POLITECNICO DI TORINO

Master's Degree Thesis

Computer Engineering - Software

Advanced data management on Distributed Ledgers: design and implementation of a Telegram BOT as a front end for a IOTA cryptocurrency wallet

Candidate: Francesco Longo

Supervisor: Giovanni Malnati
Co-supervisor: Edoardo Calia

Academic year 2017-2018

Summary

Nowadays the 98% of the information collected by humans in the world is recorded in digital format. Many of our daily actions generate data digitally stored somewhere in the world. Often this process is automatic, due to the many devices and appliances equipped with sensors that we use every day (in contexts like automotive, home automation, industrial machines, up to consumer gadgets). As a consequence, digital data grow exponentially. Whoever owns this data can turn it into an enormous economic value (but only if they know how to analyze them). The collected data can be used to customize advertising for production optimization and products distribution, for the predictive maintenance of industrial machines and much more. Unfortunately the majority of these data sets are stored in a few global centers, generating an unsustainable inequality, and at the same time exposing the data to cyber attacks and subsequent threats to privacy and security of citizens. The value which is associated to data is so large that someone proposed a comparison with the old economy saying that "Data is the new oil": data are extracted, refined and exchanged through a new generation of infrastructures represented by the telecommunication networks, cabled or wireless. The data economy and the use of Distributed Ledger Technologies (DLT) are the starting points of my Thesis. My work started with the analysis of different DLTs to find potential use cases at the intersection of these two topics, and later I focused on the design and implementation of an application prototype using a specific DLT. One of the greatest innovations of the last 10 years is Bitcoin, the first digital cryptocurrency. We often hear about Bitcoin for its economic value, which increased significantly since its introduction, and its volatility, which makes it very fluctuating and unstable, generating in time both gains and losses. This great volatility and the associated potential ease of earning attracted many people (traders and investors), leading to record growth. The Blockchain is the architecture behind Bitcoin and it is the real innovation of this system. This architecture enables users who have no trust among each other to create and maintain a shared network dedicated to the exchange of financial virtual (crypto) assets, offering reliability, consistency, immutability without using a central entity who takes the responsibility for the provided service. Bitcoin uses the blockchain as a Distributed Ledger: a sequence of blocks linked to each other. Each block refers to the previous one and contains a set of transactions. Special users named miners validate new transactions and store them in the global ledger (the blockchain). On average, a block (the structure containing transactions) is mined every 10 minutes. The mining process consist in finding a complex solution for an np-complete mathematical problem that requires a lot of time and computer resources to be solved. Miners keep their own copy of the blockchain, synchronized

with the entire network, and compete to solve this difficult mathematical problem based on a cryptographic hash algorithm, carrying out what is known as a Proof-Of-Work. This basically shows that a miner did spend a lot of time and resources to solve the problem. When a block is 'solved', the transactions contained are considered confirmed, and the amount indicated in the transactions can be spent by their owners. Once a block is mined and confirmed by the other nodes in the network, it cannot be modified or replaced in the future, because every block includes the hash of the previous one. From a certain point of view, blockchain offers the requested decentralization, but gives power to other entities (the miners) that, in fact, control this the network and maintain it. Considering the data-centrality vision of our modern economy, it is natural to think whether or not it is possible to use a DLT structure for the storage of general purpose data (or sensitive encrypted data) and not only financial transaction data. Currently the Blockchain is not the only existing DLT, but there are several other architectures that, taking a cue from Blockchain and offering the same advantages, aim to solve its weaknesses: high energy consumption, slow transactions confirmation, not-centrality of the data, limited scalability and high fees associated with each transaction. For my thesis I have chosen to study and analyze IOTA, a DLT created specifically for the IoT and focused on the data granularity. IOTA is called "a Blockchain without Block and without Chain" because it promises the same advantages of a Blockchain using, however, a different structure called Tangle. The tangle is basically a Directed Acyclic Graph (DAG) where, instead of building a chain of blocks added at specific intervals, transactions are sent directly, and each one becomes a node of the graph. In IOTA there are no transaction fees. As the Tangle grows and more users make transactions, the whole system becomes faster, more robust and more secure. The goal of my thesis was to understand and analyze data storage based on Distributed Ledger Technologies (DLTs). After a little period studying the Blockchain architecture I focused on IOTA. Nowadays the way to store and secure large amounts of sensitive data is moving from the classical database to DLTs, where security, availability and consistency are essential. The future is seen as a set of devices that save data and interface through the web to store and exchange information with each other: this is the IoT vision, where many features and scenarios become possible: machine economy, digital twin, faster transactions and all of this in a scalable environment. These goals can be reached in a more feasible way using DLTs. The main innovation behind IOTA is the Tangle, a revolutionary new blockless distributed ledger which is scalable, lightweight and for the first time ever makes it possible to perform a transaction, in which any kind of data can be stored without any fees. Differently from the Blockchain paradigm, in IOTA consensus is no-longer delegated to a sub-community of users (the miners) but is instead reached through the active contribution of all users (therefore becoming an intrinsic part of the system), leading to a real decentralized and self-regulating peer-to-peer network. I started by studying the theory behind the IOTA architecture and the Tangle structure understanding the theoretical principles described in the IOTA whitepaper. Then I have studied how to start and maintain a full-node (IOTA application to retrieve and synchronize data on the Tangle) manually-managed using the IOTA Reference Implementation (IRI). I set up and managed 3 full-nodes fully synchronized, in-

stalled at ISMB. As an IOTA developer, I learned how to use the dedicated client libraries (JAVA, iota.lib.js) in order to interact with the Tangle. As my Master's Degree Thesis project I developed a Telegram Bot managing an IOTA wallet for each user. This bot permits to send and receive IOTA tokens easily from inside a chat. I decided to develop my project using the IOTA Javascript library to exploit all the potential of nodeJS and npm packets manager.

Acknowledgements

I would like to thank Professor Giovanni Malnati for giving me the opportunity to continue our academic relationship proposing me this research thesis about stimulating and advanced topics. I'm grateful to the ISMB research institute (Istituto Superiore Mario Boella), SiTI and Links Foundation for providing me with a comfortable workspace and environment to accomplish this thesis project. A special thank you to the deputy director of ISMB Edoardo Calia for assisting me in carrying out this thesis and for always giving me all the necessary support during my stay in ISMB. Another special thank to Dr. Andrea Vesco for his advice and for helping me in times of need.

A generous thank you thank you to Chris Dukakis, Lewis Freiberg and Giogio E. Mandolfo that helped me to understand details about IOTA libraries during my thesis.

A personal thank you to the members of "Links Foundation" team: Andrea Vesco, Michele Osella, Gian Marco Toso, Alberto Buzio, Jure Rosso, Edoardo Calia for sharing with me their experiences during the EUIPO Blockathon 2018 competition in Brussels.

I would like to remind and thank you my colleagues, classmates, and friends that have cheered up those days spent working on my studies and thesis making this period a pleasant experience.

Thank you to my extended family (geographically near and far): grandparents, uncles, cousins for all their advices, positive and not, they gave me. A big thank you to my family: my parents that allowed me to carry on all this giving me all the necessary, material good and not, to move forward and my two sisters for their support.

Last but not the least, a special thank you to my life partner Alessia for always being by my side and for supporting me throughout this my long journey appreciating me as I am and never stopped believing in me.

Contents

Chapter 1

Introduction: the new data-economy

Nowadays 98% of the information collected by humans is recorded in digital format. Many of our daily actions generate data digitally stored somewhere in the world. Often this process is automatic, due to the many devices and appliances equipped with sensors that we use every day (in contexts like automotive, home automation, industrial machines, up to consumer gadgets). As a consequence, digital data grow exponentially. Whoever owns this data can turn it into an enormous economic value (but only if they know how to analyze them).

1.1 The data-centrality of the new economy

Availability, Integrity and confidentiality of digital data are three essential requirements for a correct management of the huge amount of information that companies, often unconsciously, already have available. In fact, what is missing most of the time is the ability to extract useful and structured information by combining data that already resides on the company IT systems.

The increasingly advanced interconnection of IT systems with each other is accompanied by the increase and availability of software capable of analyzing and processing data. In fact, we are moving from accumulating raw data in big heterogeneous repositories known as big data or data lakes , to a more sophisticated paradigm where complex analyses are carried out on the same data, generating smart data (real information), processed and immediately usable by the company's management. Those data can create competitive advantages on the market through the generation of significant information to establish the characteristics of a new product or service, set business strategies and evaluate the positioning on the reference market. More in detail, the large amount of data generated from different sources such as mobile phones, credit cards, web traffic, online purchases or other devices, has contributed to automatically generate a mass of information, without having to go to collect them on demand, as happened previously, requiring appropriate acquisition activities. Today, what it is really needed is a proper way to consult those data in an understandable format in order to satisfy the needs of consultation, representation and storage of information. It is also necessary to ensure the information security of what constitutes a strategic company asset, thus managing the risks, the network, access control, privacy and system compliance. At

the same time it is necessary to develop or make use of analytical tools, moving on from the data collection to its processing to extract useful information, essentially passing from a quantitative accumulation of data to a selection aimed at achieving the quality of the same as smart data.

1.2 The data analysis process

A typical analysis process consists of several phases:

- data collection, in which the data is acquired verifying its reliability;
- data management, in which the information is managed and stored according to the needs of the subsequent phase (functional data science) aimed at extracting useful information for the company;
- data visualization, or how the extracted information is presented to make it effectively usable.

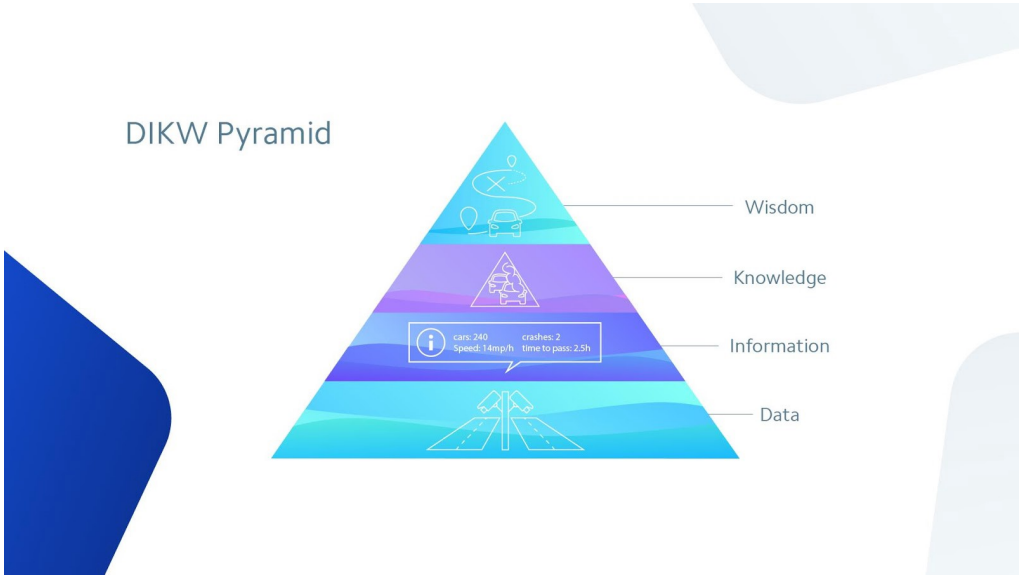
Quality and ease of use constitute the added value that allows big data to become smart data that can actually be used as a company asset. The wide adoption of smartphones and mobile apps, together with tracking tools used in web sites make it possible to perform an accurate profiling of Internet users. A user can be tracked

saving its IP address and saving data referring to each interaction, collecting more and more details for each user in every browsing session. The youronlinechoices.com portal offers Internet users the possibility to check the amount of data collected from their online activity, also offering the possibility to disable some of the tracking. Data is one of the most imperative ingredients in the machine economy and the connected world.

1.3 The data: a fundamental ingredient

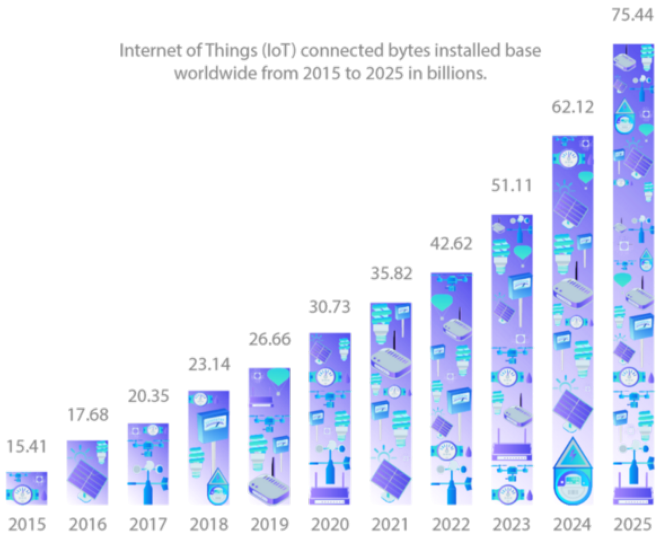
David Sønstebø (Founder of IOTA) wrote the following text as part of the presentation of the IOTA Data Marketplace project:

"Data is one of the most fundamental ingredients in the machine economy and the connected world. It is the foundation upon which the other strata of the Data, Information, Knowledge, Wisdom (DIKW) Pyramid are dependent. Without this primary substrate, nothing can be established about the world. A datum (plural: data) is the raw value of a qualitative or quantitative variable; a pure unfiltered input from reality. Data sets that are structured and thus have had meaning extracted from them constitute what we consider to be information. Information tells us something concrete and coherent about the world through context. Contextualized information thus makes up what we know as knowledge, which in turn gives rise to our (and our machines') ability to make wise decisions. Wisdom is knowledge applied. The illustration above shows how the different echelons of this hierarchy are intertwined. The sensors, here cameras, monitoring the road are receiving signals from the photons reflecting off the surface of their environment. This is data. The measured speed of the moving objects and their relative positions is the meaningful information extracted from these datum inputs. This information is next contextualized and distilled into knowledge, telling the drivers on the road via Over-The-Air updates that the road may be congested due to a crash, which allows drivers to apply the wisdom to take an alternative route."



1.4 Machine Economy

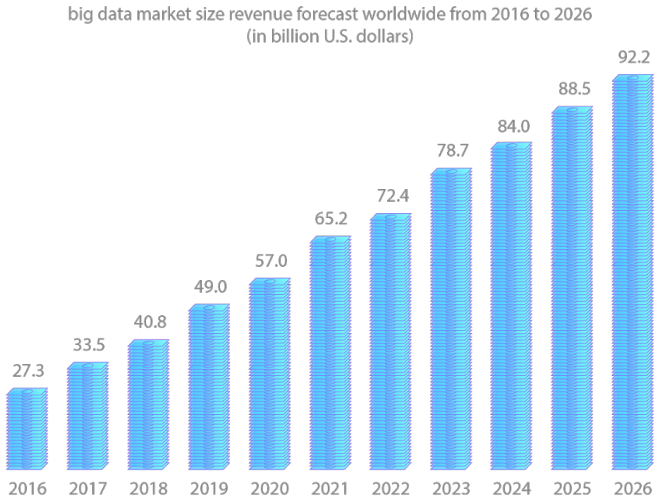
In the next 10 years there will be more than 75 billion connected devices that interact each other in different manners. This will increase the so-called "machine economy" where devices will be enabled to autonomously trade resources like storage, computation/analytics to electricity and sensor data.



In 2017 more data was generated than in the past 5000 years. Of course the "digital age" started recently but these numbers are destined to grow exponentially in the next decade.

1.5 Data Economy

The sheer magnitude of data and the influence it has had - and increasingly will have - on our society comes with huge business opportunities that will be worth tens of billions over the next few years. Trading data will be a mutually beneficial exercise that boosts innovation for companies and creates entirely new revenue streams from data that would otherwise simply be ignored or eventually deleted. The largest obstacle preventing the fulfillment of the grandeur envisaged by 'Big Data' is the fact that the overwhelming majority of data remains locked in what are called 'Data Silos'. Data silos do not, or at best very rarely, share data outside their own closed environment. This leads to enormous quantities of wasted data: often over 99% of available data is lost to the void (source: McKinsey 2015), data that could potentially contain extremely valuable information if allowed to flow freely in data streams that create an open and decentralized data lake made accessible to any compensating party.



1.6 Data wants to be free, but not for free

There are several reasons for the cumbersome and wasteful status quo. On the one hand, data wants to be free in the sense that its storage and transmission costs less and less over time; on the other hand, large quantities of data are extremely valuable and are not free to generate. These diametrically opposed conditions cause a gridlock that needs to be broken in order for Big Data to become truly big. A major cause of this is the fact that, while data sharing is becoming cheaper from a technological perspective, it is prohibitively expensive to sell fine, granular data in real-time due to intermediary fees, not to mention all the red tape one has to cut through in order to complete a single data purchase. These conditions make real-time data trade all but possible. By 2025 it is projected that around 95% of all data will be generated by IoT devices in real-time (source: IDC 2017), so this is a pressing issue. A third obstacle is the lack of ensured authenticity and audit trails of data. Before adoption of Distributed Ledger Technologies, data transmission protocols and databases were susceptible to various attacks, including 'man in the middle' attacks and data tampering. Data is only as valuable as it is valid. In short, if the data input is garbage, the output will also be garbage (GIGO).

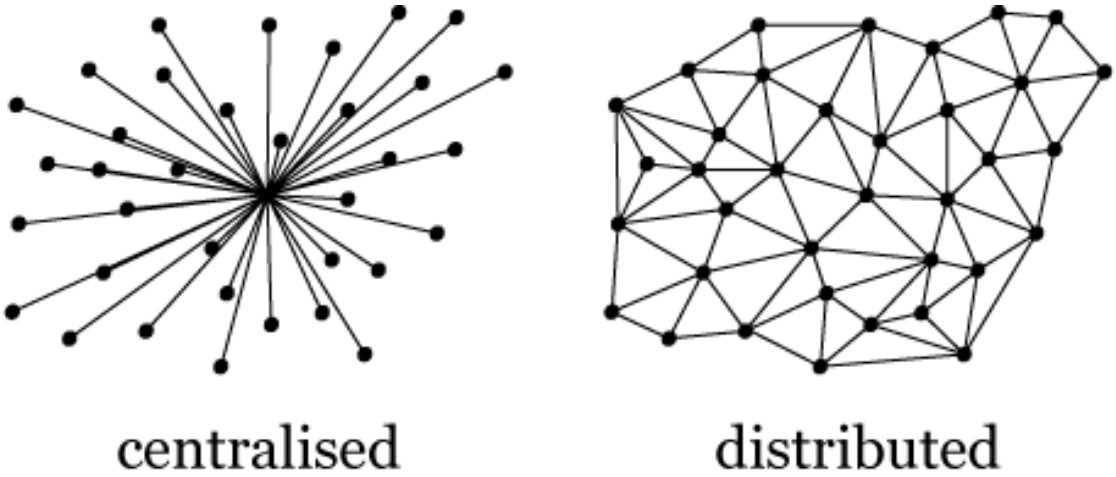
This is why the IOTA architecture and protocol (see IOTA in chapter 3 and more details regarding Data Marketplace in appendix C) play a key role in unlocking data's gigantic potential.

Chapter 2

Distributed Ledger Technologies

A distributed ledger (the concept behind the emerging distributed ledger technologies or DLT) is a peer to peer solution aimed at safely maintaining a list of records in a robust and secure way. A DLT is characterized by the lack of a central authority or maintainer, replaced by a community of peer participant nodes. One of the key goals of DLTs is the ability to function properly in a trustless environment (provided that a majority of the participating nodes are not malicious). Thanks to the distributed consensus mechanism, the DL guarantees the immutability of records and the proper identification of the users that originated every single transaction. Many DL architectures and protocols implement BFT (Byzantine Fault Tolerance), meaning that they tolerate up to 1/3 of nodes to be malicious.

