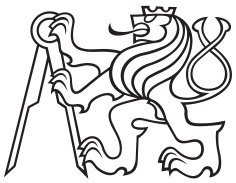
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Decentralized authentication of IoT devices based on blockchain technology

Viktoriia Chvykova

Supervisor: doc. Ing. Stanislav Vítek, Ph.D
May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Chvykova** Jméno: **Viktoriia** Osobní číslo: **485392**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Decentralizovaná autentizace IoT zařízení založená na technologii blockchain

Název diplomové práce anglicky:

Decentralized Authentication of IoT Devices Based on Blockchain Technology

Pokyny pro vypracování:

1) Proveďte rešerši technologií Blockchain a Smart Contracts. Vyhodnoťte vhodnost využití těchto technologií pro autentizace IoT zařízení.
2) Na základu rešerše navrhněte decentralizovaný systém autentizace IoT zařízení.
3) Navržený systém implementujte pomocí bezdrátových modulů s mikrokontroléry (např. ESP32).
4) Proveďte analýzu navrženého řešení z hlediska potřebné výpočetní kapacity a porovnejte s jinými přístupy autentizace.
5) Zhodnoťte dosažené výsledky.

Seznam doporučené literatury:

[1] ANTONOPOULOS, Andreas M.; WOOD, Gavin. Mastering ethereum: building smart contracts and dapps. O'reilly Media, 2018.
[2] ALFANDI, Omar, et al. A survey on boosting IoT security and privacy through blockchain. Cluster Computing, 2021, 24.1: 37-55.
[3] LONE, Auqib Hamid; NAAZ, Roohie. Applicability of Blockchain smart contracts in securing Internet and IoT: a systematic literature review. Computer Science Review, 2021, 39: 100360.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Pavel Hazdra, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Podpis studentky

Datum převzetí zadání

Acknowledgements

I would like to thank my master's thesis supervisor, doc. Ing. Stanislav Vítek, Ph.D., for our consistent communication and pleasant collaboration.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 20. May 2022

Abstract

This master's thesis examines the issues of IoT architecture and offers decentralized alternatives utilizing Blockchain and IOTA Tangle, with the provision of comparative evaluation. Onward, provided the operating concept of the IOTA Tangle (Chrysalis protocol version) and authentication principles. In the practical section, the X-CUBE-IOTA1 software package for the B-L4S5I-IOT01A platform is described, as well as the design and development of an application extension that provides remote device control utilizing Tangle technology on the Mainnet network. The proposed solution is implemented in the C programming language by utilizing STM32CubeIDE.

Keywords: Tangle, DLT, Blockchain, Security, decentralization, Internet of Things, constraint devices

Supervisor: doc. Ing. Stanislav Vítek, Ph.D
Praha, Technická 1902/2, místnost: B2-730

Abstrakt

Tato diplomová práce se zabývá problematikou architektury IoT a nabízí decentralizované alternativy využívající Blockchain a IOTA Tangle se zajištěním komparativního vyhodnocení. Dále je poskytnutý operační koncept IOTA Tangle (verze protokolu Chrysalis) a principy autentizace. V praktické části je popsán softwarový balík X-CUBE-IOTA1 pro platformu B-L4S5I-IOT01A a také návrh a vývoj aplikačního rozšíření, které umožňuje vzdálené ovládání zařízení s využitím technologie Tangle v síti Mainnet. Navržené řešení je implementováno v programovacím jazyce C s využitím STM32CubeIDE.

Klíčová slova: Tangle, DLT, Blockchain, Bezpečnost, decentralizace, Internet věcí, omezovací zařízení

Překlad názvu: Decentralizovaná autentizace IoT zařízení založená na technologii blockchain

Contents

Figures

Tables

Chapter 1

Introduction

The idea of the *Internet of Things* is not new; for the first time, its idea in its simplest form, by modern standards, was implemented in the early 1990s. But since efficient communication protocols had not yet been devised at that time, and besides, computer chips were too large, this model of the interaction of things did not succeed in accordance with its ideological potential.

At that time, the term "Internet of Things" itself did not even exist, which was first used by Kevin Ashton (co-founder of the Auto-ID Lab at MIT) in 1999 to describe a system in which the Internet is connected to the physical world through ubiquitous sensors, including RFID (radio frequency identification).

Ashton described the idea of IoT as: "Today computers—and, therefore, the Internet—are almost wholly dependent on human beings for information. The problem is, that people have limited time, attention, and accuracy—all of which means they are not very good at capturing data about things in the real world. If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss, and cost. We would know when things needed replacing, repairing, or recalling, and whether they were fresh or past their best. The Internet of Things has the potential to change the world, just as the Internet did".[8]

Today, living in the era of automation of everything and the unprecedented development of communication technologies such as 5G, Starlink, advanced WAN protocols, and others, ever-greater data throughput and transmission speed has become possible. In this regard, now many companies, and startups are promoting and implementing IoT technology in smart manufacture (automation 4.0), smart homes, smart cities, healthcare systems, environmental protection, logistics, wearables etc. In this situation, we are dealing with a huge system that provides comprehensive services from the physical to the application layer. And the situation is complicated by the trend of a constant increase in the number of connected devices to the network.

According to a recent study by Statista (a popular market statistics and consumer data company), more than one billion connected devices will be added to the world of the Internet each year. There will be 38.6 billion devices

connected to the Internet of Things (IoT) worldwide by 2025. In addition, projections suggest that around 50 billion of these IoT devices will be in use worldwide by 2030, creating a huge network of interconnected devices, covering everything from smartphones to kitchen appliances.[9] Therefore, modern technology is increasingly in need of solving the existing problems of its implementation. Like *highly centralized systems*, *data privacy*, and *access control*, *data integrity and authenticity*, *identity* and *data management*. So, to face this issues, over the past few years, significant efforts have been made to create technology that would help in solving them.

One of the most innovative solutions is the use of *Distributed Ledger Technology* (DLT). DLT is known to the world thanks to cryptocurrencies. But its application is much wider. It is also applicable to IoT and offers a solution to the main problems associated with its existing architecture. A distributed ledger is a form of digital database that is updated and held by every member independently in a large network space. In this type of ledger there's isn't any central authority to broadcast the record to every member. Every kind of DLT has its own way to reach an agreement while storing the information on the ledger. Onwards I consider two types of DLT: Blockchain (Bitcoin based) and Directed Acyclic Graph (DAG - IOTA based).

Traditional blockchain technology and DAG adopt the principle of DLT with architectural disparity. Their data structure is different, Blockchain consists of digital information organized as "blocks" and recorded in public databases known as "chains", weather DLT, known as the Tangle in IOTA, employs a mathematical concept known as directed acyclic graph. But both of them have next key characteristics, which we are seeking to add to the current IoT architecture: *immutability, network decentralized structure, enhanced security, distributed ledgers, consensus, faster settlement.*

At the same time, these technologies have some important differences that determine which one is more suitable for integration with IoT (this issue will be discussed in more detail in the next chapters). Based on all of the above this thesis aims to describe how distributed ledger technologies work in relation to the IoT. I take IOTA as a basis as it is a promising project with ambitions to become the basic protocol for decentralized IoT, taking into consideration why it is more suitable than blockchain-based technology to give readers a comprehensive sense of usability of this cutting-edge approach.

In addition, I am determining how to implement IOTA technology, and in particular the procedures required to interface with Tangle using the C programming language on STM32 microcontrollers. And based on this, I develop software that enhances the usefulness and application possibilities of IOTA Tangle on constrained devices.[4][6]

1.1 Introduction to Internet of Things

This chapter focuses on how the IoT architecture is currently implemented and what are the shortcomings of such an implementation.

1.1.1 What is IoT?

The term IoT is not standardized, there are many definitions of it, therefore, for a universal designation, it is best to describe what layers the IoT architecture, in general terms, consists of and what functionality it ultimately performs.
The Internet of Things, in general, is a system composed of four levels:

1. Thinks or Physical layer represents spatially separated electrical devices connected to the Internet, capable of collecting information from the environment (sensors) or performing actions (actuators).

