wallet failure. Accordingly, the main goal of the work may be considered successfully fulfilled, offering a solution that allows recovering the status channels in case of wallet failure. Moreover, all the thesis goals, stated in Sections 1.1 and 5.3 were successfully accomplished and they conduct this work in the desired direction.

The main contributions of the thesis are examined below.

Analysing the Bitcoin technology, Blockchain and the Scalability problem
In Chapter 1, 2 and 3, we have offered an analysis of Bitcoin explaining its mechanism and its main features. In more details, in Chapter 2 we have introduced the main aspects and we have described its advancement, through some statistics and the Scalability Problem. In Chapter 3, we have proposed an overview of Nakamoto white paper [Nak], integrating it with the most recent literature.

Studying Lightning Network and its main properties.
The main focus of this thesis was Lightning Network, a quite new protocol that, as we have seen, promises to solve the Scalability Problem. In Chapter 2 and 4, we have examined it deeply, studying the white paper written by Poon and Dryja ([PD16]), and demonstrating through some statistics, that despite its infancy, it is already widespread.

Performing a literature survey of the state of the problem.
In Chapter 5, we have performed a literature survey of the state of the problem. More specifically, we have analyzed the solution proposed with Eltoo, through the Electrum Wallet, a Lightning Wallet based on Olympus server and Piln. We might demonstrate that there is not a solution that solves completely the addressed problem and respects the decentralized nature of Bitcoin technology. Additionally, in Chapter 6.1, we have explained the methodology that we have adopted in our literature research.

Propose a possible solution to the recovery mechanism in Lightning Network.
In Chapter 7, we have proposed Açai Protocol: a solution that permits to recover the status channels in case of Lightning wallet failure. More precisely, it bases its mechanism on using the Watchtowers not just for monitoring the channels, but also as a backup service. As we have demonstrated in Section 7.4, it respects the decentralized nature of Bitcoin, the anonymity of users, integrity and confidentiality of data and allows to have an immediate recovery service. In conclusion, since the concept of Watchtower is quite new and it has not been officialy formalized yet, we have proposed our definition of Watchtower, which includes our mechanism of backup.

Implement the Açai Protocol
In Chapter 8, we have delineated a possible implementation of the proposed Açai Protocol. Through our JAVA RMI project, we have demonstrated how a Lightning node may recover all status channel, only having the public key that is derived from the derivation path, and the own seed. Hence, through the use of the seed, a Lightning Node is able to recover not just its own Bitcoin wallet funds, but also its own Lightning wallet funds. One of the drawbacks of our system has been built on the concepts of Eltoo and Watchtowers, that are not still available in Lightning Network. We are confident that the system may be develop to guarantee a backup mechanism in the next future.

10.2 Results and Future Works

As we have already mentioned in Chapter 1, Bitcoin and Lightning are quite new technology and a lot of work has to be done, to permit these two technologies to improve continuously. In this section, we propose some possible Future projects, that might extend further the work done in this thesis to include new interesting features.

Watchtower service
This protocol may be proposed from wallet providers and e-commerce companies, which they might manage a backup service through watchtowers and offer the solution to their own clients, in charge of a fee.

Lightning Wallet including the Açai Protocol
As we have analyzed in Chapter 5, there are not applications that may offer a backup solution without utilizing an on-cloud service or centralized systems. Therefore, it would be very useful to realize a Lightning wallet that permits to guarantee a backup service through the Açai Protocol, maintaining a decentralized nature, typical of Bitcoin.

Introduce IPFS system in the Açai Protocol
InterPlanetary File System (IPFS) is a protocol and network designed to create a content addressable, a peer-to-peer method of storing and sharing hashed

hypermedia files in a distributed file system [Fin16]. A possible idea is creating a shared hashed file where different watchtowers may store all data information for back up. In this way, a Lightning node may ask to one of all watchtowers, that can access the IPFS file, the information to recover its own status channel. Moreover, this permits to solve the problem of memory storage in the watchtowers. In fact, in this solution, it is assumed that data stored in the watchtowers are not deleted or substituted/tampered. In real life, this is causing a waste of memory and having a solution, which includes IPFS, might be helpful.

Trade-off solution for frequent and sporadic user
As we have discussed in previous Chapter 9, Açai protocol is designated mainly for a frequent use and might result not convenient for a sporadic user. We might consider it a sort of trade-off: offering a useful solution for frequent user, at the expanse of sporadic users. It would be interesting analyzing how this solution might be modified to permit to be convenient for each type of user, maintaining all its features, such as privacy and decentralization. A possible idea might be to create two versions of the Açai Protocol: one dedicated to the frequent user, based on the design presented in Section 7.4, and one for sporadic user as presented in Chapter 9.