2.1 Introduction



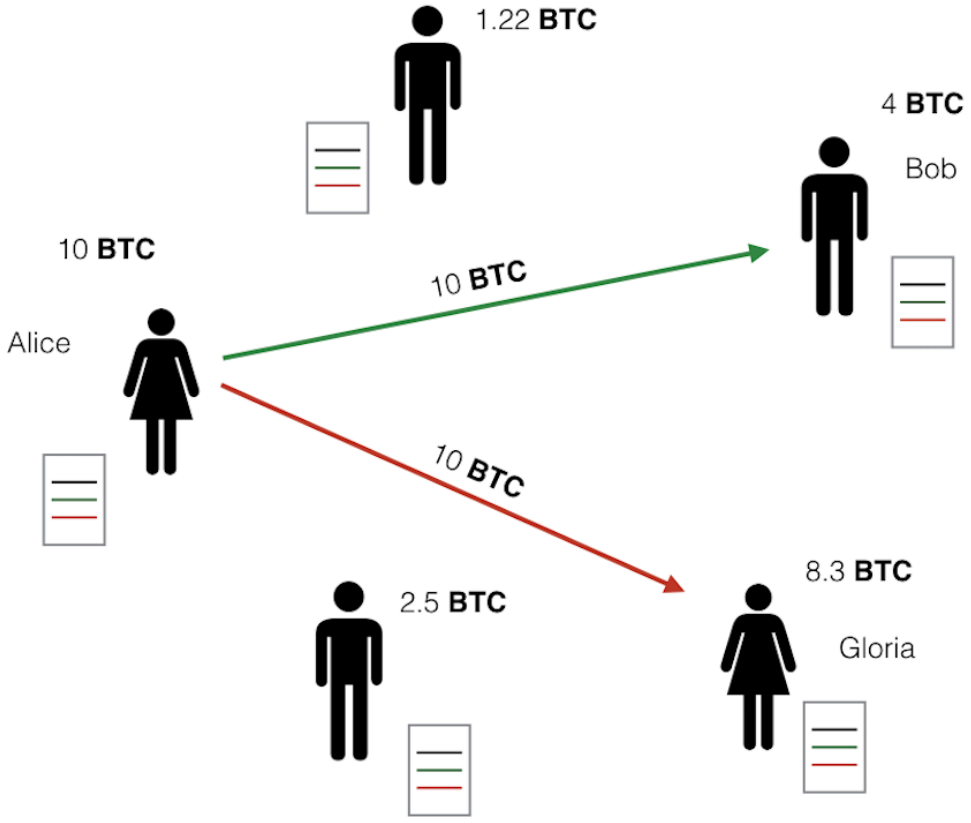
In a centralized environment there is one entity that controls data integrity and guarantees accuracy. All the users must trust their relationship with such central entity. Typical examples in our day to day life are represented by banks, insurance companies, public administrations, service providers etc On the other

hand, in decentralized environments the main objective is to operate without a trusted central authority: the system is built around a network of anonymous peers who do not know each other, neither they trust each other (trustless environment). In this type of architecture the trust is needed in order to know the identity of the originator of a transaction to avoid that a malicious node can generate fake transaction and, of course, to make sure that the sender of a transaction actually has all the assets needed to complete the transaction itself. The first issue can be solved by requiring to include in the transaction the sender identity, and that all transactions must be digitally signed using the sender's private key. In this way everybody can verify that the transaction has been generated by the sender using his public key. Most cryptocurrencies on DLT do not keep balance information. Instead the information about the available amount is calculated going back in time following all the transactions that had an influence on the balance.

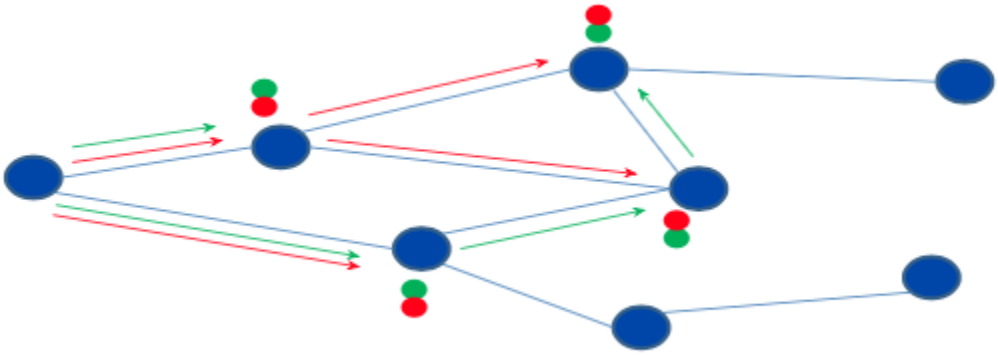
2.1.1 Distributed ledger types

Distributed ledgers can be public (also called permissionless, meaning that everyone can join the community and become part of the network) or private (permissioned, where the possibility to join the network is subject to approval by the ledger's owner). They also vary in their structure and size. Permissionless blockchains require computer processing power to confirm transactions. Distributed ledgers can be composed by anonymous users (nobody trusts anybody) and in this case it is called permissionless blockchain or identified users (users can trust each other) and in this case it is called permissioned blockchain. In the first case each user has a copy of the ledger and participate in confirming transaction independently, while in the second case a permission is required for users to have a copy of the ledger and participate in confirming transactions.

2.1.2 Double spending problem

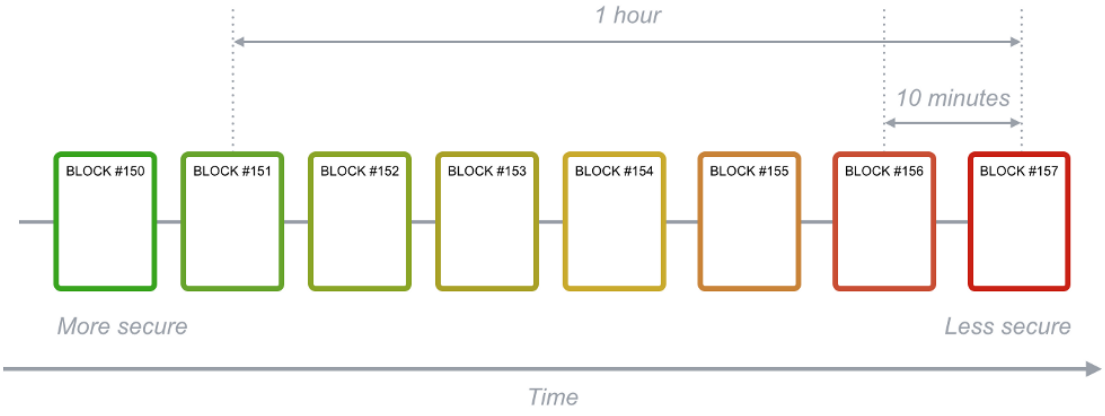


In this type of architectures one of the most known issues to avoid is the "double spending" problem: a user can spend money only if he has not yet spent it. It is possible that a malicious user tries to generate a transaction to send money to a different user and then immediately after generates another transaction to send back to himself the same amount of money. This is not an easy problem to solve, and even if you can think of adding a timestamp this can be easily manipulated. The absolute timestamp may be needed for some applications. For financial transactions the order of arrival of transactions is used to establish priorities. Ordering transactions is not easy because while transactions propagate over the network, they may reach different nodes in different order.



Ordering transactions is needed on a distributed ledger to reach a common consensus on the network. Essentially, a ledger is an ordered list of legitimate transactions (not necessarily of financial type). In a ledger the rules used to reach consensus on the legitimacy and the order of the transactions must be clearly defined, and all the nodes will update their copy of the ledger according to such decisions.

2.2 The blockchain



The Blockchain represents one of the first solution to the distributed consensus problem. Transaction are taken from a common pool (sometimes called the mempool) and grouped in blocks. The blocks are used to order transactions (all the blocks are linked to each other to represent time ordering). In common applications, a transaction represents and tracks the change of ownership. Transactions too are linked to each other: more specifically a transaction's output specifies a spent amount, while a transaction's input specifies an amount earned. When sending money from A to B the output of A is connected to the input of B.

2.2.1 Distributed consensus

As soon as a block is added to the blockchain, all the nodes participating in the distributed consensus procedure enter a competition to generate the next block. Each node, known as a miner, takes a group of transactions from the mempool and starts to create a block activating a complex procedure to verify the legitimacy of the selected transactions: by following the chain of transactions backwards it is possible to verify the consistency of all the amounts that were spent or earned, therefore verifying the current balance and the validity of the transaction being checked. Once the block is complete with the desired number of transactions, the miner generates the hash of the block content by performing what it is called Proof of work. The first node (miner) succeeding in this completion among all other miners has a next valid block for the blockchain. This block is broadcast to all the other nodes, and each of them will add the block at the head of the chain after verifying the legitimacy of all the included transactions continuing the consensus process.

2.2.2 Block verification and mining

The Proof of Work (Pow) is part of the verification process of a block. It is carried out by running a complex algorithm requiring significant resources. This is the process that is performed by miners to maintain this type of network.

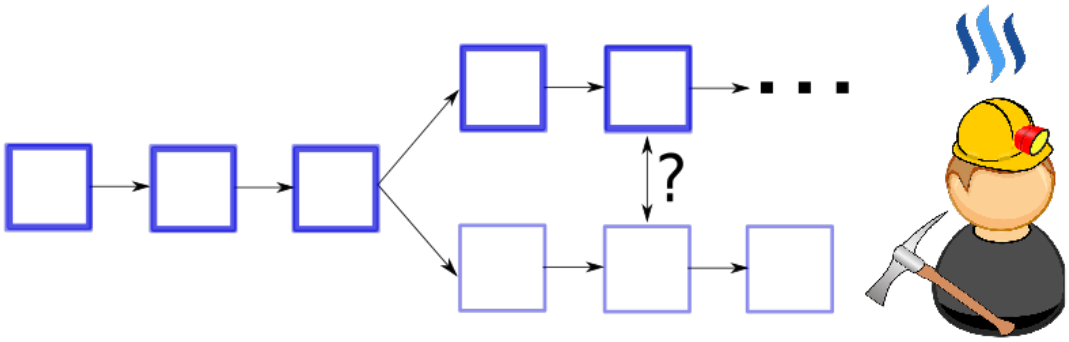
During the mining process, the new block is built including:

- All the (unconfirmed) transactions selected from the mempool;
- The digital signature of the current last block;
- A nonce, a special variable whose value can be changed by the miner in order to solve the cryptographic puzzle described below.

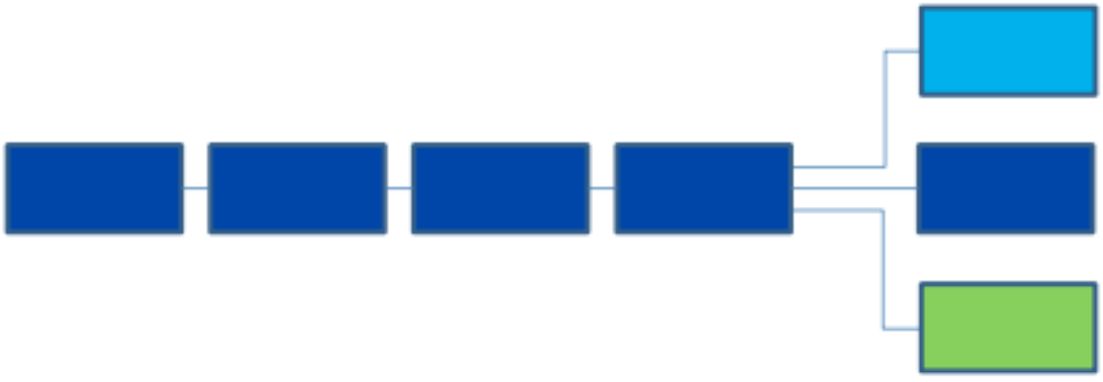
The algorithm consists in repeatedly applying a SHA256 hash function to the whole data set, incrementing the nonce value at every iteration until the resulting output (the hash) is smaller than a predefined value. This procedure is commonly called block mining, and its result consists of the proof of work for that block.

2.2.3 Blockchain branches

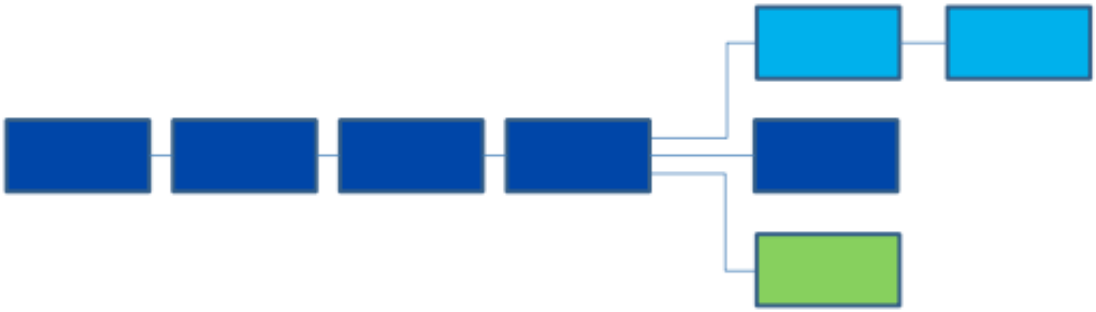
The mining process is completely distributed and executed locally by each miner independently (not synchronized with the other nodes). Because of this, it is possible that two nodes come up with a new valid block (containing different transactions) roughly at the same time.



In this scenario several "next blocks" (all valid) can be broadcast to the network, generating one or more branches in the chain. The dark blue chain represents the consolidated chain, while the other 3 blocks represent 3 new (just mined) blocks candidate to continue the chain.



In case of a fork, a miner tries to build the next block on top of the first valid next block received from the network. As soon as one of the chains of the fork becomes longer than the others, that one is selected as the main valid chain In the picture below, if a new block is received for the top branch of the blockchain (even if it is not the first received/generated), the corresponding branch prevails over the others. In this case the light blue chain becomes the most likely candidate to continue the chain.



Once the longest branch of a fork has been identified, all the blocks on the other branches are automatically discarded by the network. All the transactions included in the discarded blocks are sent back to the unconfirmed transactions set (mempool). In this case the green block and the dark blue block are discarded and the chain continues with the 2 light blue blocks.

2.2.4 Miners reward

Carrying out the mining process is extremely expensive in term of computing power and energy. According to the Digiconomist, Bitcoin's annual electric use is approximately 24 TeraWattHours per year (TWh/yr). It is estimated that for a Visa transaction the power used is just 0.01 Kwh, around 37 Kwh for an Ethereum transaction and around 200 Kwh for a Bitcoin transaction. A compensation mech-

anism must be provided so that the miners are encouraged to keep bearing this high cost. Such compensation comes in two forms: the block reward and the transaction fees.

The block reward

In the Bitcoin Blockchain the block reward is an amount of Bitcoin awarded by a miner for the completion of a block. This amount, originally equal to 50 BTC, decreases in time: more or less it is halved every 4 years (today the block reward is 12.5 BTC). The Bitcoins paid as block reward are created from thin air (hence the name of mining). At the genesis of all Bitcoin, a part of those tokens were put aside to be mined as block rewards, and therefore keep the mining process active over time. Those Bitcoins are earned just by the miner who completed a valid block which becomes part of the official blockchain. Since this amount of Bitcoin decreases in time, at some point in the future all miners will have to give up this reward and the mining process will be less appealing. The reward will be equal to the sum of the fees associated with all the transactions included in the mined block.

Transactions' fees

Every user who wants to make a transaction, of any amount, must include a fee to be associated with the transaction. All the transactions' fees, related to the transactions included on the block just mined, are earned by the miner who mined that block. This system will allow to maintain the structure of the blockchain even when the block reward ceases to exist. Depending on the amount of the included fee, transactions can have shorter or longer confirmation time. Sometimes the information is given to the users about the minimum suggested amount to be included as fee in order to see your transaction approved in a reasonable time. Since the miner can choose any transaction to approve from the mempool, it is foreseeable that the transactions with higher fees will be given precedence over all the others because each fee is equivalent to a profit for the miner. The price of the suggested fee follows the economic law of supply and demand: if at a specific time the network has a high number of transactions to approve, the average of associated fees will be high, otherwise if the number of transactions per time unit is low, also the average fee will be lower. The rewarding policy can have an impact on the overall operation of the mechanism, also with some risks. If mining becomes a business, larger and larger miners will show up aiming at getting a larger slice of the pie, creating new concentration of power and leaving out smaller miners. If the reward is too small, miners would leave the system. If the reward is high enough, more miners would join (interestingly, the equilibrium in case of zero cost for joining and leaving happens when the miners' profit is 0).

2.2.5 Final considerations

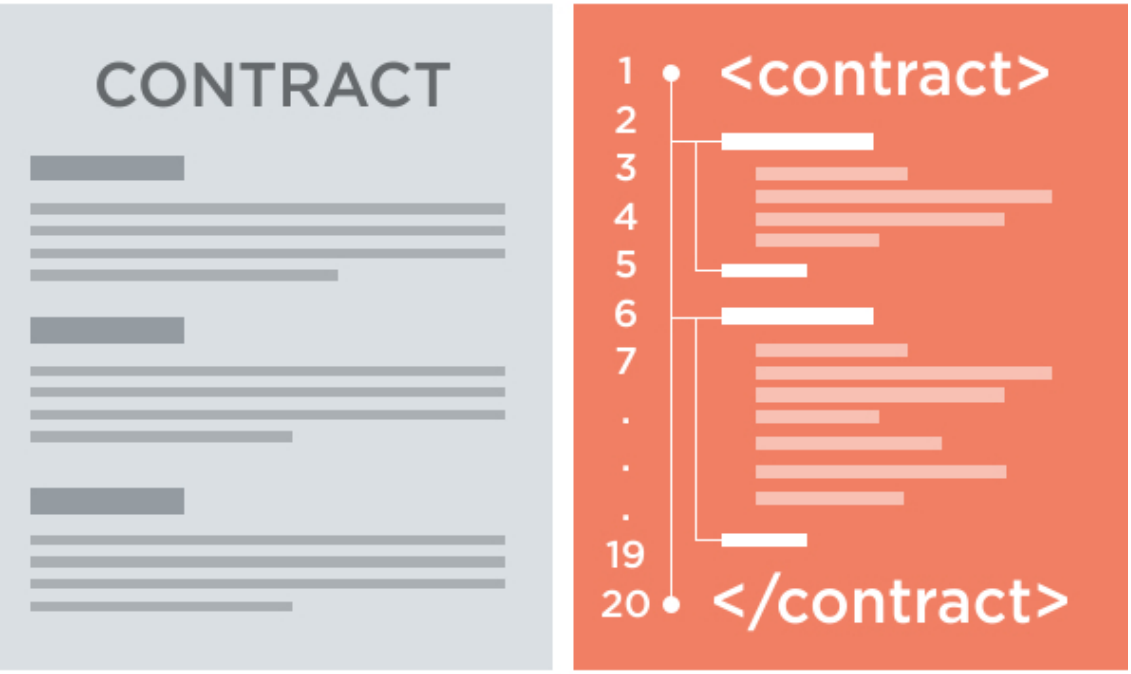
Coming back to the double spending problem it is impossible that a node can be able to generate a legal transaction from A to B and then a transaction to himself to have back the spent tokens. The only possible way to perform this attack is to force a longest branch to bury the second transaction causing the first transaction to become invalid. But pre-calculating a long branch is not feasible because its first node's hash is influenced by the hash of the current "last node" of the chain. And also because calculating a branch cannot start before the rest of the network starts

mining the same "next node", and beating the network requires a huge amount of time and resources.

2.2.6 Non financial application

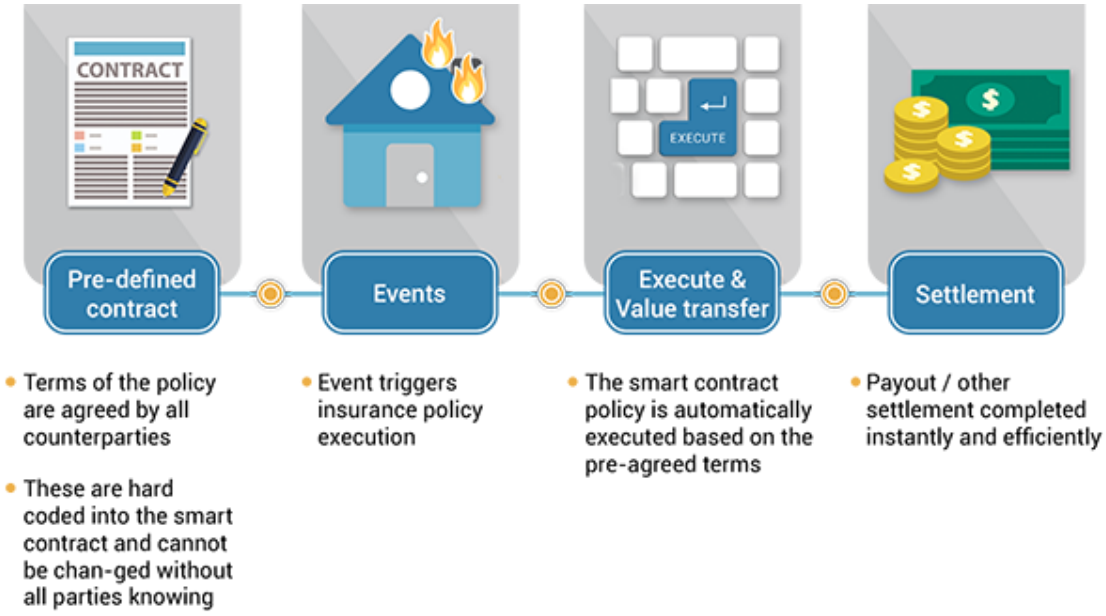
Once the power of the DLT architecture able to store data in secure, a tamper-proof, certified way is understood, it is possible to imagine other useful applications in addition to the most known financial ones. DLT structures can be used to store sensitive data or to create a shared database in order to have data shared among many people in environments without trust.