2. Network layer includes Gateways and edge computers to supply a translation of data flow from the physical layer to the Internet, and it includes the Internet itself (IP, TCP, UDP provision).

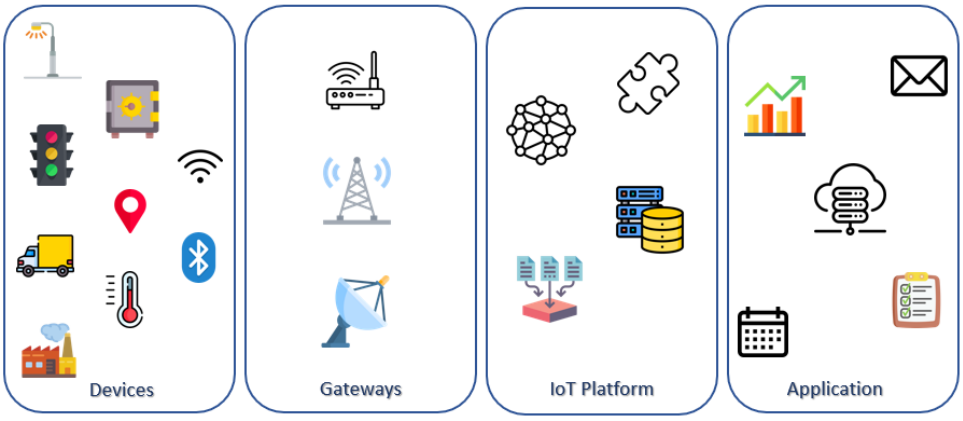
3. Platform layer is situated between the network and application layers and is often hosted in the cloud. Its purpose is to control and coordinate overall communication orchestration which means the responsibility for communication with downstream devices and ingesting large amounts of data at high speed. The platform is also responsible for storing the data in a time series and structured format for further processing and analysis.

4. Application layer is an intermediary serving the end-user as an interface for communication with the system.

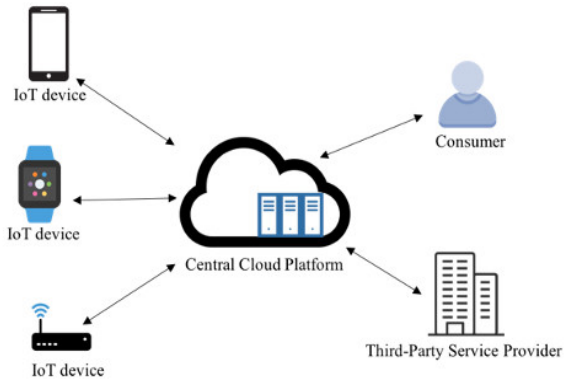
For the end user, these layers working together, providing the function of big data analysis, remote control, remote monitoring, automation, etc.

1.1.2 IoT architecture illustration

The pictures below is showing the building blocks of an current IoT solution. Figure 1.1a illustrates the described architecture where the four levels are clearly visible.
Figure 1.1b, in turn, is intended to draw attention to the main concept of modern IoT architecture, namely its centralization.
It is the factor that determines the number of problems inherent in such implementation.



(a) : Functional blocks focused



(b) : Centralization focused [1]

Figure 1.1: IoT solution

Let's see what exactly these disadvantages are.

1.1.3 Present architecture drawbacks

To understand why do we need to implement changes to present IoT architecture, first of all, necessary to figure out what shortcomings of it we are facing. As I mentioned earlier centralization is a source of problems in the IoT architecture, let us take a closer look at the main ones:

- ***Authentication and Authorization***

  The authorization ensures that only authorized users can access data on the network. Certain mechanisms are required to make communications secure and secure. In the absence of a strong authentication mechanism, spoofing can get in the way. In general, any situation where an attacker pretends to be someone else can be considered spoofing. It can take many forms, but most often it is a situation where fraudsters disguise themselves as a real phone number, e-mail, IP address, or a credible-looking website. An attacker can disrupt the entire IoT network by

stealing sensitive data, flooding the network with useless messages, or stopping message transmission.
Solution: To prevent this, authentication mechanisms must ensure access only to authorized participants.
Authentication ensures that the user is who he claims to be so that it does not come to the fact that the user impersonates someone else and withdraws money from another user's account or similar.

- ***Single Point of Failure***

  Since the management of the entire system takes place on a centralized cloud server, the functioning of the IoT system completely depends on its performance. With large volumes of data arriving at the server, its operation can significantly slow down or lead to a failure in service, which as well can be caused by a DoS attack. Beside this on cloud database data can be corrupted or can be modified by anyone who is in control.
  Solution: Authorization mechanisms must be designed in such a way that only the authorized person has access to data.

- ***Security and Data Privacy***

  If there are compromising nodes in the network, then the data in the IoT network, passing from the source to the target by many nodes providing communication, can be subject to theft. Therefore we need to ensure data confidentiality and integrity. This means that the data transmitted from the sender to the recipient should not be accessible or modified by third parties.
  Solution: These requirements can be achieved by using data encryption protocols.

- ***Trustless Environment***

  As long as the data is stored on a central authority server acting as a third party, then the client-server relationship is built on trust in the service provider. Therefore, there is no guarantee that the data will not be sold or manipulated.
  Solution: This problem can be solved by a decentralized network, where data immutability is ensured by mathematical algorithms, and not by a trust factor.

- ***Energy inefficiency***

  Devices that are part of an IoT network are often limited in terms of memory, power, and charge.
  Therefore, it is important to pay attention to this type of attack, such as flooding of messages, which depletes the resources of the device.
  Solution: For proper working of the IoT network, mechanisms need to be designed to identify and eliminate or minimize flooded messages in the network.

- ***Device Security Issue***

  For peer-to-peer communication, some mechanisms must provide device authentication. This is necessary because attackers can use their devices to fake and collect data.
  Solution: There must be a specific protocol for this since devices cannot authenticate the neighbor devices themselves.
  So that the data origin can be trusted.

- ***Access Control Issue***

  In the IoT network, the data transferred is shared with all the nodes of the network.
  Solution: Protocols can be designed to restrict the sharing of data with peers. This will also help in restricting data to unauthorized access in the network.

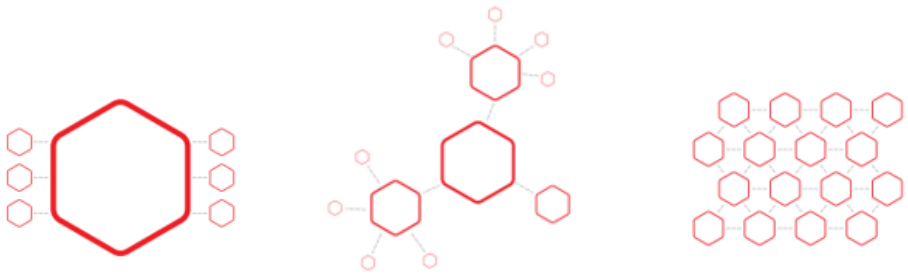
To address discussed above issues it is proposed to integrate DLT into the IoT network. According to the results of my research, of all DLTs implementations, the most applicable for IoT are blockchain and DAG. Further, I am providing a description of each of them and conduct a comparative analysis.[5][4][6][7]

Chapter 2

Distibuted Ledger Technology

In a Distributed Ledger IoT environment, DLT manages a distributed public ledger that stores the communication and transaction data of multiple parties without requiring a trusted central authority.
Everything we will talk about in the future is based on DLT, so for a start, it is worth understanding what it is.

There are three types of organization of systems in a network: centralized, decentralized, and distributed.
Figure 2.1 below illustrates these three architectures.