Bibliography

[And12] G. Andresen. *Bip 16: Pay to script hash*. Jan, 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (cited on page 17).

[Ant] Andreas M. Antonopoulos. *Bitcoin Q&A: Eltoo, and the early days of Lightning*. URL: https://www.youtube.com/watch?v=%20o6eFZ5aI9N0&feature=youtu.be&fbclid=IwAR3chExMAD3khj%20EYazFa0rMSvUeKzp2196HGb-vhbg6kU-u6awhJqu1oFCo (cited on page 44).

[Ant14] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. 2014 (cited on pages 1, 5–7, 9, 22–31, 33, 35, 39, 41, 42, 49).

[Aso18] Bisola Asolo. *Mempool Explained*. Nov 1, 2018. URL: https://www.mycryptopedia.com/mempool-explained/ (cited on page 18).

[Bit18] BitRewards. *Blockchain Scalability: The Issues, and Proposed Solutions*. Apr 25, 2018. URL: https://medium.com/@bitrewards/blockchain-scalability-the-issues-and-proposed-solutions-2ec2c7ac98f0 (cited on page 12).

[Blo18a] Blockchain-Hub. *Cryptography and Blockchain*. Sep 10, 2018. URL: https://blockchainhub.net/blog/blog/cryptography-blockchain-bitcoin/ (cited on page 51).

[Blo18b] Blockchain.info. *Blockchain Size*. Sep 29, 2018. URL: https://www.blockchain.com/charts/blocks-size?timespan=3years (cited on page 9).

[But17] Vitalik Buterin. *The Meaning of Decentralization*. Feb 6, 2017. URL: https://medium.com/@VitalikButerin/the-meaning-of-decentralizati%20on-a0c92b76a274 (cited on page 8).

[Can18] Canya. *What is Game Theory and how is it applicable to Cryptocurrency?* May 28, 2018. URL: https://medium.com/canyacoin/what-is-game-theory-how-is-it-applicable-to-cryptocurrency-6bf1fa5b14e2 (cited on page 51).

[CK17] Jonathan Chiu and Thorsten V Koeppl. *The economics of cryptocurrencies–bitcoin and beyond*. 2017 (cited on page 1).

[Con+18] Mauro Conti et al. *A survey on security and privacy issues of bitcoin*. 2018 (cited on pages 16, 20, 21).

[Con18] Jonathan Chester Contributor. *Your Guide On Bitcoin's Lightning Network: The Opportunities And The Issues*. Jun 18, 2018. URL: https://www.forbes.com/sites/jonathanchester/2018/06/18/your-guide-on-the-lightning-network-the-opportunities-and-the-issues/#5012c11e3677 (cited on pages 2, 11).

[Cro+16] Kyle Croman et al. *On scaling decentralized blockchains*. Springer, 2016 (cited on pages i, 11).

[Des18] Coin Desk. *Bitcoin Calculator*. Jan 10, 2018. URL: https://www.coindesk.com/calculator (cited on page 5).

[Dev18] Mycelium Developers. *Mycelium Bitcoin Wallet*. Dec 15, 2018. URL: https://play.google.com/store/apps/details?id=com.mycelium.wallet&hl=en_US (cited on pages 5, 6).

[DRO] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. *eltoo: A simple layer2 protocol for bitcoin* (cited on page 43).

[Dry] Tadge Dryja. *Unlinkable outsourced channel monitoring*. URL: https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinka%20ble-outsourced-channel-monitoring/ (cited on page 58).

[Eco15] The Economist. *The great chain of being sure about things*. Oct 31, 2015. URL: https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things (cited on page 2).

[Fin16] Kurt Finley. *The Inventors of the Internet Are Trying to Build a Truly Permanent Web*. June 20, 2016. URL: https://www.wired.com/2016/06/inventors-internet-trying-build-truly-permanent-web/ (cited on pages 47, 89).

[Gmb] Electrum Technologies GmbH. *Electrum Bitcoin Wallet*. URL: https://play.google.com/store/apps/details?id=org.electrum.electrum&hl=en_US (cited on page 46).

[how18] howMuch.net. *Transactions Speeds: How Do Cryptocurrencies Stack Up To Visa or PayPal?* Jan 10, 2018. URL: Transactions%20Speeds:%20How%20Do%20Cryptocurrencies%20Stack%20Up%20To%20Visa%20or%20PayPal? (cited on pages 10, 11).