2.3 Smart contracts



Another interesting application of these DLT structures can derive from the possibility to store on them some code. This type of application are called smart contracts. After the contract is in the blockchain, anyone can use it by sending a transaction to the address of the contract. Every transaction (except the ones used to store the contracts in the ledger) specify a recipient (destination) address. The effect of receiving a transaction for a contract is to execute the code using the data and parameters specified inside the transaction. In order to avoid the risk represented by infinite loops in the code, a contract has a limit to the time or resources (also called "gas") it can use. While the contracts run, it uses gas units up to the limit. If the limit is reached, all the actions are rolled back. Applying the same mechanism of the transactions' fees, miners give precedence to transactions

specifying higher gas unit price and higher maximum gas.



In the picture above a potential use of a smart contract is shown, referring to the use case of management of an insurance agreement. The usual contract signed with an agency guarantees that, if certain conditions apply or specific events occur, the user gets a pre-defined compensation. A contract like this could be stored in a blockchain, making it visible to any party interested in checking all its clauses (a transparency feature not commonly found in traditional contracts). In case some of the events covered by the contract actually happen, the code written in the contract starts execution evaluating conditions and then performing the transaction if the conditions are satisfied.

2.4 Weaknesses of the blockchain

Bitcoin is the most famous DLT architecture and since its introduction in 2009 it has generated great interest for the new features introduced. Nowadays the blockchain is considered as the most famous DLT architecture to be taken as an example in this category and many other technologies are rising trying to emulate its advantages and overcome its drawbacks.

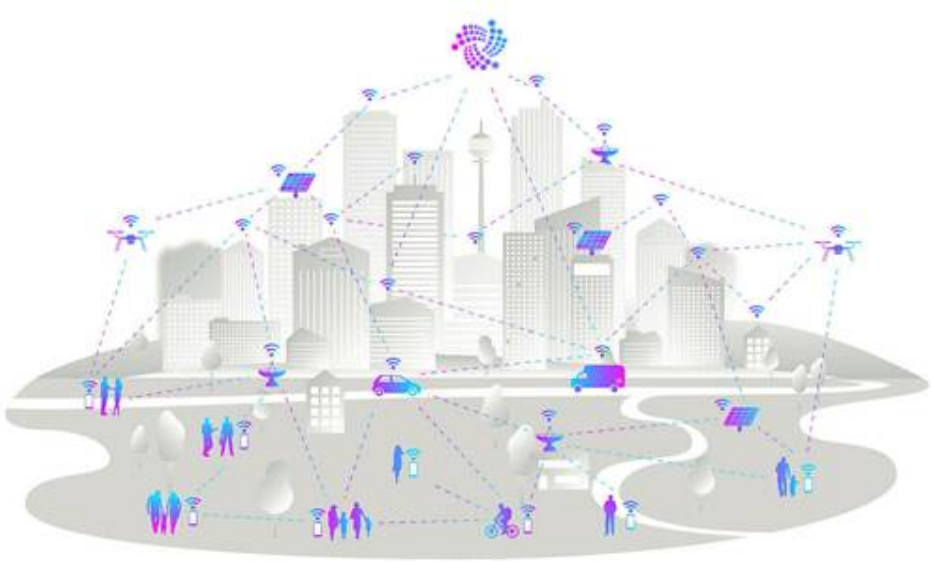
The main advantages of this specific DLT structure have been already covered. The list below includes some of the weakness of the blockchain, which led me to consider a different type of DLT:

- Transaction fees;
- Energy consumption (due to the high work caused by the complexity of the PoW);
- Low scalability (transactions confirmation rate limited by design);

- Not really decentralized: the miners' community actually controls the network operation and performance;

The impact of the above mentioned limits depends on the specific application context and requirements. My thesis is referring IoT (which requires low costs, high scalability and fast transaction confirmation time), for which Blockchain is definitely not the best solution.

2.5 IOTA DLT



Several families of DLTs have been designed and implemented in the past few years (permissioned, permissionless, private, public, etc...). After a starting period where I analyzed DLT basic functionalities and major advantages they can offer, I decided to focus entirely on a DLT architecture called IOTA.

IOTA is:

"A next generation permissionless distributed ledger that utilizes a novel invention, called a Tangle, at its core."

In the next chapter I analyze the IOTA main features, with some references to Blockchain in order to underline the advantages and disadvantages of IOTA compared to the most known DLT architectures.

Chapter 3

IOTA

Due to its innovative architecture IOTA offers a range of unique features:

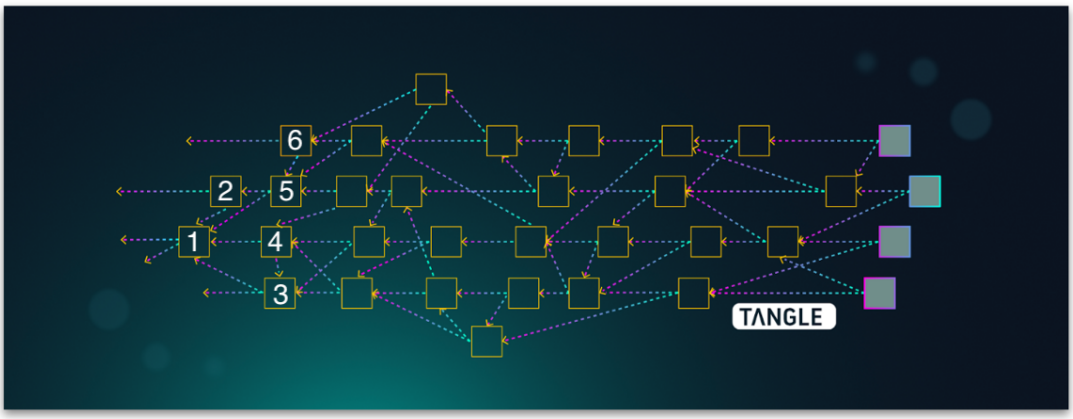
- **Scalability:** IOTA can achieve high transaction throughput thanks to parallelized validation of transactions with no limit as to the number of transactions that can be confirmed in a certain interval;
- **Decentralization:** IOTA has no miners. Every participant in the network who wants to put a transaction on the tangle actively participates in the consensus. This makes IOTA more decentralized than any Blockchain;
- **No Transaction Fees:** IOTA has no transaction fees;
- **Quantum-immunity:** IOTA usesa next generation trinary hash function called Curl-p, which is quantum immune (based on Winternitz one time signatures).

3.1 The Tangle

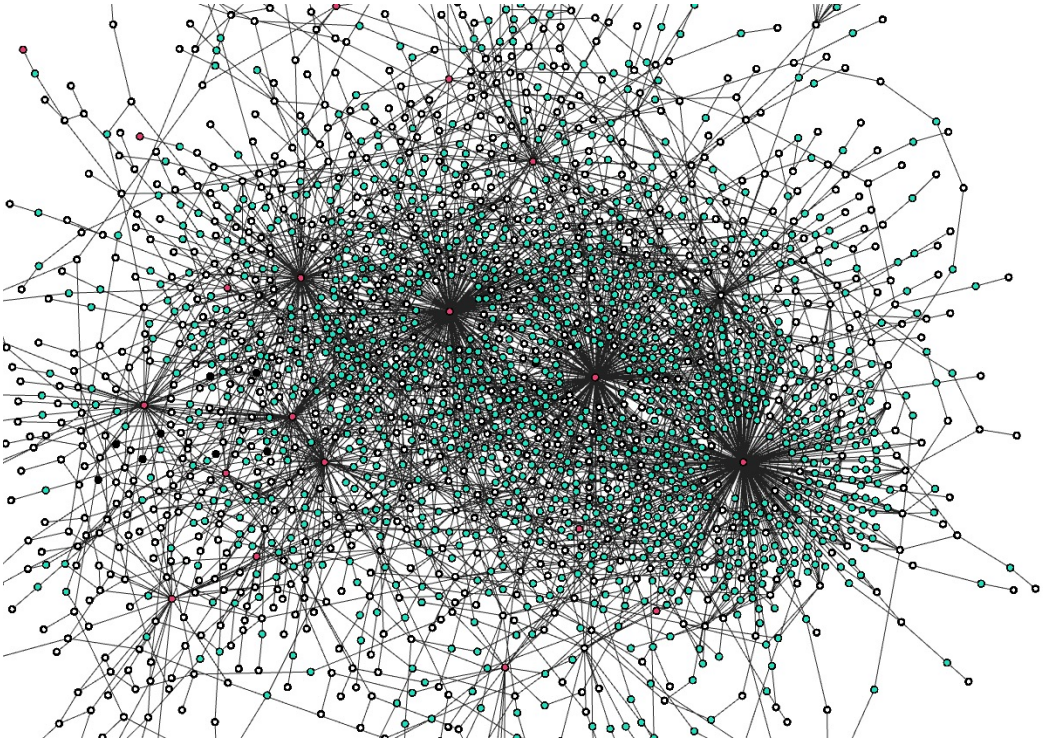
The Distributed Ledger used by IOTA is a data structure based on a Directed Acyclic Graph that the project team decided to call tangle. Sometimes referred to as "a blockchain with no block and no chain", the Tangle has also no miners: every participant has to carry out his/her own (small) proof of work therefore contributing

directly in the consensus procedure. This radically new architecture solves some well known issues shown by BlockChain-based systems and other Distributed Ledger Technologies.

3.1.1 Structure of the tangle



A directed graph is a collection of vertices (squares on the picture above), and edges (arrows that connect squares). The tangle is a particular kind of directed graph in which the vertices are transactions. When a user wants to add a new transaction she chooses 2 transactions already present on the tangle and confirms them by creating 2 new references (which become two arcs going from her new transaction to the two selected ones). In this picture, for example, the transaction number 5 is linked to transactions 2 and 1, (meaning that it confirms those 2 transactions). Unconfirmed transaction are called tips: in the tangle shown in the picture above the 4 gray transaction at the rightmost end are all tips (the last added transactions). While the picture above shows a simple representation of a part of the tangle, the picture below shows a real part of the tangle in the IOTA mainnet displaying how complex this structure can become.



The main motivation for the choice of a DAG structure is scalability: a blockchain has an inherent transaction rate limit related to two fundamental parameters: the block size (which limits the number of transactions that can be included in a block) and the block issuance rate. If blocks are issued too frequently, or are too large, the blockchain will experience frequent forks. When a fork happens, several new blocks are added to the chain at similar times, and the network must eventually decide which ones belong to the actual chain and which ones must be discarded with a rollback action. In a DAG, many forks are accepted and they occur often. Unlike blockchains, a fork is not final. Diverging branches can be merged back together, as long as they are consistent with each other. The transaction rate is therefore bounded only by the nodes' latency. DAG is an alternative ledger architecture seen as a variance of blockchain. With reference to the CAP theorem of distributed systems, this new structure favors Availability over Consistency, which is guaranteed eventually.

The first transaction in the Tangle is referred to as the genesis. All the IOTA tokens were created in the genesis, and no new ones will ever be created. All transactions in the tangle reference the genesis directly or indirectly (see section 3.1.2).

As said before, the last transactions added to the tangle that have not yet been confirmed by other transaction are called tips. The IOTA consensus algorithm is expected to chose tips to approve, rather than older transactions already referenced helping to move the consensus forward through the tangle.

Each transaction contains payment information, in the form of "A paid B X IOTAs". Approving a transaction implies verifying that it does not break the consensus rules: in particular, that none of the accounts have negative balances. It

also checks that the two chosen transactions are not conflicting with each other.

3.1.2 Tips selection

As was said before, the nodes of the DAG represent transactions and the edges represent approvals. 2 types of approvals are possible:

- **Direct approval:** If there is a direct connection from a Tx A to another Tx B (A directly references B);
- **Indirect approval** : Tx A is said to indirectly confirm Tx B if there is a multihop path along the graph from A to B.

The IOTA network is composed by nodes connected in pairs using a P2P protocol through which they continuously interact each other. Each node is capable of storing a portion or the whole tangle, and it keeps in sync with all its peers. Each node represents an entry point to interact with the tangle, and offers features such as the selection of the two tips to be confirmed by a new Tx that is being sent to the tangle. Node are designed to follow a recommended tip selection algorithm. Tip selection is done by performing a weighted random walk from the genesis towards the tips. The walk stops when it reaches a tip. The walk is performed twice, and so two tips are chosen. This tip selection algorithm is designed to not favour transactions that have been longer in the DAG,therefore creating an incentive to select and confirm newer transactions.

3.1.3 Consensus in the Tangle

As a payment network, IOTA provides a method to know when a transaction can be considered, without error, as confirmed. When a Tx is considered confirmed it is accepted by the public consensus and cannot be removed from it since other Txs refers to it and it is considered a solid part of the ledger.

Currently there are two approaches to establish consensus in the tangle: the distributed approach suggested in the whitepaper and the current (temporary), centralized procedure orchestrated by a coordinator.

The coordinator

The coordinator is an entity controlled by the IOTA Foundation, which issues a zero-valued transaction (called a milestone) every two minutes.. Referring to this method, the definition of consensus is very simple: any transaction referenced by a milestone is confirmed, and the others are not. Since IOTA structure it is completely based on decentralization, this method cannot be a long term solution, because it is centralized. This is a temporary solution introduced to allow the Tangle to grow in a controlled and secure way. When the tangle is large enough it will be able to self-maintain, and the coordinator will be turned off.

Markov chain Monte Carlo (MCMC)

In the IOTA whitepaper a probabilistic and distributed approach is described to guarantee the consensus of a transaction approval, based on the MCMC algorithm. This is similar to Bitcoin and other distributed ledgers, where at any given time a transaction has a confirmation confidence, which is an indication of its acceptance level. In order to know the confirmation confidence for a particular transaction, the tip selection algorithm is performed 100 times. Then it is computed how many of the 100 selected tips reference the considered transaction. If it is referenced by 60 tips out of 100, for example, the transaction is considered confirm with a confidence of 60%. Currently, in the IOTA network, if a transaction has not been confirmed in 30 minutes, the probability that it will be confirmed in the future drastically decreases. If this happens, It is necessary to rebroadcast the transaction or make a new one with the same data.

3.1.4 Proof of Work

IOTA uses Proof of Work to protect from spam. This PoW is similar (but much simpler) to the Hashcash[1] mechanism used for the same purpose. PoW is a short computational operation which consists in finding a number of trailing zeroes (in trytes) in a transaction hash. The number of trailing zeroes is called Min Weight Magnitude (MWM) and is directly related to the difficulty of the Proof of Work. The MWM is proportional to the Pow difficulty and every extra zero to be found increases the difficulty by 3 times. A device that does the PoW will bruteforce the transaction hash (changing a dedicated field called "nonce" at every iteration) until a hash with the required number of trailing zeroes is found. For the IOTA Mainnet the MWM parameter is equal to 14, while the value for the testnet is 9.

3.2 The IOTA token

The IOTA architecture has an associated "crypto-token" that has the same name (IOTA). This token is built upon the Tangle technology.. The token operates on a permission-less Tangle network known as the Mainnet. Everyone is able to interact with this network by operating a personal full node that is peered with neighbours manually added or by relying on a node operated by a third party. All IOTA coins which will ever exist have been created with the genesis transaction. This means that the total supply of IOTA will always stay the same (no mining allowed). The total supply of IOTA tokens is:

$$\frac{(3^{33}-1)}{2} = 2779530283277761$$

[1] Adam Back. Hashcash, May 1997. Published at http://www.cypherspace.org/hashcash/.

Since the IOTA architecture was thought to support IoT, it is specifically designed to collect data at a high throughput: the high supply of the associated token makes it optimal to support a high number of very small transactions while still keeping efficiency. IOTA use the SI notation to write in a convenient way a large number of tokens. In all the trading sites IOTA is referred as MIOTA (Mega IOTA) referring to 1,000,000 IOTA.

3.3 Seeds, Private Keys, Accounts

An account in the IOTA system is identified with a seed. The seed is a number represented using 81 trytes, and it is the unique access key to the proper account and to the corresponding funds. The seed has to be securely stored and never shared. A seed isn't associated to any information about its owner. Its property is based on ownership: if someone has a seed, he or she can access to the token associated to that seed. IOTA allows to choose among 3 security levels. Each security level is associated to an average number of rounds for hashing. This means that a single seed can correspond to 3 different accounts. The client libraries make it possible to easily switch and choose a security level. A user having access to a seed can generate a private key, and from that an address. Incrementing a parameter called key index it is possible to generate an almost infinite number of private keys and addresses from the same seed. To guarantee security, all sensitive functions are implemented client side and thanks to this it is possible to generate private keys and addresses securely in the browser or on an offline device. IOTA uses Winternitz one-time signatures: it is strongly advised to not reuse a previously used private keys. Continuous reuse of private keys can disclose part of the private key and ultimately the seed, having access to the associated founds.

3.4 Anatomy of a Transaction

A transaction in IOTA consists of 2673 trytes (if encoded). When you decode the trytes you get a transaction object which has the following structure:

Field	Type	Description	Length
signatureMessageFragment	String	In case there is a spent input, the signature of the private key is stored here. If no signature is required, it is empty (all 9's) and can be used for storing the message value when making a transfer. More to that later.	**2187**
address	String	In case this is an output, then this is the address of the recipient. In case it is an input, then it is the address of the input which is used to send the tokens from (i.e. address generated from the private key)	**81**
value	Int	Transaction value	**27**
obsoleteTag	String	User-defined tag (removed soon)	**27**
timestamp	Int	Timestamp (not-enforced, can be arbitrary)	**9**
currentIndex	Int	Transaction index in Bundle	**9**
lastIndex	Int	Index of last Transaction in Bundle	**9**
bundle	String	Hash of the current Bundle, which is used for grouping transactions of the bundle together. With the bundle hash you can identify transactions which were in the same bundle.	**81**

branchTransaction	String	Transaction being approved	**81**
trunkTransaction	String	Transaction being approved	**81**
attachmentTag	String	User-defined tag	**27**
attachmentTimestamp	Int	Timestamp after POW	**9**
attachmentTimestampLowerBound	Int	Lower Bound of timestamp	**9**
attachmentTimestampUpperBound	Int	Upper Bound of timestamp	**9**
nonce	String	The nonce is required for the transaction to be accepted by the network. It is generated by doing Proof of Work (either in IRI via the attachToTangle API call, or an external library).	**27**

3.4.1 Bundle

IOTA uses an account-like scheme. This means that we have inputs (addresses) which you have to spend in order to transfer tokens. Addresses are generated from private keys, which in turn are derived from a tryte-encoded seed. A transfer in IOTA is a bundle consisting of outputs and inputs. Bundles are atomic transfers, meaning that either all transactions inside the bundle will be accepted by the network, or none.

Index	Purpose	Value
0	The Output/Desintation of the bundle. Recipient of the transaction	Greater than 0
1	First bundle entry that spends the entirety of the address input. This bundle entry also contains the first part of the signature (in the example case, it'll be the first half of Alice's signature)	Less than 0
2	Second half of Alice's signature.	0
3	Remainder Output. If there is a remainder (Alice didn't spend her entire balance at the respective key index), it will be sent to a remainder address.	Input minus Output

The construction of bundles is done by client software. To manually create a bundle it is necessary to retrieve all the available input addresses (all the addresses created from the seed with a value greater than 0 of IOTA coins) for the considered seed and then choose a set or all of those addresses to satisfy the requested amount of IOTA for that transaction. If the sum of IOTA of the provided input addresses is larger than the requested amount of IOTA, it is necessary to provide an address to which all the unused IOTA will be sent. Inside the bundle there is the transaction moving the requested amount from input to the recipient address (the one where we want to sent IOTA) and then a set of transaction to set the balance 0 from the previously used input address. It must be noticed that all used input address should not be used twice for security reason. All of those operation can be performed in an autonomous way (some function performs everything in background) or manually, so an algorithm must be implemented to store all used inputs and provide correct inputs when needed.