(a) : Centralized **(b) :** Decentralized **(c) :** Distributed

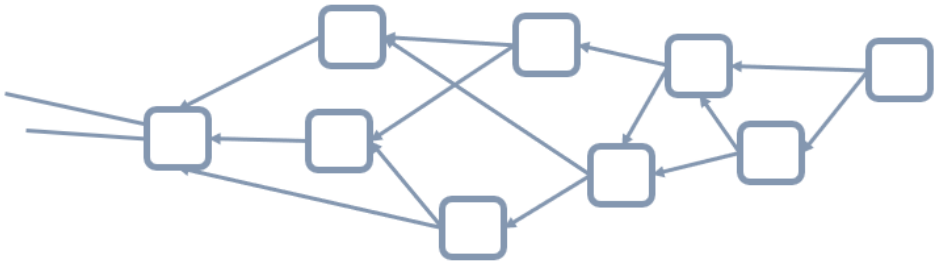
Figure 2.1: Network architectures[2]

- In the centralized architecture, one node does everything;
- On the decentralized several nodes distribute work to sub-nodes;
- On the distributed all the nodes are equal – peer-to-peer network architecture.

The concept of DLT originates from a peer-to-peer network that brought a new principle of data storage, namely distributed. Peer -to peer network creates a group of devices connected together which can exchange data with one another without the need for a central authority. Each peer in the network can serve as both a client and a server at the same time. Such a network does not have a central server on which data could be stored in the form of well-known databases, so the so-called DLT or "shared ledger" in other words

arose.

So, in general, DLT is a digital system for storing transactions in a peer-to-peer network. The data registry is stored on each node that is part of the network. This registry may have a different structure, depending on the specific implementation. For example, it can be a blockchain - that is, a chain of blocks sequentially linked to each other by a cryptographic hash.
Or if it is a DAG, then its structure consists of vertices (in our case, represented by blocks) and edges - cryptographic hashes that connect blocks, in such a way that they will never form a cyclic (closed) structure. As in the case of the blockchain, the blocks are linked by a hash, but at the same time, several previous ones can point to the next block.
Figures 2.2a and 2.2b below illustrate the difference between these structures:



(a) : Directed Acyclic Graph

(b) : Blockchain

Figure 2.2: DLT structures

DLT has several common characteristics, regardless of the specific structure of its implementation. The key ones are listed below:

- ***Immutability***

  Cryptography is used to establish immutable and secure storage in a distributed ledger. This ensures that data can't be modified or altered once it's been stored.

- ***Append only***

  Append-only distributed ledgers provide complete transactional history. This is in stark contrast to a typical database, which allows data to be changed for the sake of functionality. However, data alterations and

manipulation might occur as a result of this, whether caused by internal or external forces.

- ***Distributed***

  The distributed nature of the ledger is another important feature. Yes, the data is not stored in a single location. In most distributed ledger technologies, each peer has a copy of the ledger.

- ***Shared***

  The ledger does not belong to a single entity. It's shared by all nodes. Some nodes are responsible for having a complete copy of the ledger, while others only have the information they need to function and efficiency. [2][6]

Chapter 3

DLT and IoT integration

This section provides an overview of two distributed ledger technologies (DLTs): blockchain and IOTA Tangle. Traditional blockchain technology and IOTA both use the DLT principle, but they are built differently. It's all about the differences between those two DLTs discussed previously and how they relate to IoT applications.

3.1 Blockchain

This chapter focuses on how the IoT architecture is currently implemented and what are the shortcomings of such an implementation.

3.1.1 Introduction

Blockchain technology is a distributed ledger technology (DLT) with extremely secure properties. With cryptographic hashing, it is immutable, transparent, and decentralized. This distinguishing feature of blockchain technology positions it as a viable alternative to the costly, unsafe, and inefficient nature of existing commercial platforms. It is composed of digital data structured in "blocks" and stored in public databases referred to as "chains." The Bitcoin network is built on blockchain technology. In 1991, Haber and Stornetta invented blockchain technology to construct a time-stamped, tamper-resistant record (Haber and Stornetta, 1991). However, Satoshi Nakamoto popularized the technique in 2009 as the backbone of Bitcoin technology.
Apart from financial applications, blockchain technology has been shown to be beneficial in a variety of industries, including healthcare, manufacturing, and supply chain management.

3.1.2 Blockchain concepts

A blockchain is a series of blocks that contain chronologically ordered transactions. Each block is connected to the preceding one by the root hash of the previous block.
The following are some fundamental concepts that are helpful for the further understanding of blockchain-based technology working principles:

Transaction: Depending on the platform and application for which blockchain is used, the transaction may be any funds transfer, any event change, code execution, or data transfer.
Hash: Blockchain stores transactions and user IDs using cryptographic mechanisms. Any length input produces a fixed length output. The cryptographic function's output is completely different when the input is changed slightly, making it extremely difficult to hack. Encrypting transactions or identities means converting them to hashes. For example, MD2, MD6, SHA-0, SHA-1, SHA-2, SHA-3, RIPEMD-128, RIPEMD-160, etc. are cryptographic functions.
Block: A block in a blockchain network is a collection of peer-verified transactions. These are kept as hashes in chronological sequence. The block size is set by the blockchain platform. With enough transactions, the block is broadcast to the network for appending to the current blockchain (after consensus algorithm). Figure 3.1 displays a blockchain block's structure.

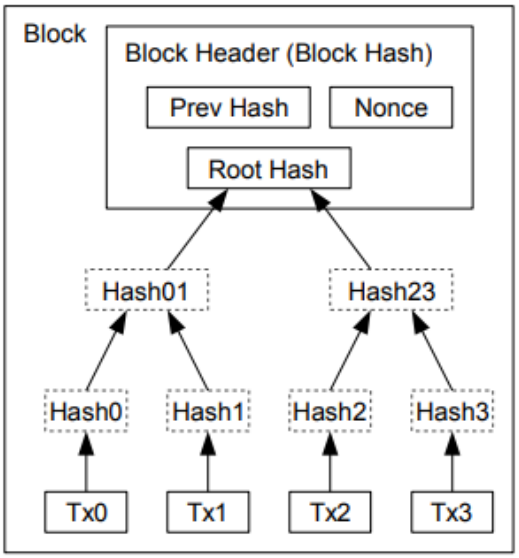


Figure 3.1: Block structure [7]

Block header: Each block has a header and a body. The block's transactions are the block's body. A block header has four parts:

- *Merkle Root Hash:* The block's root hash. As demonstrated in Figure 3.1, it is calculated by combining transaction hashes. The block's transactions are first hashed using a cryptographic method. At each level, two neighboring hashes are joined to generate one hash.
- *Preceding Root Hash:* The previous block's root hash. The blocks are linked together by the preceding root hash, producing a block chain.
- *Nonce:* A number in the block header that miners find such that the block's hash is less than or equal to the current network threshold.

- *Timestamp:* This is the block's creation time.

Mining: Creating a block involves collecting real-time transactions, verifying them from peers, turning them to hashes, and discovering the root hash and nonce. Many miners are involved in mining.
But only one of the blocks made by mines is picked and added to the blockchain. (That miner would be chosen who solved the consensus algorithm faster than others). Blocks are generally rewarded to the miner.
Peer/node: Anyone who participates in the blockchain network is referred to as a peer/node. A node might be a miner or a transaction processor.
Consensus: Consensus is a way for peers in a distributed computing system to agree on a block version to append to the blockchain. Proof of Work is the initial consensus protocol for Bitcoin. Additionally, there are other consensus protocols such as Proof of Stake, Proof of Burn, and Practical Byzantine Fault Tolerance.
Smart contract: Smart Contract is a kind of blockchain that is pre-programmed with rules and milestones to function autonomously without a third party. These contracts are maintained on blockchain and monitor real-time activities. Smart contracts are widely utilized in online voting, mortgage loans, insurance, supply chain management, and copyright protection.

3.1.3 Blockchain-based network operation concept

Blockchain is a decentralized technology. Every node in the network must be kept up to date on all network transactions.

Architecture of blockchain P2P network

Nodes may be classified into three types based on the sort of work they do.

In other terms, a node is a computer with specially installed software that knows how to communicate with the rest of the nodes in a certain decentralized network.