[HR17] Garrick Hileman and Michel Rauchs. *2017 Global Cryptocurrency Benchmarking Study*. 2017 (cited on page 1).

[Inc16] Blockgeeks Inc. *What is Bitcoin? The Most Comprehensive Guide Ever!* Oct 8, 2016. URL: https://blockgeeks.com/guides/what-is-bitcoin/ (cited on page 2).

[Inv18] Investopedia. *Game Theory definition*. May 30, 2018. URL: https://www.investope%20dia.com/terms/g/gametheory.asp (cited on page 50).

[Kon18] Shedrach Kongvong. *The State of Lightning Integration in Electrum*. Oct 19, 2018. URL: https://bitcointechweekly.com/front/update-on-lightning-integration-in-electrum/ (cited on page 45).

[Lar18] Aleks Larsen. *A Primer on Blockchain Interoperability*. Dec 21, 2018. URL: https://medium.com/blockchain-capital-blog/%20a-primer-on-blockchain-interoperability-e132bab805b (cited on page 12).

[Lev15] Alexander Levakov. *Nash equilibrium: a case study*. March, 2015. URL: http://rstudio-pubs-static.s3.amazonaws.com/%2067398_890ed9%20c3ff97466a8dd33472e6118f19.html (cited on page 50).

[Lui18] Alberto de Luigi. *Bitcoin e adozione di massa: cosa (e quanto tempo) manca a Lightning Network*. Nov 3, 2018. URL: http://www.albertodeluigi.com/2018/11/02/bitcoin-mass-adoption-lightning-network/#4 (cited on page 46).

[Mer87] Ralph C Merkle. *A digital signature based on a conventional encryption function*. Springer, 1987 (cited on page 17).

[Mye13] Roger B Myerson. *Game theory*. 2013 (cited on page 50).

[Nak] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system* (cited on pages 4, 6, 7, 10, 15, 16, 18, 19, 21, 87).

[Net16] Financial Crimes Enforcement Network. *Statement of Jennifer Shasky Calvery, Director Financial Crimes Enforcement Network United States Department of the Treasury Before the United States Senate Committee on Banking, Housing, and Urban Affairs Subcommittee on National Security and International Trade and Finance Subcommittee on Economic Policy*. Oct 9,2016 (cited on pages 5, 15).

[Net18a] Lightning Network. *Lightning Network Summary*. Dec 27, 2018. URL: https://lightning.network/lightning-network-technical-summary.pdf (cited on pages 3, 12, 13).

[Net18b] Lightning NetworkAuxledger. *Blockchain Scalability Hindrance and How to Overcome It*. Nov 18, 2018. URL: https://medium.com/auxesis/blockchain-scalability-hindrance-how-to-overcome-it-54378166b804 (cited on page 11).

[OR94] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. 1994 (cited on page 50).

[PD16] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016 (cited on pages 2, 13, 38, 45, 87).

[Pea16] Jordan Pearson. *Bitcoin Unlimited' Hopes to Save Bitcoin from Itself*. Oct 14, 2016 (cited on page 2).

[Put+] Deepak Puthal et al. *The blockchain as a decentralized security framework* (cited on pages 2, 8).

[SA18] BLOCKCHAIN LUXEMBOURG S.A. *Blockchain charts*. Dec 23, 2018. URL: https://www.blockchain.com/en/charts (cited on pages 1, 7).

[she18] Medium sheinix. *Payment Channels in Bitcoin*. Apr 7, 2018. URL: https://medium.com/coinmonks/payment-channels-in-bitcoin-470b28e4%207bb0 (cited on page 33).

[Tut] Tutorialspoint. *Java RMI - Introduction*. URL: https://www.tutorialspoint.com/java_rmi/%20java_rmi_introduction.htm (cited on page 73).

[Vis19] Bitcoin Visuals. *Lightning Network Channels*. Jan 1, 2019. URL: https://bitcoinvisuals.com/ln-channels (cited on pages 3, 12).

[Wal] Lightning Wallet. *What does Olympus server do*. URL: https://lightning-wallet.com/using-lightning-wallet#forced-channel-closure (cited on page 47).

[WP17] Oscar Williams-Grut and Rob Price. *A Bitcoin civil war is threatening to tear the digital currency in 2 — here's what you need to know*. March 26, 2017 (cited on page 2).