3.4.2 Performing a transaction

In IOTA there are no miners, and the process to build a valid transaction is different from the Blockchain out there today. Here are the steps needed to perform a transaction :

1. "Signing": sign the transaction inputs with a generated private keys (this can be done offline);
2. "Tip Selection": the MCMC tip selection algorithm is used to randomly select two tips, which will be referenced by the new transaction. The 2 selected transaction are called "branchTransaction" and "trunkTransaction";
3. "Proof of Work": in order to have your transaction accepted by the network, you need to do some Proof of Work - similar to Hashcash (spam and sybil-resistance). This usually takes around 30 seconds on a modern PC.

After these steps, the 2 tips selected and the "nonce" leading to a hash which satisfies the MWM are written in the transaction object. At this point this object is ready to be sent to the network. Then it will be further confirmed when other transaction reference it, directly or indirectly.

3.5 IOTA nodes

When participating in the IOTA network it is possible to choose to operate a light node or a full node. With the GUI wallet (available on the official IOTA GitHub repository) it is possible to choose to run it as a light wallet or as a full wallet (corresponding to a light or full node).The information provided so far shows that in IOTA it is possible to identify 2 abstraction layers. The first is represented by the P2P network built around the IOTA full nodes, and the second represented by the Tangle. Every full nodes keeps a copy of a portion or of the complete tangle. Full nodes continuously synchronize with their peers exchanging information about the last tips, and they broadcast the transactions generated by light nodes application (like the light wallet wallet) that are linked to full nodes. A full node is needed to interfacing with the tangle but it is not necessary for a user to run his own full node.

3.5.1 Light-nodes

A light node is a simple application that relies on a IOTA full node to carry out the most complex functions needed to attach transactions to the tangle.. The most convenient (but complex) way is to run and manually manage a full node and then use other applications, such as the IOTA GUI wallet, referring to that full node. In this way it is possible to have the complete control of the process and it is the best way to understand the complete process to interface with the tangle. It should be noted that all sensitive functions (such as hashing and signing) are performed client side. This means that the use of a light wallet (even if provided by a third party)

guarantees anyway that the seed or the private keys never leave the local device Not even the light wallet provider can access the IOTA funds

3.5.2 Full-nodes and IOTA Reference Implementation (IRI)

A full node can be runned with the IOTA Reference Implementation, a java application that stores locally its own copy of the tangle and synchronizes with other full nodes added as neighbors. This application can be retrieved on the IOTA GitHub page and can be run in background to start the synchronization phase. This is a full featured IOTA node with a JSON-REST HTTP interface. It allows users to become part of the IOTA network as both a transaction relay and network information provider through the easy-to-use API. The IRI has a lot of parameters that can be set in a initialization file (iota.ini) passed as input parameter to the java process. In this file it is possible to specify port numbers dedicated to the APIs or udp/tcp connections to other neighbors. Among the parameters it is also possible to include a static list of neighbors. The IRI offers the possibility to build other modules called ixi modules that support extended features. Among the existing IXI modules one is dedicated to making the neighbor discovery procedure automatic or to checking the status of the current neighbors (for example CarrIOTA is one of the most used for the autodiscovery). All those modules are welcome from the IOTA Foundation but are not part of the official provided code and are in beta version.

3.6 IOTA Client Libraries

Currently there are some IOTA client libraries available to set up an interface with the tangle and many others are a "work in progress" or in a beta version. With Client libraries it is possible to read and write from and to the tangle. One of the most used (and stable) libraries is the Javascript library called iota.lib.js. A python library and a Java library are also available. For my thesis project I decided to use the JS library because it implements both the official API and some newly proposed functionality (such as signing, bundles, utilities and conversion). The iota.lib.js offers the possibility to use convenient functions to send a transaction calling just one wrapping method, but also to manually write the entire process to create a transaction by manually calling all the needed methods. In this way it is possible to customize data inside a transaction or to perform a zero-value transaction and to choose how to broadcast it and when to rebroadcast the transaction if it doesn't get confirmed within the desired time.

Chapter 4

IOTA Wallet Bot

After spending a few weeks studying the IOTA architecture and the main functionalities of the tangle, my thesis' project concentrated on a specific application: the design and development of a Telegram Bot able to manage an IOTA wallet for each user. This bot allows to send and receive IOTA tokens easily directly from inside a chat. I decided to develop this application using the official IOTA Javascript library to exploit all the potentials of nodeJS and npm packets manager. When I started thinking about the the bot features I imagined 4 different use cases:

1. Private chat between a user and the bot: to send/receive IOTA the user needs to manually copy/paste the required addresses;

2. Enhance the function mentioned at the previous item permitting to send IOTA to a username. The bot will manage all the process in the background avoiding the manual handling of addresses;

3. Open multi-user wallet for users belonging to a Telegram group: each user belonging to this group can perform any operation and the corresponding transactions will remain on the chat history showing who has performed it;

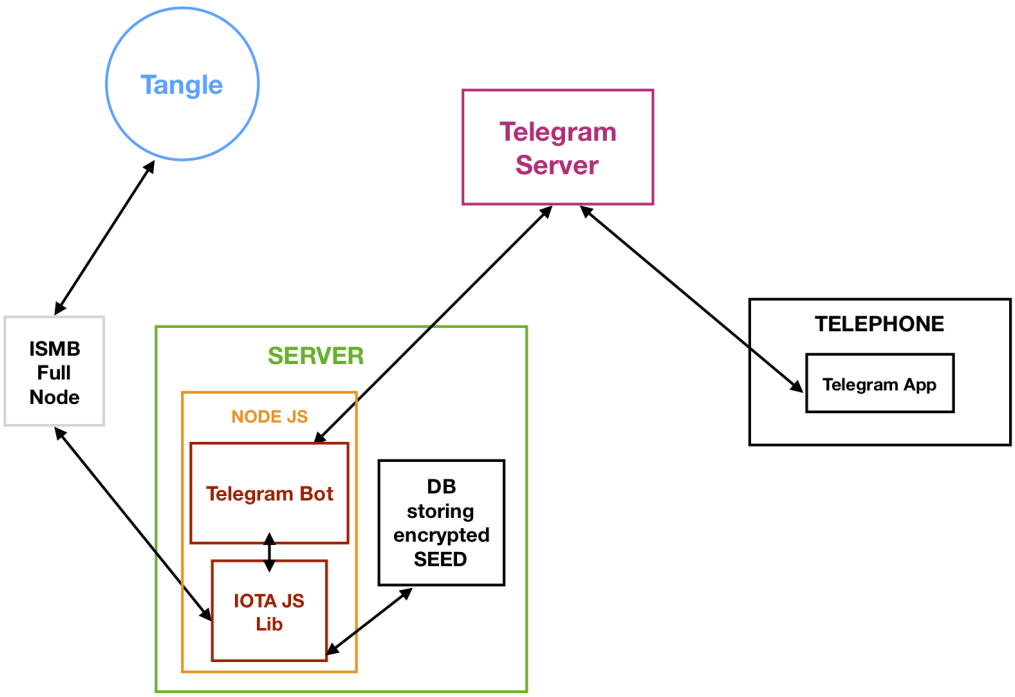
4. Restricted multi-users wallet for a group of user on the same Telegram group: all users must authorize each operation inside this group.

For my Master's Degree Thesis project I focused and analyzed all the aspects referring to use case n°1. The other use cases will be analyzed and developed in future releases. The goal of this bot is to facilitate the exchange of IOTA without having to use a dedicated wallet application. The Bot isn't intended to store and handle a large amount of IOTA to avoid unpleasant situations.

Particular attention was dedicated to the bot usability: starting using this Bot just requires to open a new chat with the user "@IOTAWalletBot". The bot is currently in beta version and for this reason I do not consider myself accountable for the misuse and/or malfunctions caused by the use of libraries under development - even if in stable versions.

All the details about this Bot project it is available on the IOTA Ecosystem site (https://ecosystem.iota.org/projects/iotawalletbot) and all the code is released as open source code on GitHub (https://github.com/france193/IOTAWalletBot).

4.1 Project's architecture



The picture above represents the architecture with all the parties involved in my project. Five different elements can be identified, depicted each one with a different color. The core is the server, which includes the sw needed to interact with the Telegram server and with the IOTA full node(s). The server runs an application developed in Javascript with nodeJS and using the node packet manager (npm) it imports all the required libraries to interact with Telegram and with IOTA.

4.1.1 Telegram Bot front-end

This bot is designed to work with telegram, so the front end application (Telegram chat client) cannot be changed. This can be a big advantage because you can reach many devices with different OS without doing any work, because Telegram is available on Windows phones, Android devices, iOS devices and has a web browser version (compatible with the most common browsers) as well as some "app" versions available for Mac OS and Windows and Linux. The only (major) drawback is that it is not possible to implement any function in the front-end: all the logic has to be implemented on the server and it is not possible to modify the Telegram application.

4.1.2 Telegram Bot back-end

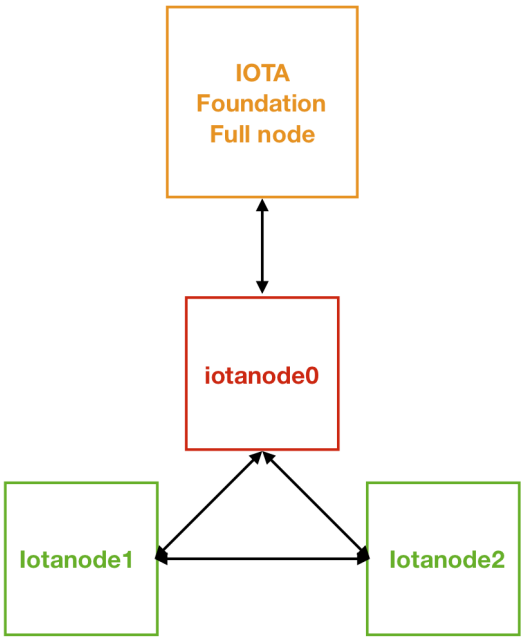
The concept of a Telegram bot is really simple: it is considered as an account people can chat with, but without a real user behind. You must then plan and design all the expected commands a real user may send to the bot, and the corresponding answers. A command is a string starting with a "/" with the mnemonic name describing the function it is intended to perform (for example "/send_hello_world"). When the user sends something to the bot, all messages pass through the Telegram server using a 100% secure and encrypted channel. The Telegram bot application receives all the commands from the Telegram server using the same mechanism (100% secure and encrypted channel). There are 2 possible paradigms the bot application can use to receive commands: polling mode or webhook. The polling method consists in continuously attempting to check (every few milliseconds) if there are messages sent by the telegram server. This method is easy to implement and fast, but it uses a lot of communication resources and power. It is also possible that, in case the internet connectivity is lost, the bot goes into an inconsistent state possibly requiring manual intervention. The second method is more efficient and resistant to network errors. It consists in creating a server on the Telegram Bot application and setup this information on the Telegram server. When an update is available for the Bot, the Telegram server knows where to send them. For this method it is recommended for the Telegram Bot server to have a static IP address and a SSL certificate. The certificate can be self-signed since its only purpose is the creation of a secure channel between Telegram server and the Bot: you can trust yourself and only the Telegram server will use this certificate to contact the bot. I developed my telegram Bot using the webhook paradigm using the "telebot" Javascript library that helps implement Bots and all the responses to the commands the bot needs to respond to.

4.1.3 Interfacing with IOTA and IOTA wallet issues

In order to interact with IOTA directly from the bot I adopted the official library "iota.lib.js". This library implements all the functions necessary to interact with the IOTA full node. Each command made available by the Bot is mapped to a sequence of commands that call the IOTA APIs in order to set up a transaction, build the bundle for that transaction and send it to the Tangle in order to wait for the confirmation. Since my application simply consists in a digital wallet with a Telegram Bot front-end, I started by studying all the mechanisms behind the IOTA GUI wallet made available by the IOTA Foundation. The features offered by a wallet are just a few, but they are very complex because (due to the cryptographic algorithm used by the tangle) the sender address needs to be changed with a new one after every transaction sent with that source address. I had to implement a mechanism in order to store - for each wallet - all the addresses created and used, each one having its own balance. Whenever a certain amount of IOTA needs to be sent, I had to select some of those balances so that their cumulative value is greater or equal to the amount to be sent. In case it is not possible to put together exactly the required amount (which happens in the majority of real cases) the remainder amount must be stored (sent) to just another new address. Only at this point the transactions' bundle can be created, and all addresses used as inputs for that transaction must be marked, on my storing system, as already used. It is also necessary to check the confirmation state of the transaction: only when it is confirmed the user can be sure that the tangle balance for that account is consistent. A potential risk of "double spending" occurs if another transaction is generated before the approval of the previous one. The IOTA algorithms would not allow double spending anyway, so one of the two (conflicting) transactions would remain forever "pending". What is not taken into account by the IOTA algorithm in the above described situation is the possibility that source addresses are used more than once, leading to a potential security breach due to possible detection of the account seed or private key (this risk is a consequence of the use of Winternitz one time signature).

4.1.4 IOTA full nodes architecture

As said before, the interface with the Tangle is always demanded to a full node capable to perform the heavy computations associated with the PoW. When using the client library it is possible to create a transactions bundle and send it to a full node to do the rest of the tasks needed to attach the transaction to the Tangle. I have developed a set of scripts in order to set up and manually manage 3 different full nodes, currently hosted in ISMB. The management of full nodes was quite a complex task because the nodes require frequent maintenance and checks. Currently full nodes do not have access to an auto discovery feature to find neighbors so it is necessary to interact frequently with the IOTA community and manually exchange IP addresses. This task is even more complicated by the fact that not all the community members know how to perform some of the delicate tasks needed to manage their own full nodes. After a long research I finally managed to create a connection with a very stable and high performing IOTA Foundation's node.



I attached the iotanode0 directly to the IOTA foundation node ensuring fast synchronization and efficiency. On this node I limited the maximum number of APIs requests in order to avoid to overload that node: in other words this node cannot be used as a full interface to the Tangle but it only synchronizes the Tangle structures with its neighbors. Then I set up 2 other full nodes (IRI applications): iotanode1 and iotanode2. These 2 are linked together and with the previous node in order to have them easily synchronized with iotanode0. On these 2 full nodes all the API are available and can be used to interact with the Tangle. Using this configuration the workload is split between the two nodes, and they stay correctly synchronized.

4.2 Managing seeds with security countermeasures

In IOTA every account is identified with a seed, which must be kept secret. For security reason I decided to store all the seeds in encrypted form on the same machine hosting the telegram bot application. In this way the seed does never need to be sent over the network / chat, even if the chat is encrypted. All the encryptions are performed using the SHA256 algorithm. As explained, all the seeds resides on the server but only in encrypted form and are never sent outside the server. All the functions requiring the use of the seeds are executed locally by the Bot server, which acts as a client in the interaction with the IOTA network. At the first interaction with the bot (after sending "/start" command) a seed is created and stored locally on a server DB; the encryption key is sent back to the user. To increase security, a

salt (a random generated string) is appended to the key and the result is hashed; the hash is stored on the server (hashed_key). The encryption key of the seed is never stored in any form on the bot server, and after it is sent back to the user it is erased forever. At each interaction the encryption key is requested to the user; the seed is deciphered using that key and it is used to execute the requested command; then the seed is encrypted again with another random generated key which is sent back to the user and erased from the server (repeating the process described earlier). The key is not stored in any form on the bot, but it is visible on the chat history on the user's device. If that key is lost, it it is not possible in any way to access to the associated wallet, and all the funds on that wallet are lost forever. For this reason it is advised to never delete the bot chat history. To ensure that the seed is correct, whenever the seed is decrypted on the bot server the reverse of the previous operation is performed. The salt is retrieved from the local DB; the salt is appended to the key sent by the user, and the result is hashed. The obtained hash is compared with the value stored in the DB (hashed_key), and if the results match the seed is considered correct, otherwise an error is generated. This is the complete mechanism developed to store in a secure way the encrypted seed.

4.3 Telegram Bot Commands

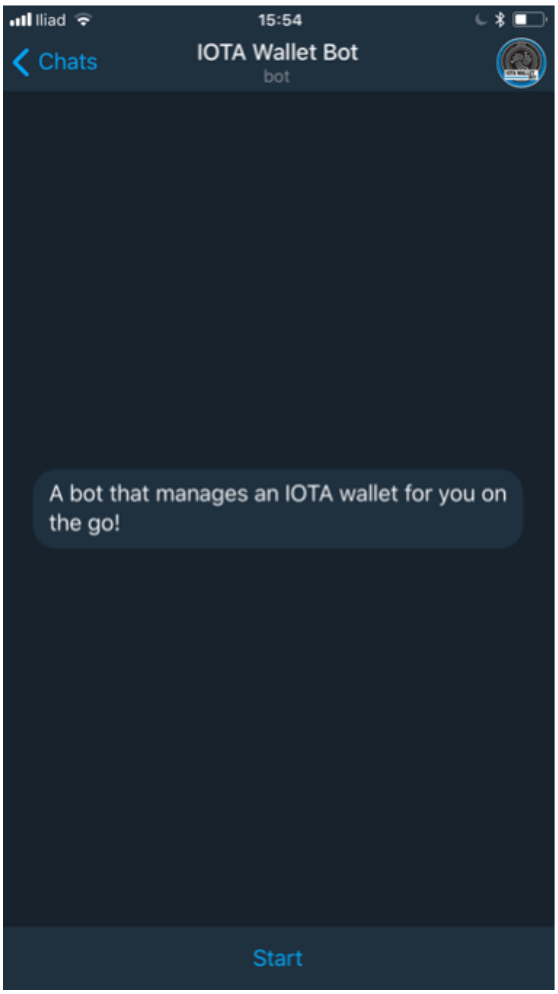
This section describes all the commands I implemented for the Bot, explaining for each one the data flow exchanged among the various entities involved in the process. Since this Bot's main function is to provide a user friendly interface for a an IOTA wallet, these basic actions are needed:

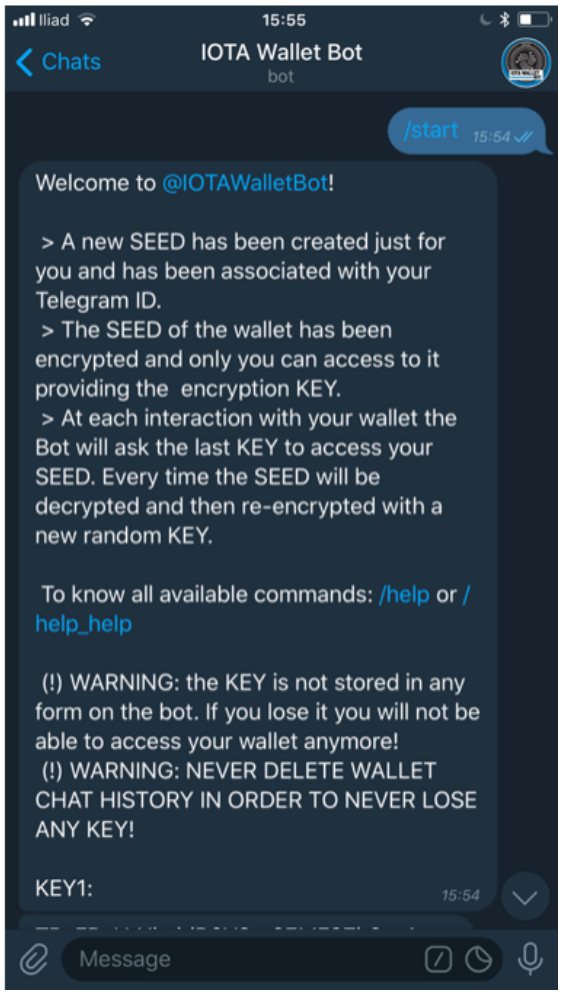
- Create a new account (as said before, in IOTA an account is created with the generation of a new seed, see section 3.3).
- Send a payment.
- Receive a payment (create an address on which to receive a payment).
- Check the available wallet balance.

In addition the Bot has some "general purpose" commands to get the current IOTA token value in USD and to get information about the full node the bot uses as a relay to interact with the Tangle.