- ***Simple Node:*** Nodes that perform/send/receive transactions are referred to as simple nodes. They are not engaged in mining or transaction validation. Simple nodes are not capable to store a copy of the blockchain.
- ***Full Node:*** Also known as validator nodes. These nodes maintain a full copy of the blockchain and assist in verifying transactions that will be included in the next block.
- ***Miner Node:*** Miner nodes also maintain a full copy of the blockchain and mine transactions in order to generate new blocks. In a blockchain, miner nodes compete with one another to get their block acknowledged.

It's worth noting that, although the Bitcoin network technically supports three different kinds of nodes, only two are utilized (primarily). Belonging to one of these types of nodes is determined by the installed software and

the power of the computing device. Thus, every member in the Bitcoin Blockchain may act as a blockchain validator by hosting a complete (full) node.
However, the major reason for running a complete node is to improve security. Regrettably, since this is an intangible benefit, it is insufficient to motivate someone to operate a complete node. As such, Blockchain Validators are mostly comprised of miners and mining pools that operate complete nodes. To get the network on one copy of the blockchain, a solid consensus algorithm is needed.
The four general phases of blockchain operation are as follows:

1. A transaction occurs when node A sends data/money to node B. Node A sends out a transaction request to the network, which is subsequently broadcast to all nodes.

2. Validation of transaction. When the blockchain nodes receive a transaction started request, they check their copy of blockchain at the sender's address, which represents the sender's account details. The blockchain is thus both visible and anonymous, as each node is represented by an address rather than an actual identity.

3. Adding a transaction to a new block: If the blockchain nodes approve a transaction, the miner nodes add it to the new block. The transaction hasn't been added to the chain. To add a transaction to a block, miners follow the blockchain's consensus procedure. Every blockchain environment has a unique consensus process.

4. Adding block to blockchain: Multiple miner nodes create block. They must solve a puzzle to find a nonce for the block. Depending on the blockchain environment's consensus protocol, one block is picked to be added to the blockchain. To keep the network in consensus, the chosen block is published to all participating nodes. The sender A's transaction is complete once the block is append to the blockchain. The time it takes from start to finish a transaction varies between blockchain networks. The public blockchain network has a longer wait time than private or federated networks.

Validation and Blockchain Consensus

In many sources, little attention is paid to the validation stage, which will not allow you to see the full picture of how the blockchain works and how it differs from the consensus. Therefore, a more detailed explanation of the validation process follows.

It's crucial to understand the difference between "validation" and "consensus." A Blockchain Validator validates transactions by ensuring that they are legal (not malicious, double spends etc). Consensus, on the other hand, is defining the order of blocks in the blockchain and reaching an agreement on that order. Consensus entails agreeing on the sequence in which validated transactions

should be processed.

Validation

How can nodes be certain that the request is authentic and that it was issued by the message's legitimate owner?
To unlock and spend money in Bitcoin, a so-called Digital Signature is required. Through the mathematical procedure that prohibits duplication or fraud in the digital environment, digital signatures establish the message's legitimacy. Each transaction requires a unique digital signature. Digital signatures are created by combining two distinct but related keys. Private Key is used to produce the Digital Signature, whereas the Public Key is used to verify it (Private Key). Public Keys are the addresses used to send Bitcoin transactions. To spend the cash, the sender must establish that they legally control the public key address to which the payments were transmitted. The Sender does this by producing a Digital Signature from the transaction message and the Private Key associated with it.

Other nodes in the network may verify that the signature belongs to the sender's Public Key by using it in a separate function. The creation of the Digital Signature and its verification on the side of the receiver is depicted in Figure 3.2.

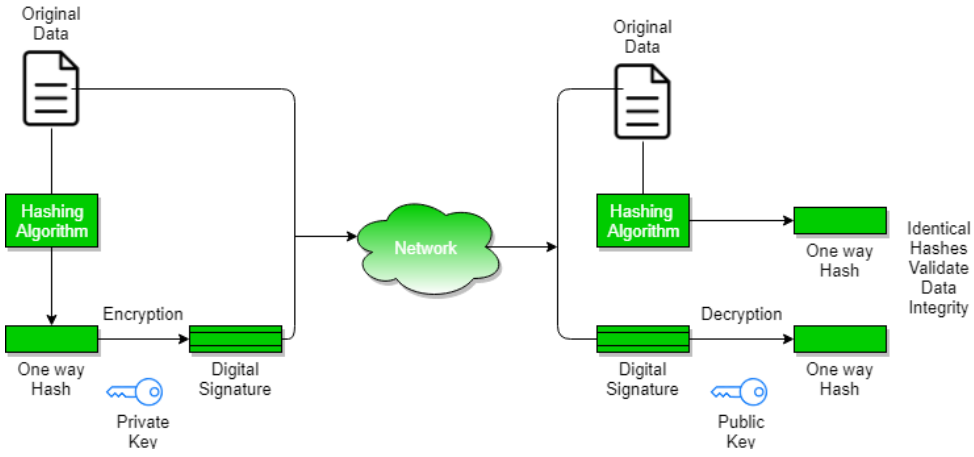


Figure 3.2: Verification and creation of the Digital Signature [11]

Nodes can verify that the sender possesses the Private Key without seeing it due to the mathematics underpinning the Digital Signature. Because the signature is based on the message, it will be unique for each transaction and hence cannot be reused for another transaction. The signature's reliance on the message also implies that the message cannot be modified.
While it is being sent via the network, as any modification to the message would invalidate the signature.

How do nodes in the network keep track of account balances?

Instead of balances ownership of funds is verified through links to previous

transactions.
For A to send 5 BTC to B, A must reference other transactions where A received 5 or more BTC. These reference transactions are called Inputs. (Outputs are funds that are going to be sent). Other nodes verifying this transaction will check those inputs to make sure A was in fact a recipient and also that the inputs at up to 5 or more Bitcoins.
The transaction that has been used once is considered spent and it can't be used again. Otherwise, someone can "double spent" input by referencing that in multiple transactions.
When verifying the transaction, in addition to the other checks nodes also make sure the inputs haven't already been spent.

To summarize the verification includes:

1. The inputs belong to the sender (proving public key ownership)
2. The sum of all inputs is greater than or equal to the amount to be sent
3. Inputs are unspent

Consensus

Consensus techniques vary according to technology but consider Bitcoin, which employs the Proof of Work mechanism and SHA-256 hashing algorithms. Proof of work needs a computer to randomly execute hashing routines until the output contains the required minimum number of leading zeroes.
The computer passes the block it wishes to add to the blockchain through the hash function, altering the Nonce value each time until the output of the hash function with a specific Nonce does not contain a hash with a certain number of zeros in front. The software determines the required number of zeros.[4][6][8][9][12]

3.2 IOTA Tangle

3.2.1 The Tangle structure

In IOTA, the Tangle is a distributed ledger that keeps track of all messages sent.

The Tangle is the only trustworthy. Any user, wherever on the planet, may transmit valid messages, previously known as transactions, to any node, and those messages will be duplicated throughout the network to establish one version of truth: The Tangle. To construct a directed acyclic network, every message in the Tangle is connected to two others - previous ones as shown in Figure 3.3.

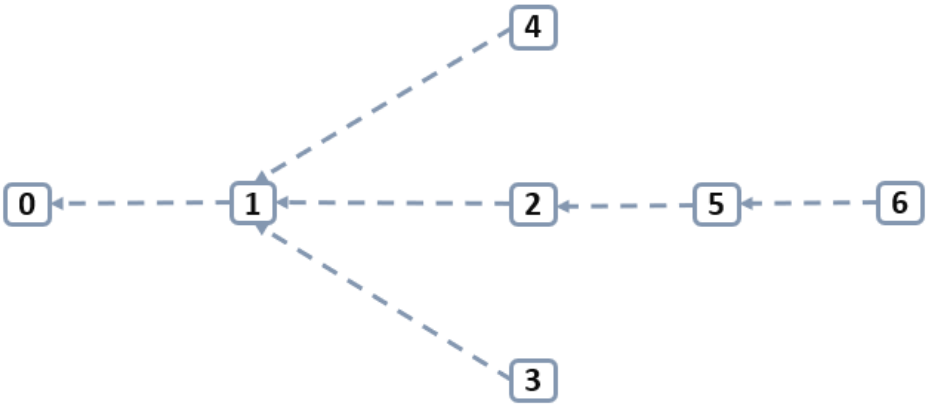


Figure 3.3: Message connections in the Tangle

A box represents each message in the graph, and a line represents each attachment. When a new message is added to the Tangle, it is connected to two previous messages, resulting in the graph being expanded by two lines.

Messages are tied to the Tangle by using the "branchMessage" and "trunkMessage" attributes to relate to other messages. (These terms are important for further understanding the work of the Coordinator using these fields to reach consensus). In conjunction with the capacity to see the history of communications, there is a notion of direct and indirect messages.

Message 5 is direct for messages 2 and 3 as it directly references them.
Message 6 is indirect for message 3 as it indirectly references it.

These references make up a message's history; for example, if the message is about a kid, the direct references are the child's parents, and the indirect references are the child's grandparents, and so on. Tangle explorers frequently provide connections to a message's parents so it's possible to trace its history backward.
Let us examine the components of Tangle in Figure 3.4, a more full picture:

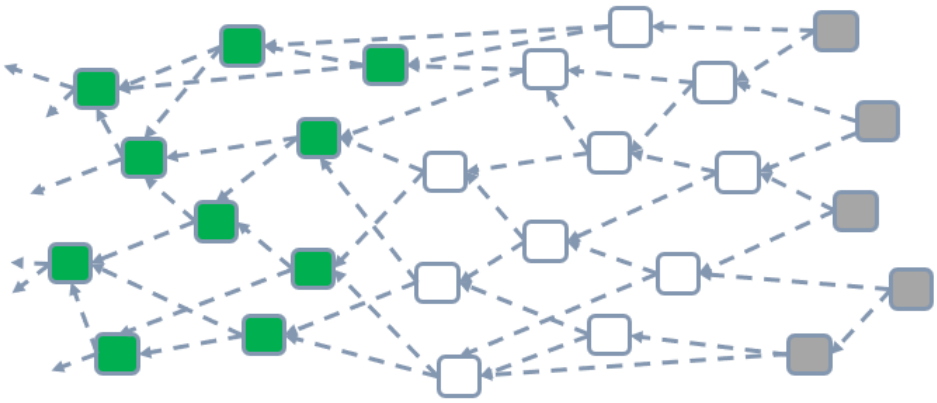


Figure 3.4: The Tangle structure

The transactions are represented as squares in this depiction of the graph, and the arrows (dubbed Edges) connecting the transactions serve as references. The older transactions are located on the left side of the graph, while the newly added transactions are located on the right side.
Transactions validate up to eight previous transactions. They can be in one of three different states:

- ***Confirmed (green):*** Transactions are confirmed if all tips reference them directly or indirectly (gray).
- ***Unconfirmed (white):*** These are pending transactions.
- ***Tip (gray):*** Tips are freshly connected transactions that haven't been referenced.

3.2.2 Sending a message at a high abstraction level

The full cycle of sending a message is shown in the Figure 3.5.

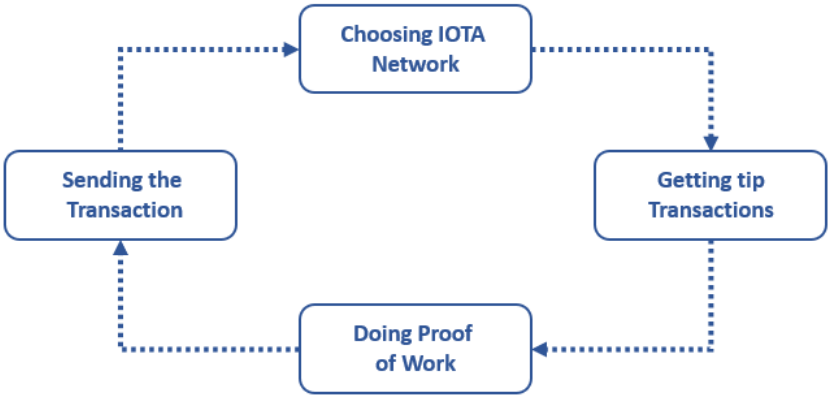


Figure 3.5: Process of message sending

The first step is to choose a network; let's have a look at the available options.

IOTA networks

Private or permissionless IOTA networks are possible:

- ***Private networks:*** Permission from the network's owner is required to access the Tangle. These networks are often operated by businesses or individuals interested in testing an application in a controlled setting.
- ***Permissionless networks:*** Because the Tangle is open to the public, all communications sent across these networks are available to everyone. These networks are comprised of nodes located around the globe. Anyone may join the network for free.

The IOTA Foundation operates the following permissionless networks, which anybody may join and operate a node on:

- ***Mainnet:*** The primary IOTA network, where the IOTA token has monetary value and is sold on platforms such as cryptocurrency exchanges.

  Instead than depending on third-party nodes to receive messages, it is recommended practice to operate your own node to have direct access to the Tangle.
  Hornet software is available for this purpose (to join Mainnet network).
- ***Devnet:*** A development network in which the IOTA token has no value other than for testing.
  Additionally, there is a community permissionless network called "Comnet".

Hornet

Hornet is a feature-rich, easy-to-use IOTA node software. It supports all node capabilities, including the Chrysalis network update.
Following are listed advantages of operating your own node:

- You get direct access to an IOTA network, rather than connecting to and trusting another node.
- By verifying messages and value transactions on the IOTA network, you contribute to the network being more distributed and robust.

Coordinator public key

Because current IOTA networks rely on the Coordinator to reach agreement, each node is hard-coded with the Coordinator's public key. Nodes use this public key, or set of keys, to validate the Coordinator's signatures in milestones.
The Coordinator is discussed in further detail later in this chapter.

Sending an IOTA message - detailed steps

The transaction procedure is condensed into five steps:

1. A transaction is created with a recipient, sender, value, and optional message.
2. The transaction is signed using the private key (seed). This verifies ownership of the made entries.
3. A node chooses up to eight unconfirmed transactions (tips) to reference in a new transaction.
4. Proof of Work is required simply for spam prevention, not consensus. Work must be done immediately, and a little math issue (nonce) must be addressed. Also, your tips are examined to verify whether the account is properly covered or if there are any conflicts.
5. The completed transaction is delivered to a node, which distributes it to all of its neighbors (other nodes). Someone else will now come and inspect your transaction in the “tip selection”, so confirm it.

The Message

In the network can be data, value or mixed transfer. IOTA supports a variety of message types. Certain messages carry value (the IOTA token or digital assets), while others merely transmit data, and other message types may include both.

This flexible message format allows decentralized data and value delivery in a single message with the utmost security and no costs. The network nodes assure the secure dissemination of all Tangle messages.

IOTA message overview

A message is an object that contains information that has been disseminated in the Tangle. Any protocol-compliant application may send these data items to a node. An Iota node’s role is to validate incoming messages and broadcast them to the network if they are genuine and meet protocol parameters.

Nodes will use the gossip protocol to transmit legitimate messages to their immediate neighbors. Every neighbor who gets the message forwards it to their neighbors. In a few seconds, every other node in the network sees the message and has the same information and knowledge of the network’s "state".

A message contains fundamental information about the message’s type and structure, as well as payloads. A payload is an attachment that may include an IOTA transaction as well as a variety of other types of information. The IOTA protocol classifies these data bundles into kinds and treats them accordingly. So, every message delivered to the network must have a unique label that identifies it and what it is for. A node will only receive and process messages

that include this information.

Creating an IOTA message

Messages are created by so-called clients. It is possible for such clients to be an IOTA wallet or any other program that sends IOTA messages. Messages are forwarded from the client to an IOTA node for processing.