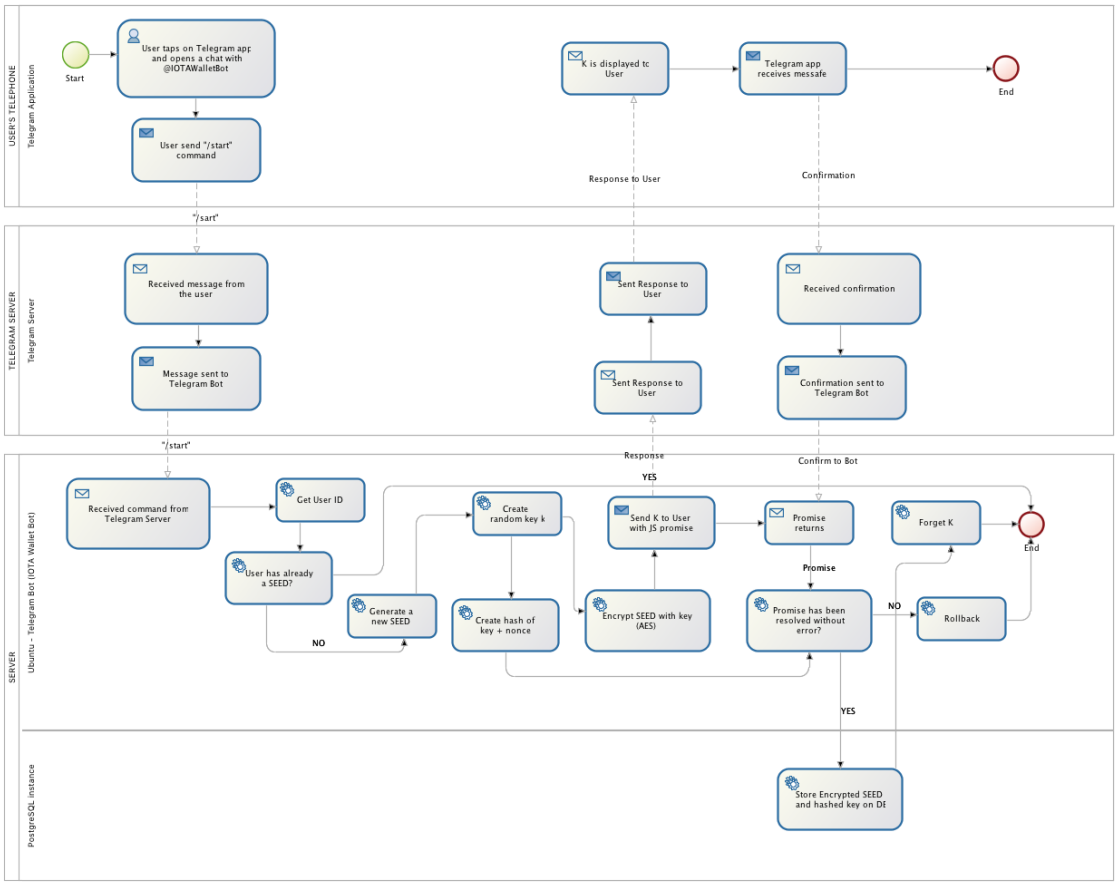
4.3.1 Creation of a new account

The creation of a new account (generation of a new seed) is performed with the "/start" command. This is the starting point of any interaction with the bot as you can guess from the name of the command itself. Before performing any command where the use of the seed is required it is necessary to have a wallet: a check is made in case a user does not have a wallet and will be asked to launch the start command to create a new account. When, for the first time ever, a user opens a chat with a Bot this "/start" command is automatically sent. This command basically lets the bot know that there is a new user. In my case I used this command to create a new wallet for the new user sending back the first encryption key used to store the encrypted seed. The next time that a user wants to perform some actions requiring the seed it is necessary that the user sends this key back to the Bot . At each interaction it is used a different key and the number of the key is stored in order to let the user understand how many operations have been done. To make the user more comfortable, each key is written in the form "#KEY_n". Using the # (hashtag) it is very easy in telegram to come back to the requested n KEY without scrolling all the chat history. Of course the user can use this counter as a security check to know how many times he used his seed. If, between 2 consecutive request, the key number is not consecutive (request n require KEY_n and KEY_m require key m, where m must be equal to n+1) there may be some problem with your seed and your account can be compromised. This number is updated in every command where the seed is needed and not in every commands (when the seed it is not used, the key it is not required).

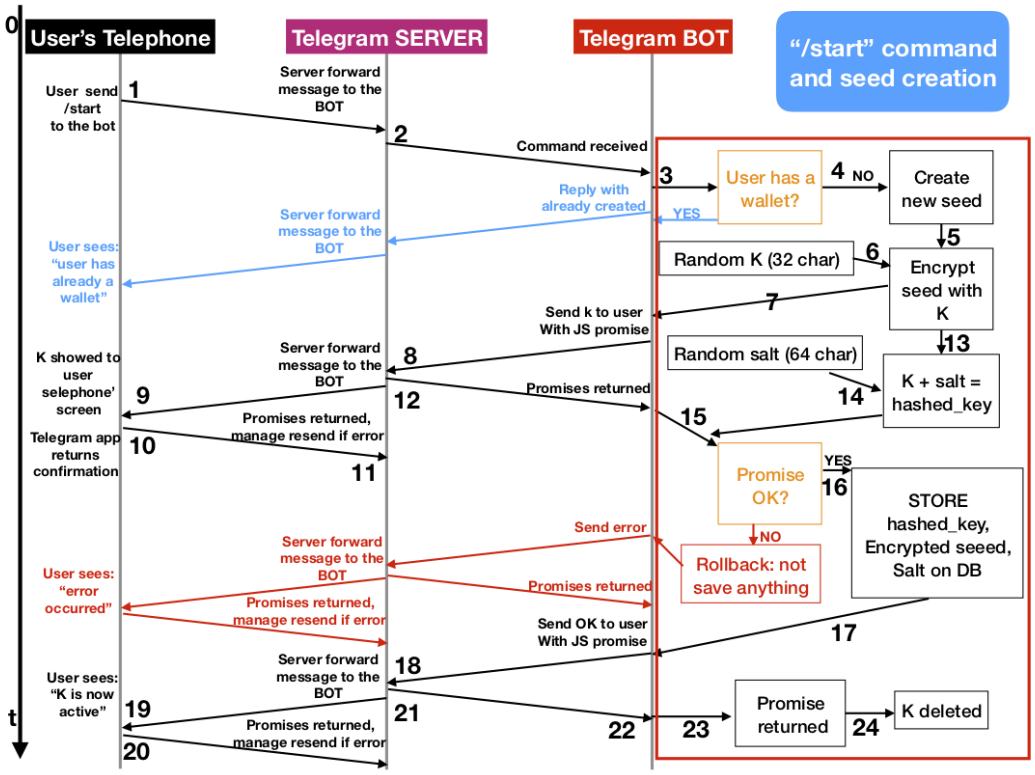

Iliad
15:54
Chats
IOTA Wallet Bot
bot
A bot that manages an IOTA wallet for you on the go!
Start


Iliad
15:55
Chats
IOTA Wallet Bot
bot
/start 15:54
Welcome to @IOTAWalletBot!
> A new SEED has been created just for you and has been associated with your Telegram ID.
> The SEED of the wallet has been encrypted and only you can access to it providing the encryption KEY.
> At each interaction with your wallet the Bot will ask the last KEY to access your SEED. Every time the SEED will be decrypted and then re-encrypted with a new random KEY.
To know all available commands: /help or /help_help
(!) WARNING: the KEY is not stored in any form on the bot. If you lose it you will not be able to access your wallet anymore!
(!) WARNING: NEVER DELETE WALLET CHAT HISTORY IN ORDER TO NEVER LOSE ANY KEY!
KEY1:
15:54
Message

This is the entire process for the "/start" command identifying which entities are involved:



Here there is a more accurate temporal explanation of the process for the "/start" command:



Notice that in this case it is imagined a use case where the user has a smartphone with Telegram mobile application but the procedure is the same in any Telegram client chat (mobile, computer, browser, etc...). Here is all the explanation:

1. The User sends the "/start" command.

2. The command is received from the telegram server (once a message from the user is received from the telegram server, the server will maintain and manage this message to the Bot). If for any reason the message does not arrive to the Telegram Bot, the message will be kept for 24 hours on Telegram server and it will be will be sent again within the next 24 hours. If the Telegram Bot is offline for more than 24 hours this message will be lost. If a message is lost here it is not critical, the user will send another request to the Bot until he receives some answers.

3. The command has been received from telegram Bot, then if the user has already a wallet the flow will follow the light blue arrow answering to the user that has already a wallet and it is not necessary to continue with a new wallet creation.

4. If the user has not got any wallet the process will go on.

5. A new seed is created using the IOTA library.

6. A new randow key K (32 characters long) is created and the seed is encrypted with this key k.

7. This key K is sent to the user and then it will be deleted (the key will be deleted only when the Bot it is sure that the user has received this key). Using telebot library it is possible to exploit all the JS power of Promises. The key it is sent using a promise: a background activity that will return with a success or an error when the request has been performed. In this case when the Telegram server receives this message (with the key K) from the Bot, the promise will be returned with a success status, otherwise an error will be generated. In this way it is possible to check programmatically if a message (that has been sent from the bot) has been received from the Telegram server or not. When the Telegram server has received a message from the bot, the problem is solved because it will manage to send the message to the user in case of error retransmitting until the message is received.

8. The Telegram server has received the message and forwards it to the user. The server also returns a confirmation to the Telegram bot in order to update the Promise status.

9. The User receive the message from the Telegram server and the key K is displayed on the telephone screen by the Telegram mobile application.

10. The Telegram application sends back a confirmation to the Telegram server (otherwise the server assumes that there was an error).

11. The Telegram server receives the confirmation and knows that the message has been displayed on the user's application.

12. At the same time of the point 8 (as described on the point 7) the Telegram server sends a confirmation to the Telegram bot updating the promise status of the request sent by Telegram bot.

13. Once the seed has been encrypted with K, we must store K on the server (not in clear) to be sure that the key that will be sent from the user is correct and deciphers the seed on demand.

14. A new randow salt (64 characters long) is generated and appended to the key K. Then, the result is passed to a hash function to create the hashed_key.

15. The Telegram bot will wait for the answer from the Telegram server to understand if it has received the message with the key K.

16. If the promise returns with an error (the bot is not connected, network error, or any type of error), all the process is aborted and an error message is sent to the user to let him know that all the operation is canceled and if he wants he has to try in another moment. In this case the seed has just been created so nothing is lost but imaging the situation that a user has a seed and he

has already used this seed (so it has some founds on it) and when the seed has been deciphered and a new key has been created to crypt again this seed, before being stored into DB, here it is necessary to roll back all the operation. In this way the user sees that there was an error and nothing changes from the previous encryption key K. In this way it is avoided that a seed is stored encrypted with a new key and the new key is lost (this can cause the loss of the entire account). So, to access to his account he can use the last received key that has been confirmed to be active. (after receiving a key, a user must receive a message that confirms its activation).

17. Of course if the seed has been used and then a transaction has been written on the tangle, this operation cannot be canceled and in this case the rollback is only not to lose the account and the tokens present on it .

18. If the promise returns without any error the process can go on and now it is sure that the user has received the new encryption key. Now, the seed encrypted with the new key, the hashed_key and the previous used salt (appended to the new key before performing the hashed_key) are stored in the local DB.

19. A message indicating that the new key has been activated is sent to the Telegram server using promises.

20. The Telegram server manages this message as described before and forwards it to the user.

21. The User receives this message

22. Confirmation is sent back to the Telegram server.

23. When the server receives the message (at point 18) it sends back a confirmation to the Telegram Bot.

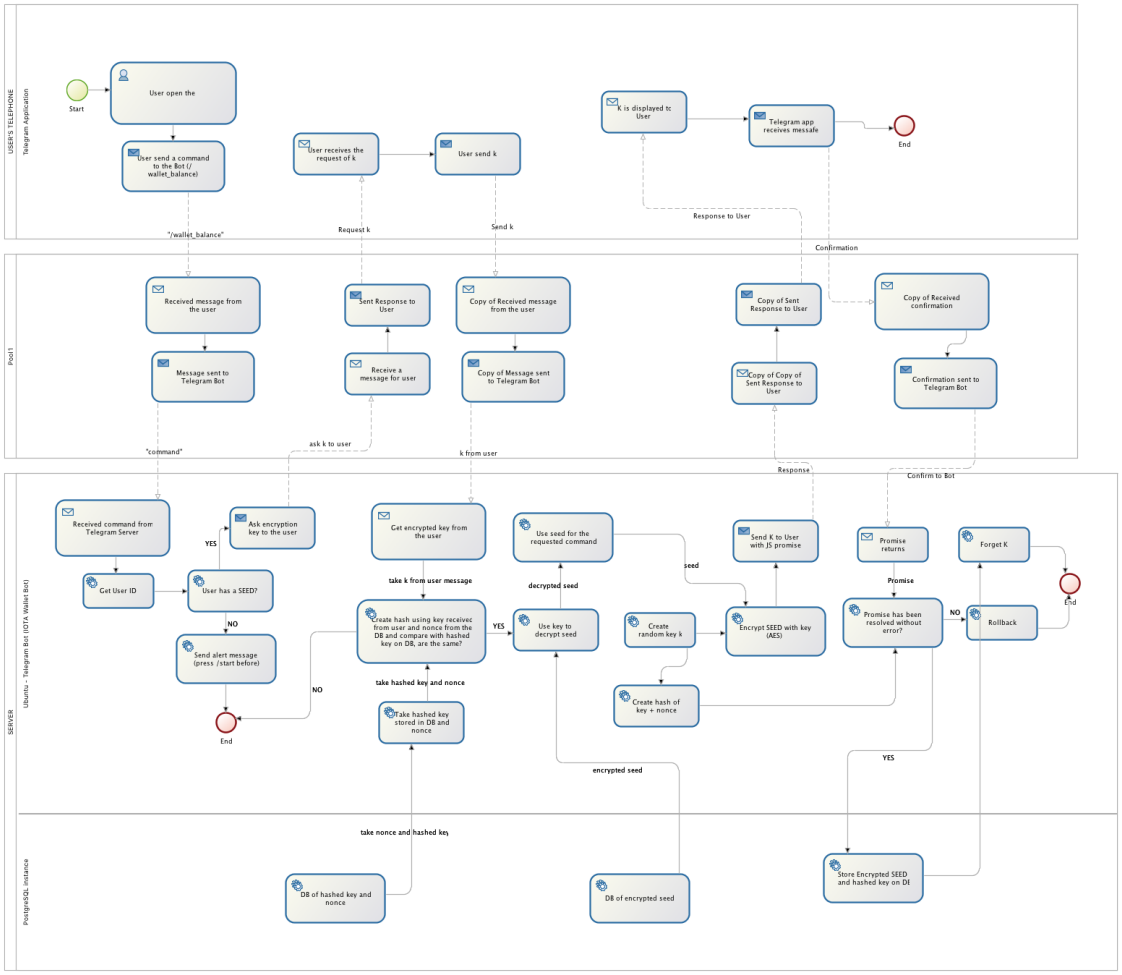
24. The Telegram bot receives the return status of the promise.

25. If the promise status has no errors the process can continue.

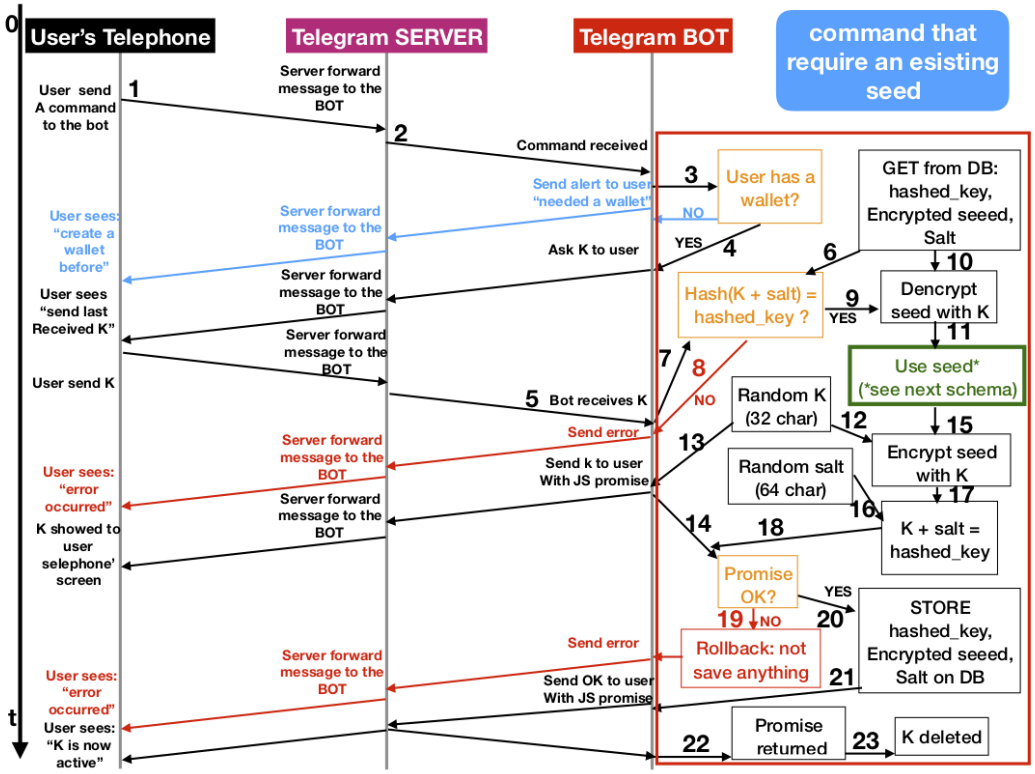
26. The key K, considering the status of the promise, is definitely canceled and not stored in clear anywhere on the server.

4.3.2 Sending payment

For all other commands the process is slightly different, because it is no more necessary to create a new wallet but only to check if the user, that is sending a command, has a wallet or not and then to check if the encryption key is correct and decrypt the seed (checking also if the seed is valid). Finally if all the checks have been passed the bot performs the requested command sending results to the user.



Here there is a more accurate temporal explanation of the entire process for a general command which involves the use of the seed (this is not shown for each command since the behaviour is very similar):



Notice that this scheme has been simplified avoiding to repeat all the same arrows and processes already explained on the previous schema (everytime the Telegram bot sends a command to the Telegram bot the command is sent with a JS promise and the bot will evaluate this result, when ready, managing all the errors as described before). The same error management is automatically done by the Telegram server when sending some messages to the user. When the message arrives to the telegram server (both from the user to the server and from the bot to the server, in practice when the sender is the telegram server the message sending is all managed by Telegram) it is considered "safe and arrived" since all the rest is done by Telegram server taking into account when to send again a message (if the message was not received).

Here it is all the explanation:

1. The User sends a message using the telegram application and this command involves the use of his seed (a payment for example).

2. The Command is received from the telegram server and it is forwarded to the telegram bot application.

3. The Telegram bot checks if the sender has already a wallet, if not it invites to run the proper command ("/start") to create a new wallet, otherwise it takes

into account the command. The blue part is not considered here not to repeat the already described process.

4. If the user has a wallet it is requested the latest received encryption key to decipher his wallet.

5. The user sends the latest received key K (the process it is not described here) to the Telegram Bot.

6. The bot retrieves from the local DB all the user information stored (hashd_key, encrypted seed, salt used inside the hashed_key).

7. The key K is received by the user. the salt is appended to this K and it is performed the hash to the previous result, then this hashed key is compared to the hashed_key previously stored inside DB. If the two hashed keys are the same the key, sent by the user, is valid.

8. If the key is not valid there is some error, the error message is sent to the user (probably the user sent the wrong encryption seed).

9. If the key is valid the process can go on.

10. The encrypted seed can be deciphered and another check is performed (using the iota function to check if the seed is a correct 81 trytes-encoded string).

11. The seed is now used (in this sample the seed is used for perform a payment). See the next schema for more details.

12. A new randow key K* (32 characters long) is created.

13. The new key K* is sent to the user with the same mechanism described before.

14. After the promise returns, if there aren't any errors, the telegram bot knows that the telegram server has received the new key.

15. The seed, previously used, is encrypted with the key K*.

16. A new randow salt (64 characters long) is generated.

17. The salt is appended to the key K*. Then, the result is passed to a hash function to create the hashed_key.

18. If the key is correctly received by the telegram server, the telegram bot knows that the telegram server has received the new key.

19. If there is some errors on the returned promise, an error message is sent to the user, rollback is performed on all the actions (the payment performed in IOTA cannot be canceled because it is not possible to delete/modify what has been written on the tangle, all the rollback actions are referred to the action performed on local DB).

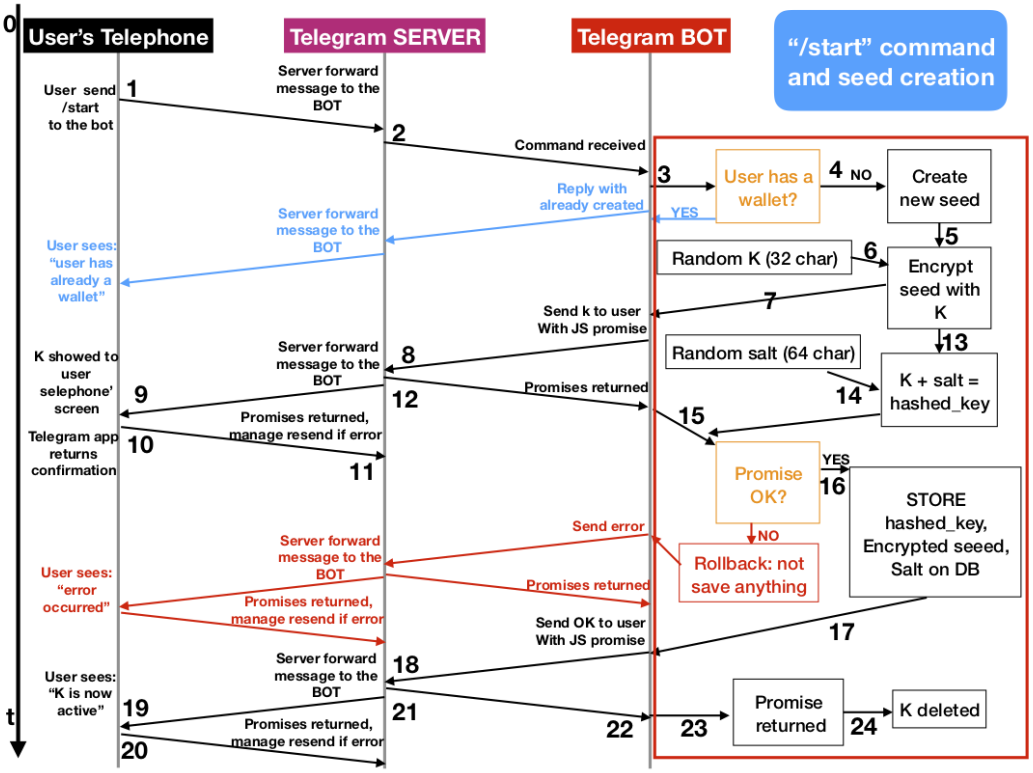
20. Everything goes well, so it is possible to store in the local DB: the new K*, the new encrypted seed, and the salt used inside the hashed key.

21. A message is sent to the user informing that the new key is now active.

22. Telegram server has received the message from telegram bot, so the promise returns correctly.

23. K* is deleted from server.

Here it is reported in detail what happens between Telegram Bot and IOTA when a command (like send payment) is requested from the user. Previously I analyzed the interaction between the user, the server telegram and the bot, now I describe how the interaction with the tangle happens:



Here it is the description of the process:

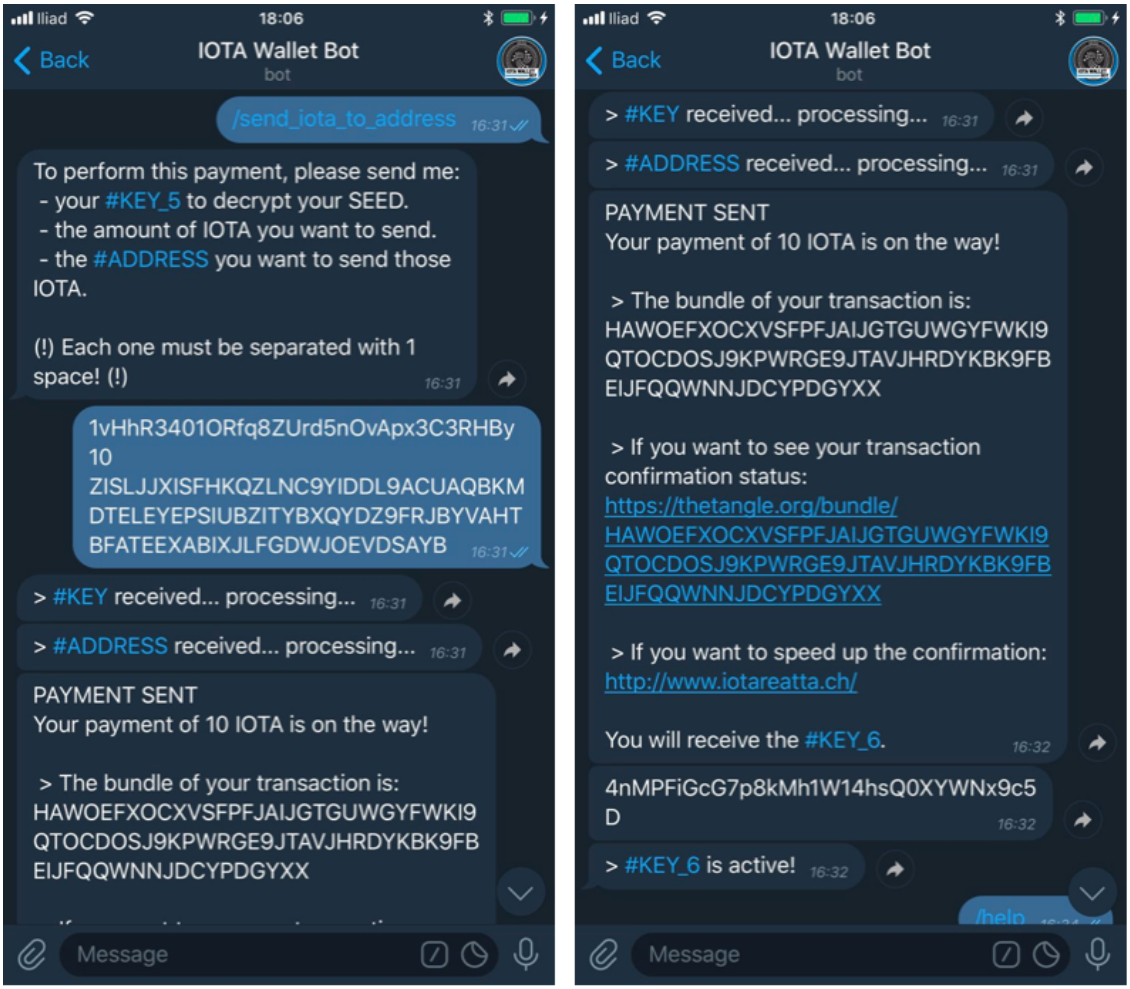
1. All the process before is skipped (the user sends the command so some iota is sent to an address, then the encryption key is requested to the user...).

2. Now, after the seed has been decrypted, and having the amount to be sent and the recipient address, the process to build this transaction can start. The Bot server needs to check the address (associated to this seed) to be used.

3. The Bot server requests to search those addresses in the tangle using the full node. It Performs findTxObjects.

4. A call to the full node API is performed (findTxObjects).
5. The full node performs this search on the tangle.
6. The results are then sent back to the caller.
7. Telegram bot receives all the transactions attached to the tangle that are related to the requested addresses (previously taken from DB).
8. Now I can use, for each address, the group of transactions retrieved from TxObjects (in the case that a transaction has been reattached more times) to compute a provisional balance for each address.
9. From these addresses I choose the address with the minimum balance removing it from the group of the TxObjects and then, this is repeated until the requested amount of IOTA is reached (if there aren't enough IOTA, the transaction cannot be performed).
10. If it is needed, a remainder address is added to the transaction (in the case the amount of inputs balances considered is more that the requested amount of IOTA for the requested transaction).
11. Create a new TxObject in trytes with prepareTransfer function.
12. Trytes are sent to the full node that will broadcast them as a new transaction when it finds and verifies 2 tips.
13. The full node receive trytes just created.
14. The full node search 2 tips from its pool or ask tips from its neighbors.
15. The 2 tips are verified.
16. If the tips are valid and verified the process can go on.
17. If one ore both tips is/are not valid (for example cannot be verified because of a double spending problem). The full node repeats the research.
18. The full node performs the PoW finding a nonce for this transaction.
19. The transaction is signed.
20. The transaction is then broadcast to the IOTA full node's neighbors.
21. The results are sent back.
22. The results of the requested transaction arrives to the Bot.
23. The application updates all the address balances marking the input addresses as used and the remainder (if used) in the local DB with the new balance.
24. All the rest of the process to communicate to the user all the results is skipped here since it is the same described previously.

Here it is a screenshot from the Telegram Bot:



To perform the payment in IOTA it is necessary to generate the bundle and then do all the requested work (get 2 tips and verify them, find a nonce doing the PoW, attach this nonce to the transaction object, attach the bundle and tips to the transaction and post everything to the tangle. After having signed the transaction, it is then transmitted to the full node's neighbors.

I implemented (as an experimental feature) an algorithm to manually compose a IOTA transaction before broadcast it. IOTA library offers the possibility to create a transaction by specifying only the amount of IOTA to send and the recipient address: this is very convenient but not enough (currently) in term of efficiency. I want to take into account also the balance of the pending transactions related to the proper account instead of waiting for all the transaction confirmation. I tried to speed up this process because the approval time of IOTA transactions is still high (on average) and fluctuates based on network performance and updates. The speed of IOTA network is higher that other DLTs and promises very well but it is not yet at the best levels promised. Imagine that a seed has 2 addresses A and B, the address A has a transaction pending (not yet confirmed: not yet referenced

by other transactions on the tangle) where it is receiving 30 IOTA and address B has another transaction pending where it is receiving 20 IOTA. The user owning this wallet wants to perform a transaction of 40 IOTA but at this moment it has an account balance of 0 IOTA. To speed up these two transactions it is possible to perform a reattach and/or a rebroadcast of them. This increments the probability that these two transactions can be selected as tips to be confirmed. To receive these two transactions, the two recipient addresses were generated and stored in the local DB. When a new transaction has to be done from this seed all the addresses, not used as inputs, must be checked. A research on the tangle has to be done to check if there are some pending transactions and then it is needed to remove, from the results, the duplicate transactions. Due to the reattach and rebroadcast done before in the tangle there are many transactions duplicates. When one of these transactions will be verified, just one among all the duplicates will be selected and verified, all other will remain forever in the pending state.

This is the complete procedure of the algorithm:

1. The situation is the one described above, with two incoming transactions in pending (total 50 IOTA incoming) and a requested transaction outcoming (to be sent 40 IOTA) to be created.

2. Before replying that this transaction cannot be done because of missing funds I check all the previously generated addresses and if there are some incoming payments in pending state on the tangle.

3. From the local DB are retrieved addresses A and B that are not yet used as inputs so can be used, as inputs, for this transaction.

4. For each address a research on the tangle is performed returning all the transactions' objects related to the searched address.

5. Then, if some object has the transaction confirmed it is immediately used to compute the balance of that address, if all of those objects are in pending all duplicate transactions must be removed from each group. Since the algorithm doesn't permit to use an address twice or more times as inputs this can be done easily. Then a provisional budget of that address is computed.

6. All the provisional budgets for each researched address are used to satisfy the requested amount of IOTA to perform the transaction to create the new bundle.

7. If it is needed a remainder address it is added to the bundle. Then it is address to the local DB with the balance (the sum of the provisional balance - the requested amount to perform the transaction).

8. The other used addresses are marked as used with a 0 balance on the local DB.

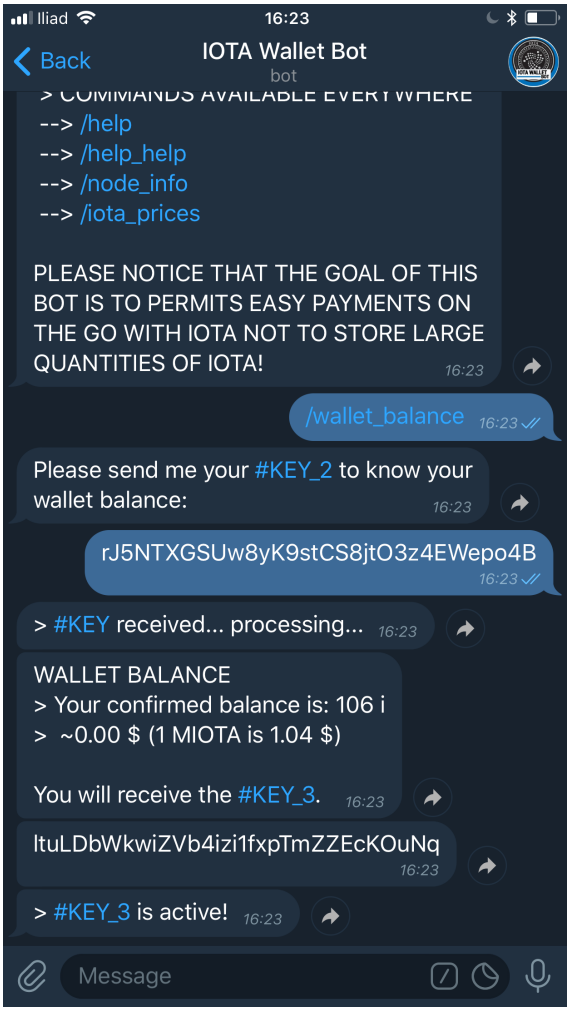
9. Then when the transaction is broadcast it is needed to check if the previous two transactions have been confirmed and broadcast them otherwise since the new transaction depends on those 2 old transactions.

This is what is done to speed up the process without waiting anymore for any transaction confirmation.

4.3.3 Check the available wallet balance

Interacting with the bot It is possible to know the proper wallet balance just running "/wallet_balance". Notice that here it is reported the available balance and not the account balance. Like a bank account, in IOTA there are two different balances: the available balance and the account balance. The available balance is referred to the balance taking into account all the confirmed transactions related to the proper account. In IOTA it is not possible to compute in a secure way the account balance since a transaction that can be in a "pending" state can be rebroadcast or performed many times. A transaction is approved on average in 5 minutes or less but it is possible that it remains in a pending state longer than that. Currently a transaction can be approved in two different ways: if the Coordinator (see section 3.1.3.1) references this transaction or if another node (that wants to send another transaction) takes this transaction as a tip to approve. If the transaction is validated from another transaction is then confirmed. If a transaction remains on a pending state for more than 30 minutes (nobody chose this transaction to confirm or there is some inconsistent information regarding this transaction), the possibility that this transaction will be selected decreases drastically. So a solution, to speed up this process, can be to perform another 0-valued transaction, or to rebroadcast this transaction to our full node neighbors or to create another transaction with the same bundle (see section 3.4.1). All of these possibilities to duplicate a transaction (to speed up the verification process) make the same transaction appear several times in the tangle. These possibilities are allowed and are not a malicious attempt to generate double spending attack. In fact when someone validates one of these duplicates just the first will be validated and confirmed, all the other copies will be discarded and never validated (this is part of the verification process, see section 3.1.2). In IOTA it is not possible to trigger the transaction confirmation event and the only possibility is to check the balance considering only the valid and confirmed transaction related to the considered seed. I tried to develop, as experimental feature, a way to retrieve the account balance trying to delete duplicates from all the same transactions on the tangle (see section 4.3.2). The bot returns also the corresponding balance in USD doing a simple multiplication of the available IOTA on the proper bot and IOTA prices retrieved from Bitfinex trading site (see section 4.3.5.2).

Here a screenshot from the Telegram Bot:


Iliad
16:23
Back
IOTA Wallet Bot
bot
--> /help
--> /help_help
--> /node_info
--> /iota_prices
PLEASE NOTICE THAT THE GOAL OF THIS BOT IS TO PERMITS EASY PAYMENTS ON THE GO WITH IOTA NOT TO STORE LARGE QUANTITIES OF IOTA!
16:23
/wallet_balance
16:23
Please send me your #KEY_2 to know your wallet balance:
16:23
rJ5NTXGSUw8yK9stCS8jtO3z4EWepo4B
16:23
> #KEY received... processing...
16:23
WALLET BALANCE
> Your confirmed balance is: 106 i
> ~0.00 $ (1 MIOTA is 1.04 $)
You will receive the #KEY_3.
16:23
ltuLDbWkwiZVb4izi1fxpTmZZEcKOuNq
16:23
> #KEY_3 is active!
16:23
Message

4.3.4 Receiving payment

To receive a payment in IOTA it is necessary to generate a new address from the proper seed and share it in order to receive it on this address. The command is "/get_address". With this command it is possible to generate an address and then, of course, it is required the encryption key. The address is generated using the proper function of the iota.lib.js library. From a seed it is possible to generate infinite addresses changing a parameter called "index". To generate the address all the indexes and the addresses already generated are stored on the local DB and this makes the procedure much faster. With iota library it is possible to generate an address for a seed without knowing the index but this process can be very slow because it starts a search on the tangle for every address starting from the address with index 0 and if that address is not found on the tangle it is considered as a "new address". This method is not so fast and secure because can be a problem after IOTA Foundation performs a snapshot and this process can slow down even more. Currently the fastest solution is to store and manage indexes of the addresses for each seed. On the local DB, as said before, it is stored the last index used and the address associated. Thanks to this when a new address is requested it is possible to have a completely new address. It is not recommended to use an already used address instead a new one to avoid the possibility to use many time the same address as the source (used as input address) in many transactions. Imagine that an address A it is generated and it is used to receive a payment. After receiving this payment on this address will reside X IOTA. It is allowed to use this address A to receive another payment of Y IOTA and then in this address will reside all the received IOTA (X+Y). If I use this address A to receive another payment of Z IOTA and in the same time I decide to perform a transaction of (Y+X) IOTA, the address where reside all the received IOTA (Y+X) is used as input address. After the payment is performed in the address A there will be 0 IOTA but then the other Z IOTA will arrive. In conclusion, to perform another payment and use the remaining Z IOTA from address A it is mandatory to reuse address A as input again and it is allowed but in IOTA it is very dangerous because on each payment it is discovered a part of the private key used to perform the sign of the transaction. So this must be avoided and for each payment in this bot it is advised to use a new address. On the bot other features are implemented to understand if it is reusing an address already used as input. Each address is stored with its proper index and when a payment is performed from that address it is marked as used and is never taken into account.

Another important experimental feature, that I implemented on my project, is the algorithm that permits to reuse the remainder address (if present) of a previous pending transaction just performed from this bot as input for another transaction. On the classical scenario this is the situation: the wallet has an address A with 20 IOTA, then a transaction is performed and 15 IOTA are spent. The bot takes the address A and uses it as input for the transaction, then it creates another address B and uses it as remainder address to put back the remaining 5 IOTA (not used on the previous transaction). The transaction is not immediately confirmed but the user needs to perform another transaction of 10 IOTA. Using the available IOTA function, when you want to perform a transaction, input addresses are retrieved

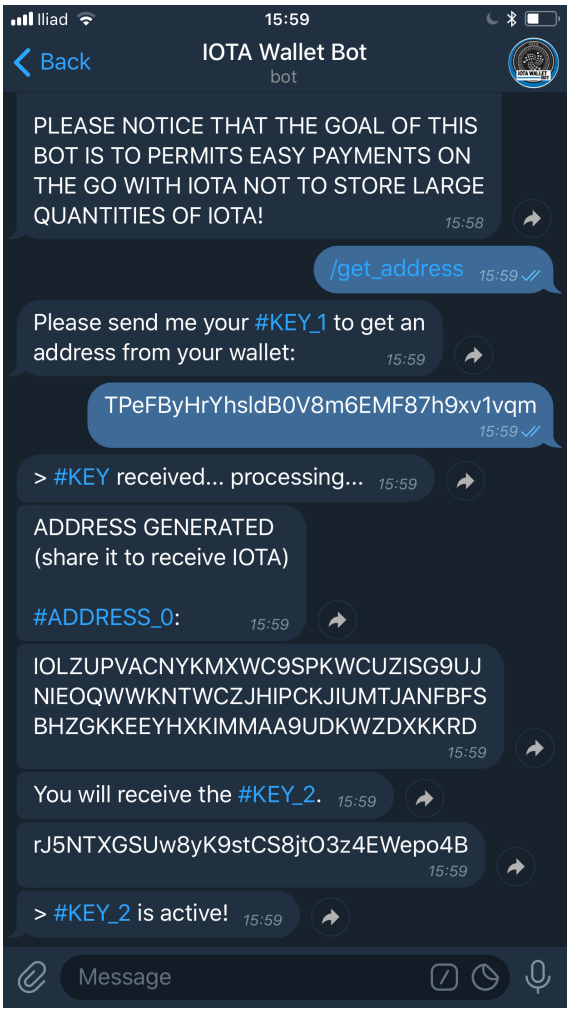
automatically searching on the tangle among the confirmed transactions (using the getInputs function) to understand what are the available addresses related to that seed that contains some IOTA and that can be used as inputs for a transactions. Considering the example described before, the address B is not considered because the transaction is not confirmed and the "getInputs" function found just address A with 20 IOTA available and then address A it is used (twice) as inputs address for another transaction. Following the IOTA rules this is not correct because the address is used twice and if one of the two pending transaction is confirmed the address A will be erased moving the remaining IOTA to a remainder address and then one of the two transaction doesn't find enough IOTA to perform the transaction.