There are numerous pieces of information that must be provided by the message label produced by a client before the message can be processed and entered into the network in order to guarantee that the message is genuine and that a node understands what to do with it.

- ***Message ID***
  The message ID is generated as a one-of-a-kind cryptographic hash of the message's bytes. The client (application) or wallet that sends the message generates it. Or the node in case of the client request. But not whole nodes support this.

- ***Network ID***
  An indication of which IOTA network the message belongs to (Mainnet / Testnet / private network) - Nodes will only receive messages that identify themselves as belonging to the network to which the node belongs.

- ***Parents length and Parents ID***
  This is the quantity and identification of the messages referred to by the new message. Every new message in the Tangle must reference 2 - 8 prior messages in order to develop the Tangle's graph structure. The node chooses those two messages and provides the IDs to the client, who must include this information in the message's "label". As a result, nodes ensure that the Tangle's data structure changes in accordance with the protocol.

- ***Payload length***
  Because messages in IOTA cannot be larger than 32kb, the message must indicate the amount of its payload to the node.

- ***Payload type***
  A description of the sort of payload included in the message. The node must be aware of this since certain payload types must be treated differently than others.

- ***Nonce***
  That is the nonce that allows this message to satisfy the Proof-of-Work requirement. *Proof of work is mostly performed locally on the device that sends the message and serves as a kind of spam defense. However, if the node permits it, PoW may be performed by the node rather than the client.* This is a useful feature since it allows extremely low-powered devices (such as sensors, chips, and so on) to send messages without performing the PoW in the local device. When such low-power devices are linked to

a node that supports remote PoW, they may transmit messages while the node (which is normally operating on a more powerful device) executes the PoW for them. This is one of the reasons IOTA is so well suited to IoT and data applications. Users wishing to transmit a big number of data messages from a large number of extremely low-power devices should just connect those devices to a node capable of conducting the PoW on their behalf (which will be, in most cases, their own node). Because the PoW requirement in IOTA is so low in general, it is possible to achieve this even for a large number of devices by a node.

Structure of the message

The message structure is depicted in Figure 3.6. All further data must be provided by a client.

Name	Type	Description
NetworkID	uint64	Network identifier. This field will signify whether this message was meant for mainnet, testnet, or a private net. It also tells what protocol rules apply to the message. It is the first 8 bytes of the `BLAKE2b-256` hash of the concatenation of the network type and the protocol version string.
Parents' length	uint8	The number of messages we directly approve. Can be any value between 1-8.
Parents	ByteArray[32 * `parents length`]	The Message IDs that are referenced.
Payload Length	uint32	The length of the payload. Since its type may be unknown to the node it must be declared in advance. 0 length means no payload will be attached.
Payload	▾ Generic Payload An outline of a general payload	
Nonce	uint64	The nonce which lets this message fulfill the Proof-of-Work requirement.

Name	Type	Description
Payload Type	uint32	The type of the payload. It will instruct the node on how to parse the fields that follow.
Data Fields	ANY	A sequence of fields, where the structure depends on `payload type`.

Figure 3.6: Message structure [12]

Message verification

If the following syntactic rules are fulfilled, a message is deemed valid:

- The message cannot be larger than 32 KiB (32 * 1024 bytes).
- Analyzing the message's syntax structure (parsing) leaves no unknown bits, indicating that all message information is completely accessible by

the node. Unreadable data may contain harmful code and is therefore prohibited.

- If the node is aware of the payload type.
- If the Message PoW Hash shows that the network's or the node's minimum PoW criteria have been met.
- The number of parent messages must range between 1 and 8.
- Only if these criteria are satisfied and the message is readable by the node will it be approved for processing.

Payloads A payload may be included in a message.

The Mainnet specifies three payload types at the moment, but developers are free to create their own and attach them to messages provided they match certain conditions. The IOTA token used as the transaction payload is only one of the numerous types of data that an IOTA message may include.
The following table summarizes the presently stated core payloads:

- A transaction payload
- An indexation payload
- A milestone payload

A communication having just an indexation payload (Data) may be sent without a signature. It may hold any data that the user chooses to convey, as long as it is parsable and complies with the required syntax and size constraints.

An index is used to characterize the message, which enables any user to find the message and the Data it contains by searching the network for this index. A message sent through the IOTA network does not need a specific receiver.

All network communications are broadcast to all nodes and are thus accessible to all network users. Additionally, the data payload is available to all message receivers (if the sender does not encrypt it). The IOTA Streams framework is designed for transmitting data messages via the IOTA protocol to a limited number of receivers. It will establish a direct connection with receivers and encrypt data from the rest of the network.

Anyone who comprehends the index of a data message, which is supplied as an indexation payload, is capable of quickly locating it. If you desire to convey an arbitrary message or sensitive data to a recipient, you must tell them of the index you are using. Recipient may then use an explorer to scan the network for any messages that have this index.

The payment receiver - an IOTA address - will be specified in a value transaction's "signed transaction payload" field. The funds may then be utilized only by the owner of that specific address by unlocking them using the receiving address's private key.

The Coordinator

Consensus in the IOTA network is done with help of the Coordinator.

The Coordinator is a client that provides signed messages known as milestones that nodes rely on to confirm messages. Furthermore, how nodes utilize milestones to decide whether messages are verified is discussed. The Coordinator is only in place for a limited time. IOTA Foundation planning to remove the Coordinator in the next network update: *Coordicide*. Current network version 1.5 is named Chrysalis.

The Coordinator's current role within IOTA is outlined below.

Nodes only confirm messages if they are referenced by a validated milestone. All IOTA nodes in the same network are preset with a trustworthy Coordinator's Merkle root address. Nodes may use this address to verify milestone signatures from their trusted Coordinator. The Coordinator sends out indexed milestones at regular intervals to keep track of fresh communications. Nodes may check their milestone indexes against the network's to see whether they are synchronized.

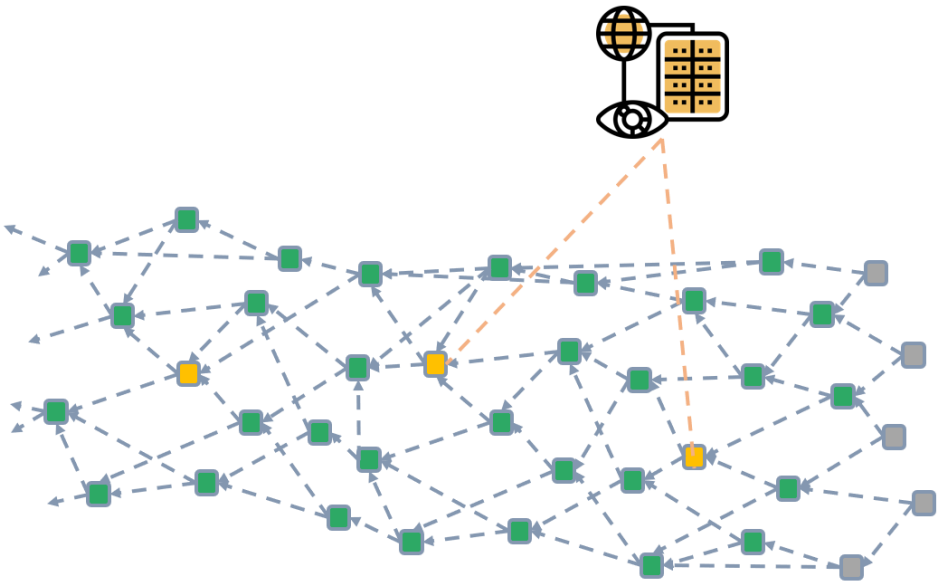


Figure 3.7: The Coordinator

The Coordinator transmits milestones in the same manner as regular messages, with one exception: *the milestone's past cone tips are utilized for confirmation.* (Past cone - a collection of transactions in the Tangle that are directly or indirectly referenced by a child transaction, including the child transaction).

This past cone comprises any pending messages that the tip messages mention. As a result, the confirmation cone is named after it as confirmation cone.