To avoid this mess I developed an algorithm, modifying also the iota.lib.js, that permits to take into account all pending transactions:

1. The User has 10 IOTA on address A and wants to send 6 IOTA to another user.
2. The Bot sends 6 IOTA, as requested, and then generates an address B, as remainder address, to put back the remaining 4 IOTA.
3. On a local DB all addresses are stored with useful information:
   (a) Address A has been used as input address: marked as used (never reuse it).
   (b) Address B has 4 IOTA (in the tangle all of this information is not considered until the transaction won't be confirmed but I have all saved in the local storage).
4. The User wants to send 3 IOTA.
5. I check my local DB if I have enough IOTA to satisfy the requested payment by checking my addresses and related balances.
6. In this case I know that address B has 4 IOTA (if I check this address on the tangle has a balance of 0 IOTA).
7. I use address B as input address for the next requested transaction creating manually a new bundle:
   (a) Address B is used as input for the 3 requested IOTA.
   (b) Address B is the marked as used and has the updated balance of 0 IOTA. It is created a new address C with the balance of 1 IOTA (the remainder of this
   (c) transaction).
8. After creating this bundle I can continue the process of the transaction creation and broadcast to my neighbors.

This is the solution that I developed to permit multiple transactions even if the previous is not confirmed. Of course all of these pending transactions must be confirmed checking every tot time if they have been confirmed or not. We need to make sure that this chain of transactions is confirmed in order.

Every time a new address is generated it is directly attached to the tangle performing a 0-value transaction. This permits to see this address on the tangle even with a balance of 0 IOTA.

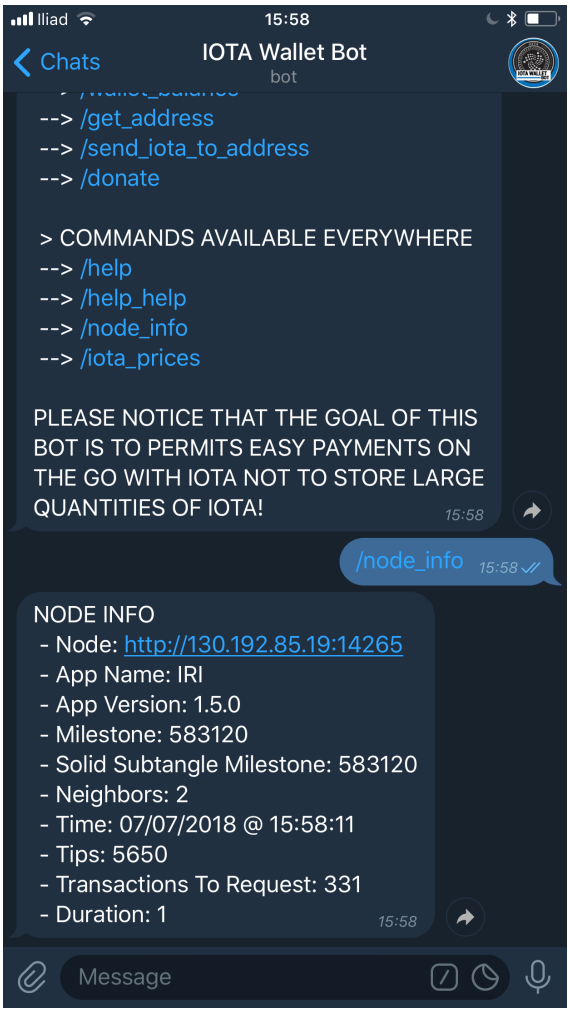


4.3.5 Other commands

These are the general purpose commands not directly related to the wallet essential features.

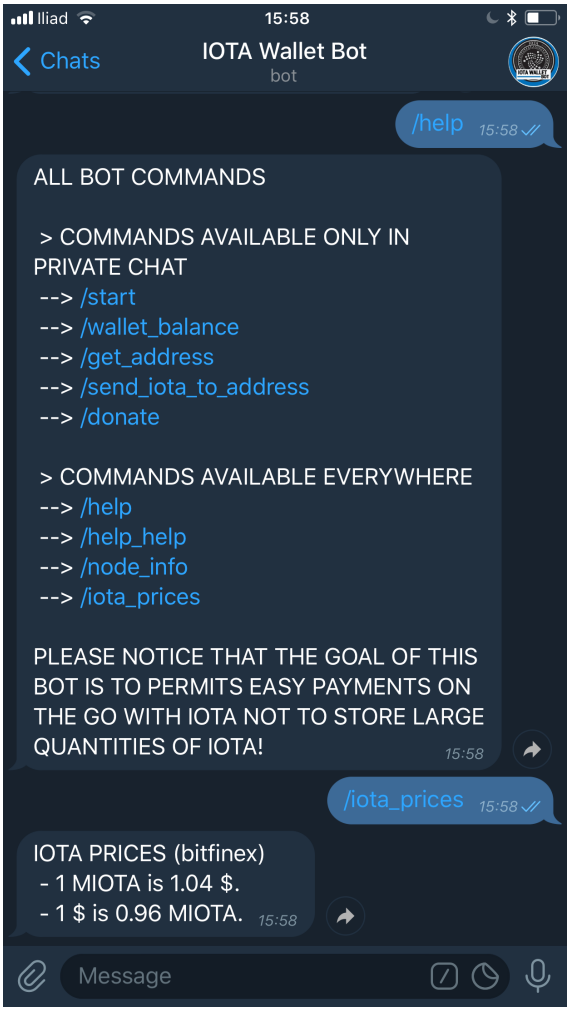
"/node_info"

With this command it is possible to retrieve all the information regarding the IOTA full node to which the bot is connected. With this command it is possible to understand if the full node is fully synchronized or not. Notice that, referring to the "Milestone" index, that represents the latest milestone emitted from the Coordinator, and checking the "SolidSubtangleMilestone", which represents the latest milestone considered on the Tangle copy of the node, it is possible to understand the percentage of synchronization. If the 2 values are the same the node is fully synchronized.



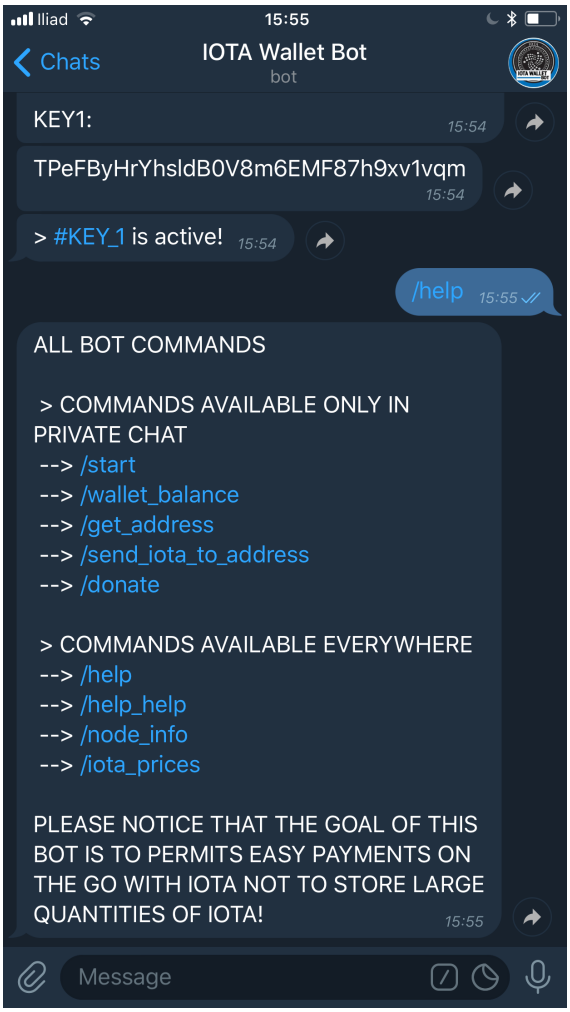
"/iota_prices"

Sending this command the Bot uses the JS "bitfinex" library to exploit the available API of the famous trading site Bitfinex to get IOTA prices. After the request completion the IOTA prices, in the form MIOTA/USD and USD/MIOTA, are returned. Remember that the basic tradable unit of IOTA is intended as MIOTA (Mega IOTA = 1,000,000 IOTA).



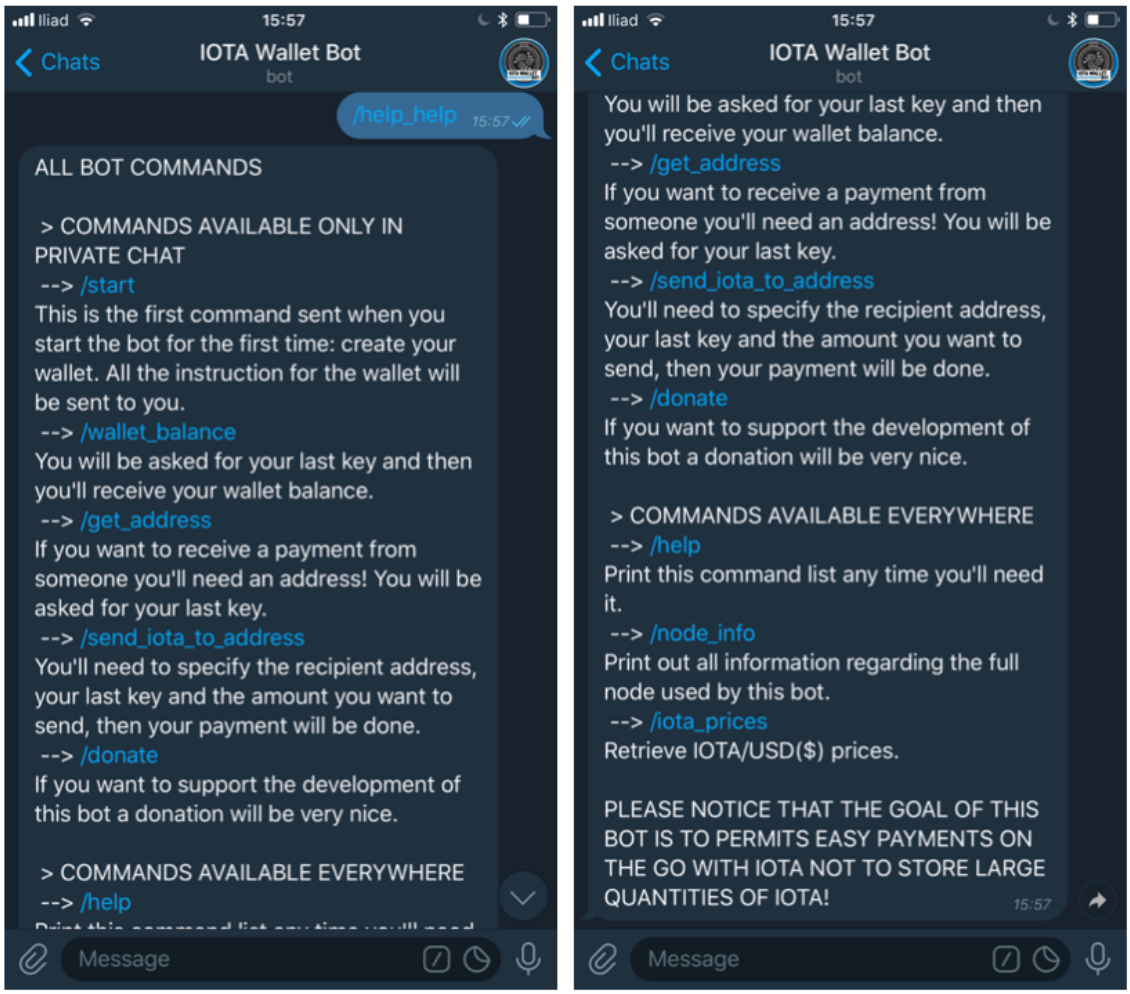
"/help"

This command can be useful to get a shortcut for all available bot commands.



"/help_help"

This command is the same of the previous command but has a detailed description of each command.



"/donate"

With this command it is possible to donate some IOTA to the developer of the Telegram bot. It is requested to the user to specify the last encryption key and the amount of IOTA he wants to donate. (the process is the same of the send payment previously described without the specification of the address since it is automatically generated from the developer's wallet).

Chapter 5

Final considerations and conclusions

IOTA is a relatively new ecosystem and it is still difficult to express final opinions about its architecture and performance. Many features are still being implemented, and even the tangle is not yet fully functional in a comòletely decentralized way. What can be definitely said is that it is very promising and the team behind the project always showed great enthusiasm, competence and attention to the technology implementation and improvement more than to other aspects such as speculation. The basic architecture is ready and stable and the IOTA Foundation developer team is carrying out constant tests to make it more and more secure and stable, so that soon it will be possible to remove the coordinator process and have a truly decentralized system. It is really impressive to see how from an innovative idea they have managed to create a group of people scattered all over the world who are very organized and collaborate on different topics, keeping themselves constantly updated. Technologies for collaboration allow all this to work fine and the passion and dedication of the team has allowed people interested in the subject, like me, to get in touch with the core developers, be accepted in the community and therefore have a priority lane to quickly learn the principles of this new technology. Nowadays there are still issues regarding the storing of non transactional data, and since IOTA is designed for IoT, this can be a big problem for users interested in using the tangle as a decentralised database. We must keep in mind that some solutions are already designed and are waiting to be implemented and tested: permanodes and automated, local forms of snapshots. Currently all of these type of problems can be addressed with the support of a traditional database. Nowadays the IOTA Foundation team is working to strengthen the IOTA base structure by releasing frequent updates of the core components such as the full-node software (IRI) and the client libraries. The IOTA Foundation is currently searching for new passionate and talented people to carry on this project collaborating and working on all the project and this is the right way to obtain great results in a short time. Aside from the development of libraries and core applications, the IOTA foundation offers many possibilities to participate in different funded initiatives such as hackathons and projects aimed at trying and testing the future applications that will be developed, such as the already mentioned IOTA data Marketplace. Our thesis group participated in this project from the very beginning when it was launched in December 2017: we connected several devices equipped with sensors configured to send

sensors' data (weather and environmental parameters in our case) to the tangle. . The purpose of this simple project was testing the MAM (Masked Authenticated Messaging) mechanism learning how to create a data stream that could be later retrieved (possibly for a fee) directly from the tangle. As previously discussed in the IOTA section about their roadmap, another really important feature is the Qubic project, currently being implemented (work in progress). Qubic is a protocol that specifies IOTA's solution for oracle machines, smart contracts, outsourced computations, and more. Finally, due to its focus on the new, data-centered economy, , technologies like IOTA will definitely be among the protagonists of the next digital revolution by enabling major disruptive paradigm like the so called machine economy.

5.1 IOTA roadmap and future implementations

IOTA was initiated with the very clear and focused vision of enabling the paradigm shift of the Internet of Things, Industry 4.0 and a trustless 'On Demand Economy' through establishing a de facto standardized 'Ledger of Everything'. Since the very first lines of code that were written in mid 2015, the project has come a long way, but still the IOTA development has only barely begun. The IOTA roadmap is full of new features and functionalities that have been announced or are in the development and testing phases. The IOTA main features are available and the mainnet is in a stable version but many more announced features are currently under development.

5.1.1 Neighbor auto discovery

Among the new features in the IOTA roadmap a relevant one is the release of an auto-discovery algorithm. Currently IOTA full nodes can add neighbors only manually. The availability of auto discovery would permit a fast way to become part of the IOTA network.

5.1.2 Automated snapshotting and permanodes

Tha tangle allows to use transactions also to store information different from a token transfer, by just attaching to a Tx any data type in a specified format (like a json object). This type of transactions (zero-iota transactions) can be lost if a snapshot event occurs. Snapshotting is a technique (currently centralized and carried out by the IOTA Foundation) introduced to reduce the size of the tangle. When a snapshot occurs, all the non-zero-iota transactions are compacted in the resulting balances; all the zero-iota transactions are simply removed.. This is a major problem for projects where the tangle is used just for data storage. IOTA is working on a sort of "Automated Snapshots" instead of the centralized actual snapshot version and on an implementation of "permanodes". Each user will be able to decide when and how to perform his own personal snapshot and will decide, with a personal permanode implementation, which data to keep and which to delete.

5.1.3 Identity of things

The IOTA project is offering its contribution to the concept of digital twin for IoT devices by supporting the so-called Identity of things. Rather than perceiving machines/devices as lifeless amalgams of metal and plastic with a specific purpose, they can be considered as having an identity with different attributes associated. A sensor should not only have its unique identifier, but also its accompanying attributes such as: who manufactured it, when it was deployed, what is the expected life time cycle, who owns it now, what kind of sensor data it is gathering, at what granularity and many other information. This is imperative for an efficient way to index and enable the Machine Economy and Industry 4.0 services such as the so called predictive maintenance.

5.1.4 Masked Authenticated Messaging

As mentioned above, the MAM makes it possible for sensors and other devices to encrypt entire data streams and securely attach them to the IOTA Tangle in a secure, quantum-proof way. Only authorized parties will be able to read and reconstruct the entire data stream. In essence it works a lot like a radio where only those with the right tunercan listen in: with MAM only users in possess of the right channel IDs can get access to the data. It is possible to imagine its applicability to industries such as Supply Chain, Banking and Industry 4.0.

5.1.5 Flash network

Since IOTA is thought for IoT devices, it is possible to perform many transactions at high speed and finer granularity. It is possible to imagine different scenarios in which many transactions can be inserted in a "private" channel between two parties (off tangle) in a faster way. When the channel is closed, a single transaction will be attached to the Tangle containing the summary of all the transactions included in the channel. This can be applied, for example, in a car park where each minute a small payment is performed (transaction) by the customer. The off-tangle transactions allow to keep track of the payments made, but just the last transaction is stored in the ledger. In this way it is possible to avoid the waste of time and resources for each transaction by doing the proof of work only once at the end.

5.1.6 Qubic and Oracles

Qubic is the last announced set of features for the tangle, a protocol that specifies IOTA's solution for oracle machines, smart contracts, outsourced computations, and more. Qubic is still under development. It was announced in June 2018 and some details about its future implementation and development are coming out in these days (July 2018). No software is available for practical implementation yet.

Appendices

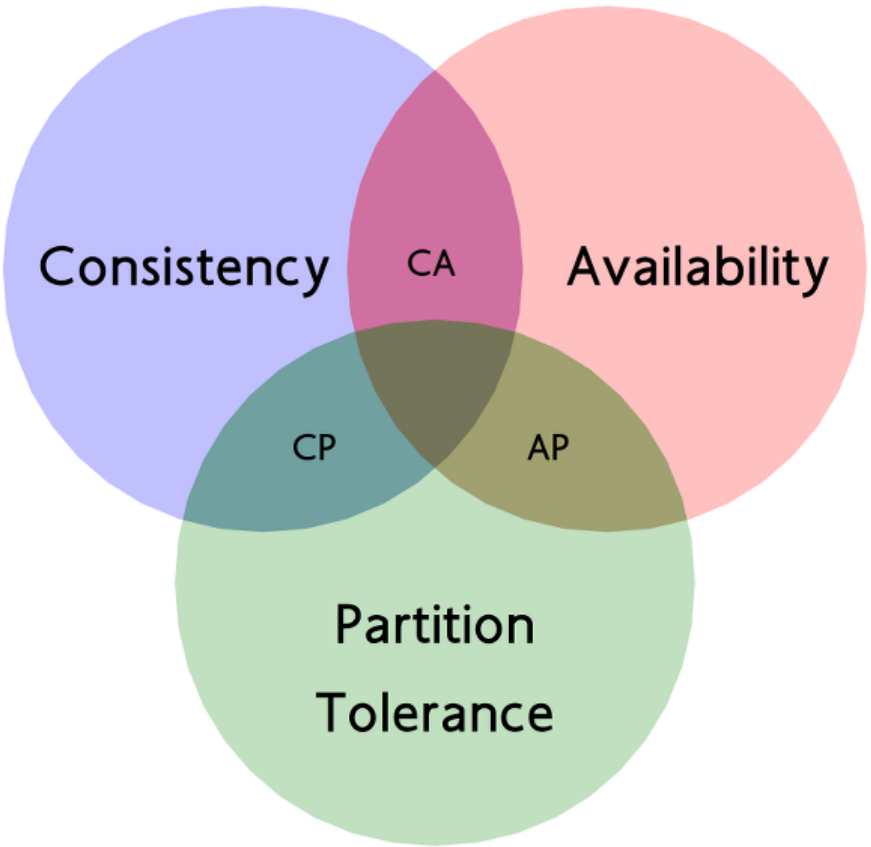
Appendix A

CAP theorem

This theorem was enounced by Professor Eric Brewer from Berkeley university during the Principle of Distributed Computing (PODC).