In case of duplicate spends, nodes and coordinator agree on which message should be verified by ordering the confirmation cone.

Ordering the confirmation cone If the confirmation cone results in a double expenditure, nodes and the Coordinator agree that the first message should be verified and the rest disregarded.

It is possible to arrange messages in Tangle in many ways. Using depth-first search, nodes and Coordinator agree to arrange the confirmation cone as follows:

1. Begin with the milestone;
2. Iterate over the trunk messages until the first verified message is located;
3. Include the most recent message in the list;
4. Retrace your steps through the remaining confirmation cone messages, using the same technique that prioritizes the trunk message first.

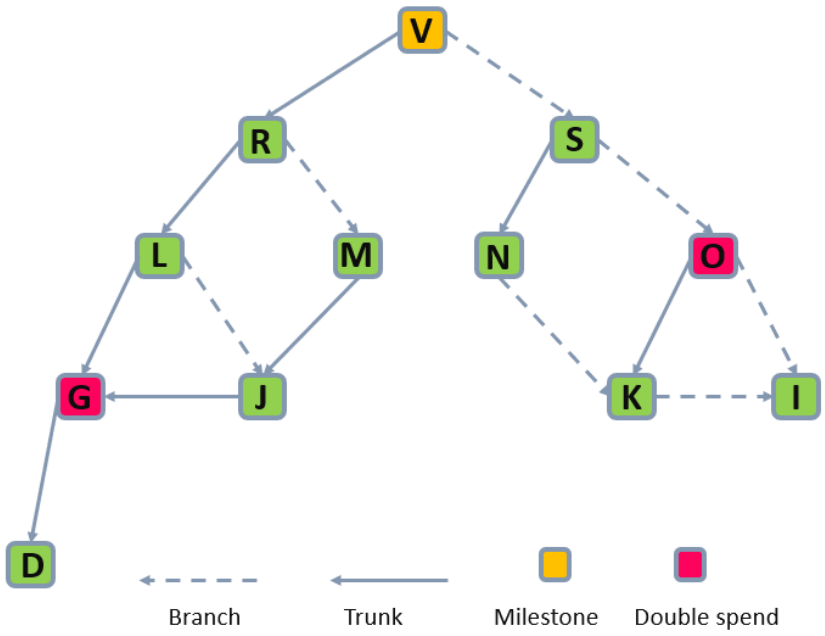


Figure 3.8: The confirmation cone

Follow the trunk down to message D to order the confirmation cone. The confirmation cone's initial message. Then you work your way back up, reaching the ultimate order. Thus, beginning with V, the messages are organized as follows: D G J L M R I K N O S V
As a consequence, since message G precedes messages O, milestone V verifies message G.

Recording the confirmed messages

To enable nodes to determine which messages in the confirmation cone have been verified by a milestone, it provides the Merkle root of the confirmed tail messages.

Constructing the bundle

After building a Merkle root, the Coordinator is ready to include all of this information in its milestone bundle for validation by nodes.

To begin, the Coordinator signs the hash of the bundle and appends the signature to the head messages. The Merkle root is then appended to the signature by the Coordinator.

The Coordinator then signs the Merkle root to establish its ownership, and this signature is then appended to the milestone bundle's other zero-value messages.

Validating milestones

Nodes must do the following to verify milestones:

1. Verify signatures to ensure that the Coordinator really submitted the milestone.
2. Verify the Merkle root to ensure the Coordinator does not confirm duplicate expenditures.
3. Nodes do the same depth-first search as the Coordinator to verify the Merkle root.
4. Thus, nodes may generate their own Merkle root and compare it to the one specified in the milestone. The milestone is legitimate if both Merkle roots match. If not, the node invalidates the milestone.

3.3 The primary distinctions between IOTA's Tangle and a Blockchain

After considering the operating principles of both technologies, which helped to intuitively understand the primary differences between the Blockchain and DLT, an overview of all the differences is provided for greater clarity, following which we can conclude that using one or the other technology is more beneficial.

1. The first significant distinction is in the structure.

   Blockchain

   In a blockchain, each block has a set number of messages. As a result, Validators can only add messages to the end block. These limits slow

blockchain network confirmation times. This causes the blockchain bottleneck.

IOTA Tangle

Each message in the Tangle data structure is connected to two to eight preceding messages. Rather than being restricted to a single location, you may connect messages to the Tangle in several locations. The protocol may handle many attachments simultaneously. Parallel processing alleviates congestion.

2. The second significant distinction is consensus.

   Blockchain

   Participants in blockchains are validators (miners) and clients. Miners need a lot of computer power to link the blocks together.
   The following motivates miners to validate messages:

   - Users' willingness to pay a charge to have their transactions included in a block;
   - A reward (Bitcoin) is given to the miner who solves the mathematical challenge first. As a consequence, mandating validators to perform PoW strengthens blockchain networks by making them more difficult to hack, alter, or shut down. The more miners that participate, the safer the network becomes.

   IOTA Tangle

   Messages are free since Tangle has no miners. PoW does not secure the Tangle. Instead, PoW discourages spam. To achieve consensus, all IOTA nodes evaluate messages and use various functions alongside messages. Currently, transactions will only be deemed valid if they reference a milestone. The Coordinator establishes these milestones.

3. Transaction fees are eliminated since in the IOTA are no miners.

   Blockchain

   In Bitcoin, clients must pay miners to get their transactions included to the ledger.

   IOTA Tangle

   In IOTA, every network member that does a transaction also actively participates in the consensus process. Because the sender must do Proof of Work for two to eight preceding transactions in order to add a transaction to the tangle.

4. IOTA Tangle is quantum resistant.

Quantum computing threatens many current encryption protocols.

Cryptographic techniques safeguard a blockchain.

Private/public key pairs protected by asymmetric encryption algorithms like RSA or Elliptic Curve (EC) cryptography are generated on blockchains.

It is expected that by 2035 quantum technology will have matured enough to efficiently break RSA2048. The US National Institute of Standards and Technology (NIST) has began the process of standardizing and deploying quantum-safe public-key encryption.

The signature system of networked transactions is susceptible to quantum attacks. The major type of attack found is against unincorporated transactions. A quantum attacker can determine a transaction's private key using the sender's public key. They can then replicate the transaction with any desired output location.

Bitcoin's system is secured via the use of a technique termed an "elliptic curve signatures scheme".

IOTA does not use elliptic curve encryption (ECC), preferring instead to utilize hash-based signatures. This is distinct from Bitcoin in that it makes the protocol more resistant to quantum computers.

IOTA is immune to quantum computing because of using "Winternitz signatures".

3.4 The advantages of IOTA technology versus Blockchain technology

To address this point, let us examine the disadvantages of blockchain technology.

The blockchain's shortcomings will be demonstrated through the use of Bitcoin technology.

Bitcoin, which is based on blockchain technology, is confronted with the following issues:

1. Scalability

   The Bitcoin blockchain takes around ten minutes to confirm a transaction, which is far too long for a system designed for microsystems. At any given moment, there are around 200K unconfirmed transactions. We want M2M, or machine-to-machine communication, to send millions of transactions each second. For example, Visa supports 2000 transactions per second. Bitcoin transacts at a rate of 5-7 transactions per second. In Bitcoin, transaction speed reduces as the network grows in size.

2. Fees

   In the Bitcoin network, the average transaction cost is one dollar. A dollar is a large sum of money for a network designed to facilitate micropayments. As a result, micropayments of less than a dollar are essentially meaningless.

3. Lots of computing power required to maintain the Bitcoin blockchain

   Lots of computing power required to maintain the Bitcoin blockchain When miners perform Proof of Work, they consume a significant amount of power, yet in the end, only one miner performed valuable work, as only one miner's block is added to the blockchain. That means, all of the other miners power was squandered, which is a significant downside of the Bitcoin system.