This theorem describes the trade-off involved in the world of distributed systems. This is the statement of the following theorem:

> 'It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees: Consistency, Availability, Partition Tolerance."



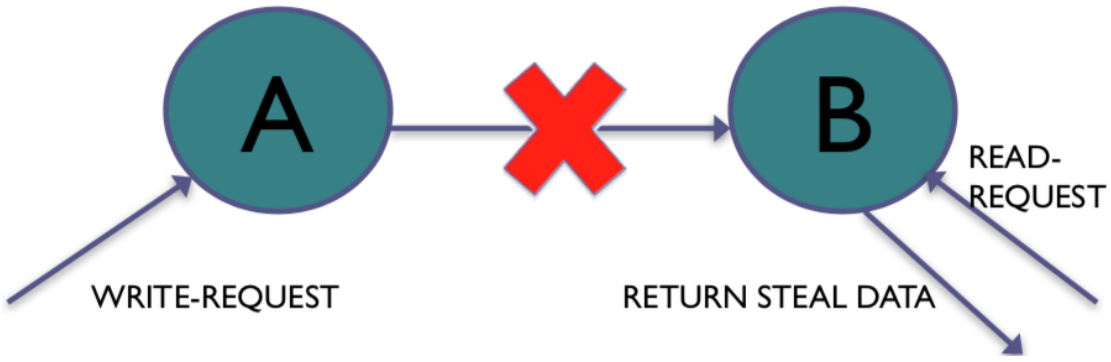
- **Consistency:** Every read receives the most recent write or an error;
- **Availability** : Every request receives a (non-error) response, without guarantee that it contains the most recent write;
- **Partition tolerant** : The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

A.1 CAP theorem properties

A correct understanding of the CAP theorem is essential for making the right decisions during the design phase of a distributed system. Those are all the properties' definitions:

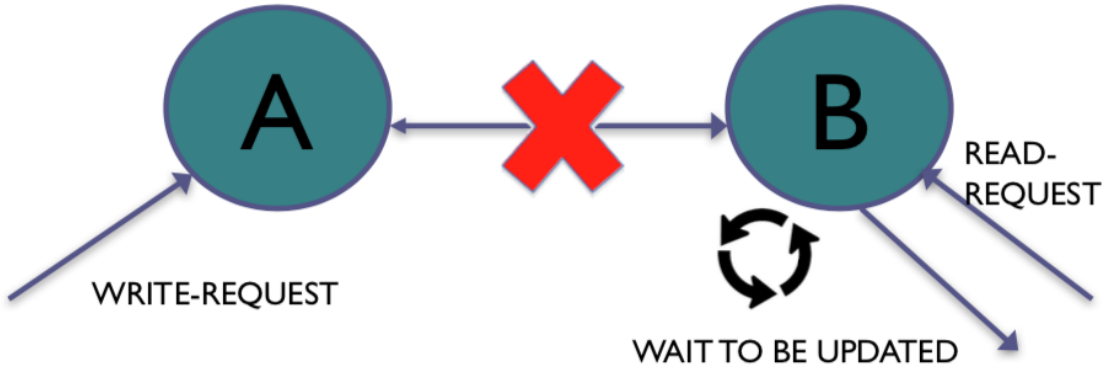
- **Consistency:** ability to have a single updated copy of data, on all replication nodes (serializability);
- **Availability:** ability of a system to always provide a negative or positive response to a query;
- **Partition tolerant:** ability of a distributed system to continue to operate even in the presence of network errors.

A.1.1 Not consistent



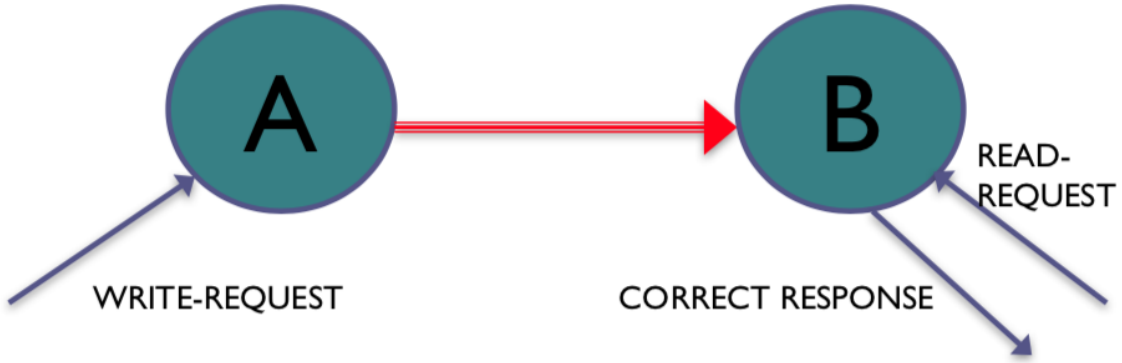
A and B have a complete copy of the data. In the case of a partition, they do not receive updates from each other and return inconsistent data. This system has the availability and partition tolerant property (AP) but not the consistent property (C).

A.1.2 Not available



A and B use an internal protocol to keep synchronized.In the case of a partition, they can not hear and refuse to respond to requests until they synchronize again. This system has the consistency and partition tolerant property (CP) but not the availability property (A).

A.1.3 Not partition tolerant



If A and B do not force mutual listening and synchronization, they can respond even in the presence of a partition. Any updates received from a node while the network was partitioned, are lost. This system has the consistency and availability property (CA) but not the partition tolerant property (P).

A.2 Consistency in a distributed system

Moving from a centralized to a distributed system, the concept of consistency must be reviewed. The consistency model Defines the set of rules that a system must follow to guarantee the type of consistency described by the model adopted. The

level of consistency between the different operations has a strong impact on the overall performance of the system and also on its usefulness in a given application domain.

A.2.1 Distributed consistency model

Moving from a centralized to a distributed system, 4 new types of consistency are needed to be defined:

- **Eventual Consistency:** sooner or later, all the replicas will have the same copy of the data;
- **Session Consistency:** within a session, a real order can be guaranteed;
- **Casual Consistency:** this model guarantees that the operations will be applied to the datastore in 'causal' order;
- **Serializability:** a global order of operations on all is guaranteed on all the replies. Each reading on a data always returns the most updated value of the same.

A.3 BASE property vs ACID property

For completeness, I report the definition of ACID propertie.

In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc. In the context of databases, a sequence of database operations that satisfies the ACID properties, and thus can be perceived as a single logical operation on the data, is called a transaction.

- **Atomicity:** transactions are often composed of multiple statements. Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes;
- **Consistency:** ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is correct;
- **Isolation:** transactions are often executed concurrently (eg. reading and writing to multiple tables at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would've been obtained if the transactions were executed sequentially. Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions;

- **Durability:** guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (eg. power outage or crash). This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

After stating the CAP theorem, E. Brewer proposed the BASE properties as opposed to the existing ACID:

- **Basic Available:** indicates the ability of a system to guarantee availability in terms of the CAP theorem;

- **Soft State:** indicates that the system state may change over time even without input, due to the eventually model;

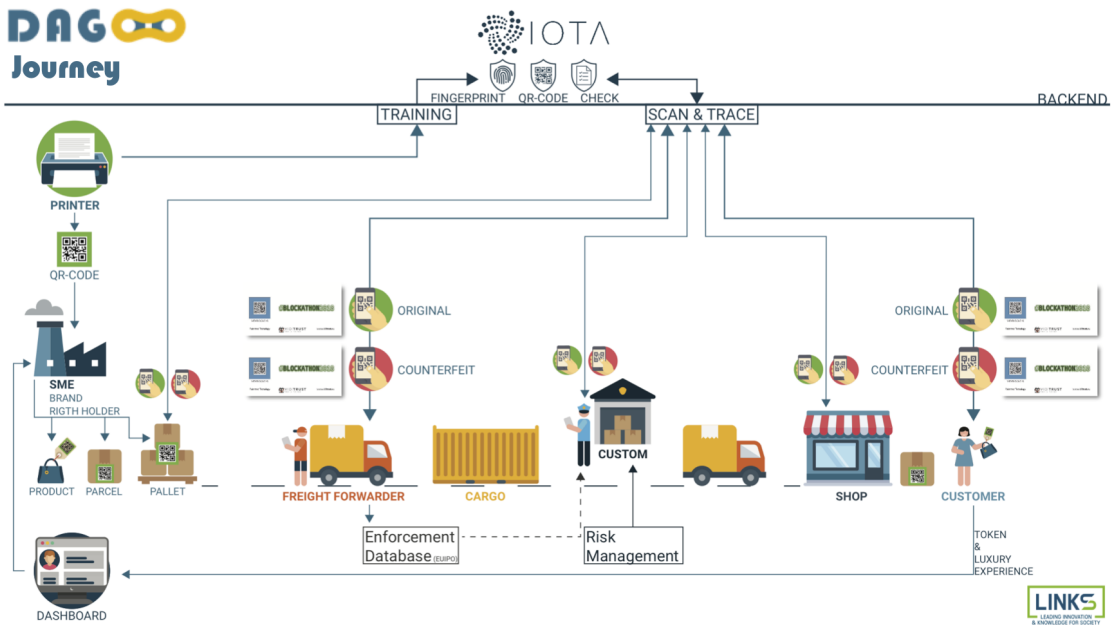
- **Eventually Consistent:** the system will become consistent according to the CAP theorem. For example, all replication nodes will have the same copy of the data, in a certain time interval during which the system will not receive input from the outside.

Many NoSQL DBs follow the BASE properties just mentioned thus providing the possibility, to those who use them, to have an architecture capable of scaling horizontally.

Appendix B

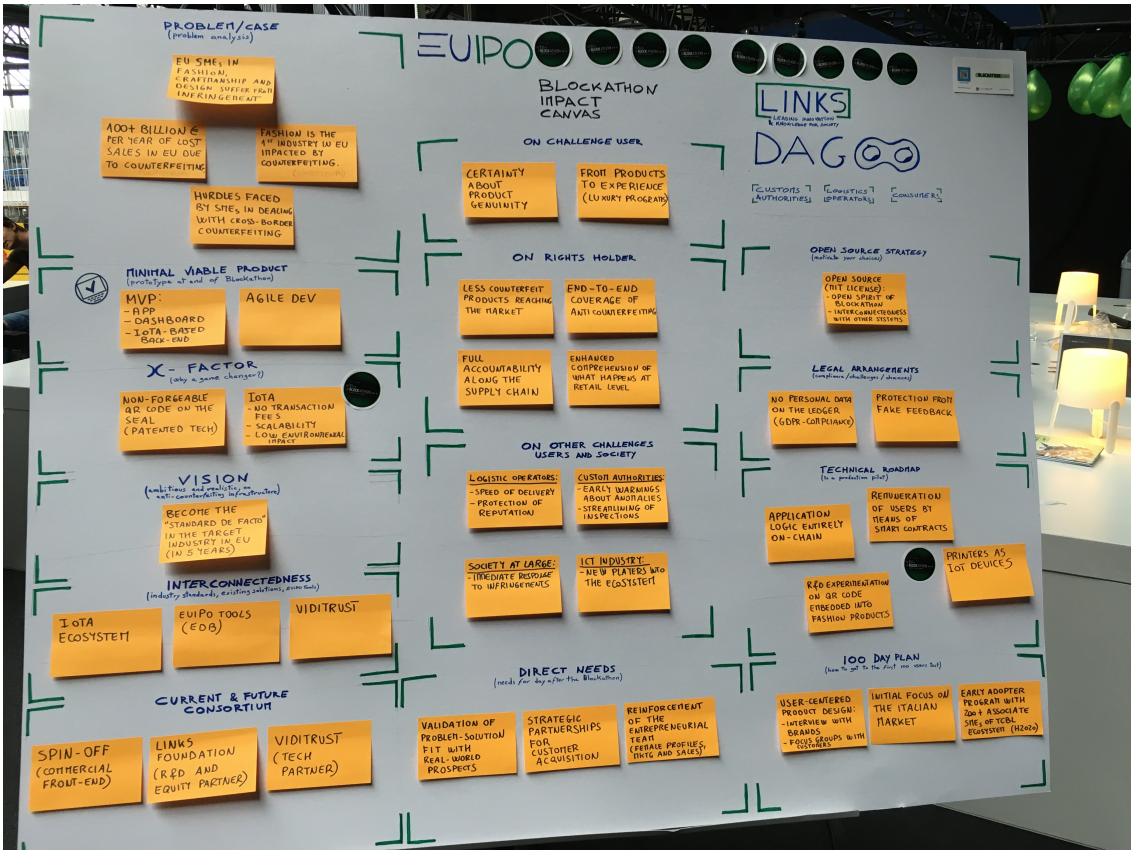
EUIPO Blockathon 2018

In the Spring 2018 the EUIPO (EU Intellectual Property Office) organized a competition to select the best and most pioneering teams to join a hackathon dedicated to design innovative solutions for anti counterfeiting based on DLT., The 11 selected teams gathered in Brussels and worked for three days directly with manufacturers, logistics companies, customs, retailers and consumers. Our team, composed by Andrea Vesco, Michele Osella, Gian Marco Toso, Alberto Buzio, Jure Rosso and me was selected as 1 of the 11 teams to join this 4-day challenge (22 June 2018 - 25 June 2018) in Brussels.



For this competition we designed and implemented a prototype of an ecosystem to keep track of a product from the producer to the consumer, giving different groups of users (end user, logistic operator, custom operator) the possibility to check whether the product they handle is original or not. The highlights of this ecosystem was the use of two innovative technologies: IOTA as blockchain-like architecture to store trusted and immutable information about the product along all the process, and a special, unforgeable QR code created by the Italian startup VidiTrust. One such QR code is associated by the manufacturer to each original product. At each step of the shipping process (including the times when the original product is packaged in bigger container or unpacked) the QR code is scanned and the retrieved information is saved in the tangle using a transaction. The IOTA Masked Authenticated Message, one of the IOTA expansion modules, offers the possibility to create a virtual chain by connecting transactions which are part of the same stream. In our project this gives the possibility to follow each product along the whole logistic path. Since at each scan it is verified the integrity of the product, checking the authenticity of the QR code, and storing the result on the tangle, it is possible to detect where the product has been substituted or counterfeited along the entire process. As already described, IOTA offers all the requirements to be used as a DLT to store sensitive information about a product. For this challenge we developed a proof of concept proving how it is possible to read/write from/to the tangle some information about the product. We developed an Android application to scan the QR codes using VidiTrust technology and then, passing the results to our server, write the data to the tangle. Then we developed a web-based application to read for each product all the information generated by all scanning events, giving the possibility to analyze all the traffic regarding original or counterfeit products.

The picture below shows the impact canvas used to illustrate the impacts of the solution, the go-to-market strategy, and its competitive implications.

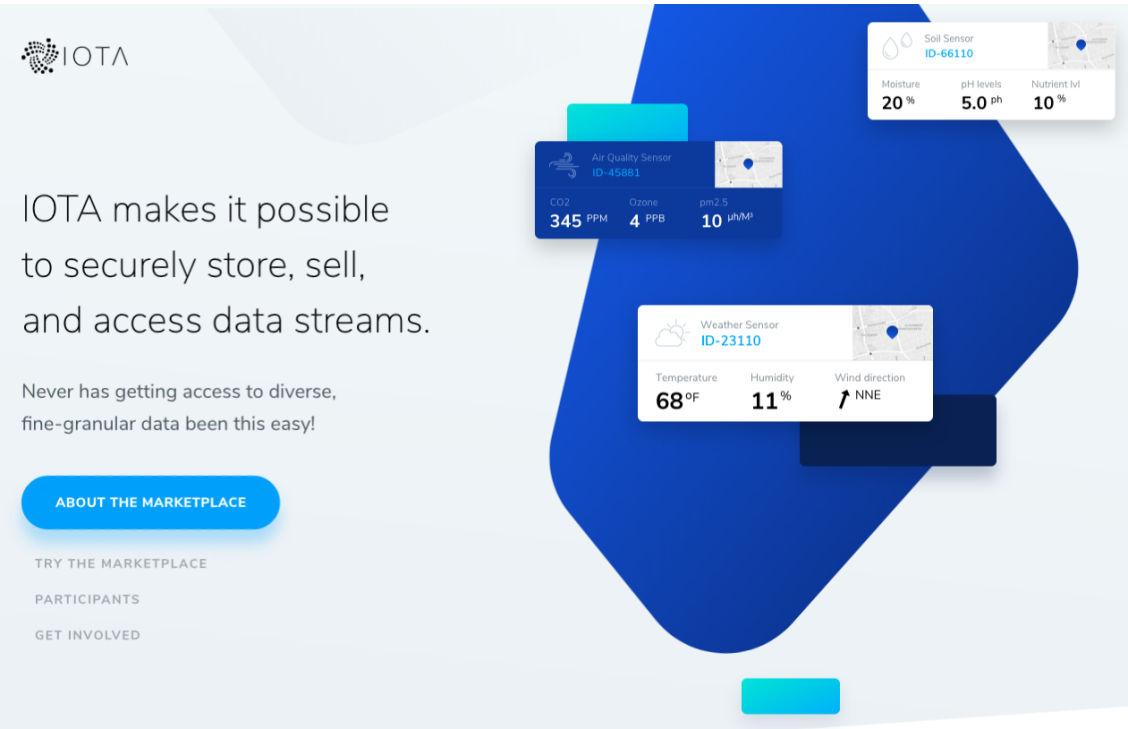


The EUIPO Blockathon was a great opportunity to propose the use of IOTA in a real and modern use case.

Appendix C

The IOTA Data Marketplace

The Data Marketplace is IOTA's most comprehensive pilot study thus far. The goal is to enable a truly decentralized data marketplace to open up the data silos that currently keep data limited to the control of a few entities. Data is one of the most imperative ingredients in the machine economy and the connected world.



I was able to participate to the datamarketplace as part of the research team in ISMB studying IOTA for my thesis. This project consists in setting up a series of devices with some sensors and then using the IOTA Masked Authenticated Messaging to post streams of data. As you can see, this project involved most important companies all around the world:

Then, after setting up some devices it is possible to use the portal, developed by

IOTA developers, to see a little demonstration showing how to buy some interested data from the choosen device. You can select a device from a map and then after clicking on it, if you already bought the data you will see all the history of that device, otherwise you will be asked if you want to buy related data. Here it is an example of some data bought from my device BerryPi_RuuviTag.

My device is a Raspberry Pi 2 B that reads every 10 minutes some data (temperature, humidity, pressure, battery percentage and accelerations of all axes of the accelerometer) from a RuuviTag device. This Ruuvitag is a device with some sensor that transmits data over Bluetooth to all devices listening.

Bibliography

[1] Il sole 24Ore cybersicurezza
http://www.ilsole24ore.com/art/tecnologie/2018-02-26/come-difendersi-minacce-informatiche-guida-sole-cybersicurezza-193316.shtml

[2] IoT data marketplace
https://blog.iota.org/iota-data-marketplace-cb6be463ac7f

[3] Practical Byzantine Fault Tolerance
Miguel Castro, Barbara Liskov, "Practical Byzantine Fault Tolerance", Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999

[4] Hashcash
Adam Back. Hashcash, May 1997. Published at http://www.cypherspace.org/hashcash/

[5] Centralized vs Decentralized
https://medium.com/@shyamshankar/centralized-ledgers-vs-distributed-ledgers-layman-understanding-52449264ae23

[6] Telegram Bot
https://telegram.org/blog/bot-revolution

[7] npmjs
https://www.npmjs.com

[8] mullwar/telebot
https://github.com/mullwar/telebot

[9] bitfinex api node
https://github.com/bitfinexcom/bitfinex-api-node

[10] france193/IOTAWalletBot
https://github.com/france193/IOTAWalletBot

[11] iotaledger wallet
https://github.com/iotaledger/wallet

[12] iotaledger iri
https://github.com/iotaledger/iri

[13] iotaledger iota.lib.js
https://github.com/iotaledger/iota.lib.js

[14] iotaledger docs
https://github.com/iotaledger/docs

[15] iotaledger mam client
https://github.com/iotaledger/mam.client.js

[16] iotaledger
https://github.com/iotaledger

[17] iota ecosystem
https://ecosystem.iota.org

[18] iota docs
https://docs.iota.org

[19] iota readme
https://iota.readme.io/

[20] iota data marketplace
https://data.iota.org

[21] links foundation
https://linksfoundation.com/en/

[22] Mastering Bitcoin: Programming the Open Blockchain Paperback Andreas M. Antonopoulos

[23] IOTA whitepaper
https://iota.org/IOTA_Whitepaper.pdf