4. Vulnerable to Quantum Attack

3.4.1 Final conclusion of technology adoption

Taking into consideration the aforementioned shortcomings of blockchain-based technology, we can infer that DLT based on DAG, namely IOTA Tangle, is a better fit for IoT integration, since it addresses all of these issues.[7][9][13]

Chapter 4

Security

End-to-end security is required for IoT devices, from the sensor to the distant location where data is stored and processed. End-to-end security challenges are critical and typically determine solution selection.

Transport Layer Security (TLS) is widely used to safeguard data transfer in sensor systems (TLS). TLS is a robust and adaptable secure protocol for device communication. A secure channel enables mutual authentication, secrecy, and integrity of data transferred.

TLS is used to create secure communication between the sensor and the Edge or Cloud device. Because an IoT device cannot open more than one TLS channel, any scenario involving multiple data analysis points must be handled at the Edge or Cloud level. TLS drains resources on IoT devices. The Tangle is meant to store data with a transaction of a specific message with an indexation payload.

A transaction with an indexation message binds data to Tangle. It is retrieved by any node eager to consume it. In order to structure data across the Tangle and allow complicated data stream transfer and easy data retrieval, a Layer 2 protocol (L2) (on top of IOTA Layer1) has been implemented. An L2 protocol of this kind is a cryptographic protocol that protects data transfer across the Tangle from end to end.

Securing data transfer across the Tangle requires an L2 protocol. In addition, it allows you to encrypt and decrypt data streams, as well as convenient access and consume data streams while confirming their origins, ownership, and authenticity. The Tangle ensures data integrity and immutability. Using an L2 cryptographic protocol with the Tangle allows for multi-point to multi-point safe data transmission. This combination allows dispersed sensor systems to safely share data in near real-time.

Heterogeneous and independent systems may communicate without re-designing and developing new unique data exchange interfaces. Any data source may offer the other device access to its data stream.

IOTA has two L2 solutions. Masked Authenticated Messaging (MAM) and STREAM. The MAM uses an obsolete IOTA Tangle. STREAM replaces

MAM and works with the new Chrysalis version of the IOTA Tangle.

STM's X-CUBE-IOTA1 expansion software package contains L2Sec, an IoT-friendly cryptography protocol. L2Sec allows a constrained IoT device to arrange data streams via Tangle and enable secure data exchange. L2Sec is a cryptographic technology for structuring, securing, and navigating Tangle data. Moreover, L2Sec additionally uses a Hardware Secure Element (STSAFE A110) to provide a HW root-of-trust at the IoT device, enhancing total system security with a secure-by-design approach.

4.1 Considered architecture

The architecture used by IoT target devices to connect with one another and with servers is shown in Figure 4.1. Each IoT device connects to the IOTA Tangle through an IOTA node that acts as a gateway for the distributed ledger.

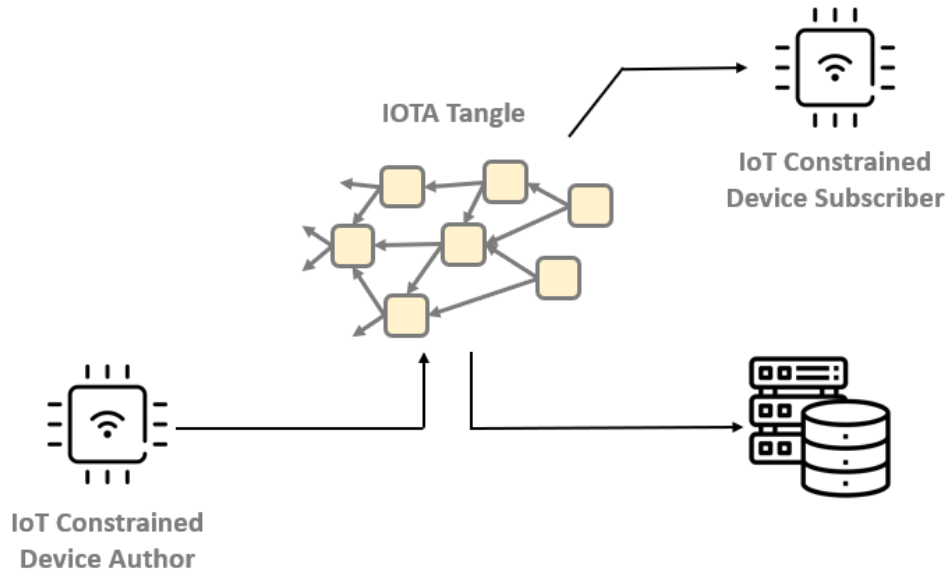


Figure 4.1: Highlevel system architecture

The IOTA Tangle is considered a safe way to send data acquired by sensors on target devices. Other network nodes (servers, sensors, etc.) may ingest data from the IOTA Tangle. Using the DLT allows for secure point-to-point, point-to-multipoint, and multipoint-to-multipoint communication between devices and servers.

4.2 Securing Data over the Tangle

To structure and traverse data via the Tangle, the IOTA Foundation created two L2 protocols.

4.2.1 Masked Authenticated Messaging

Masked Authenticated Messaging (MAM) is compatible with the IOTA Tangle's older (legacy) version. MAM allows any device to publish to Tangle. So only authorized devices can read, reconstruct, and consume a data stream safely attached to the Tangle.

The legacy IOTA Tangle introduced zero-value transactions concept to write data to the Tangle. The IOTA L1 protocol handles data transactions, however they are neither secured nor validated. In order to decrypt and authenticate data streams sent across Tangle, MAM is used as an L2 protocol.

The MAM introduced data channels. Devices allowed to receive and consume the channel's data may detect malicious attempts to write false data or seize control of the channel. MAM channels are thus a simple way to check a device's data dissemination. If device publish data to a channel, it get a channel ID, which other devices can use to subscribe and get data. Any data on a channel includes the address of the following data on the channel.

Public, private, and restricted channels exist. STREAMS has replaced the MAM protocol, so these changes will not be detailed in depth here. Good to know he recognized the present Chrysalis benefits.

4.2.2 Streams

STREAMS is a new IOTA Foundation protocol. It's a way to structure and navigate secure data in the Tangle.

To ensure the data structure's integrity and immutability, STREAMS allows any device to organize messages (data) into a uniform and interoperable structure. A Publisher device can publish messages in a Stream for everyone to see or restrict access to messages using public key encryption.

Other devices, called Subscribers, can subscribe to a Stream and pull data from it. Subscribers can also contribute messages to a Stream by cross-referencing. Unlike MAM, where only the channel owner could post unsigned messages.

With the new linking mechanisms, each message can be linked to another, allowing for more flexible data structures than MAM. STREAMS reformulates the message types. STREAMS introduces many types to make it easy to publish several messages on a single channel and identify them by their headers. Moreover, STREAMS improves channel access control by allowing multiple cryptographic mechanisms to be applied to each message type to provide various access control rules.

4.3 L2Sec—A Cryptographic Protocol for Internet of Things Constraints

L2Sec is a lightweight cryptography technique for the IOTA Chrysalis Tangle. It's designed to fit on IoT devices. For example, the L2Sec protocol employs the indexation payload of a Chrysalis message. There's an index and some odd data in the indexation payload. It encapsulates any L2Sec protocol message. L2Sec creates a data stream as a Tangle chain. Indexes link the data in the stream. By reading at arbitrary indexes over the Tangle, subscribers can reconstruct it. Every data message contains the current and next message's indexes. The L2Sec protocol uses the IOTA Chrysalis Tangle's native binary encoding, removing the need to convert data to and from trits (i.e., ternary data representation, as it is implemented in MAM). For extra protection, L2Sec combines Authenticated Encryption with Associative Data (AEAD). Due to L2Sec's use of Sodium's IOTA client(for cryptographic functions), no other libraries are required. The L2Sec protocol design allows for cryptographic operations to be performed on hardware. An IoT device's electronic identifier can be derived from the secure element's hardware Root-of-Trust.

4.3.1 Operating and Security Principles

A L2Sec message is wrapped within an IOTA Chrysalis message's indexation payload. Figure 4.2 illustrates the structure of an IOTA Chrysalis message and an L2Sec message.

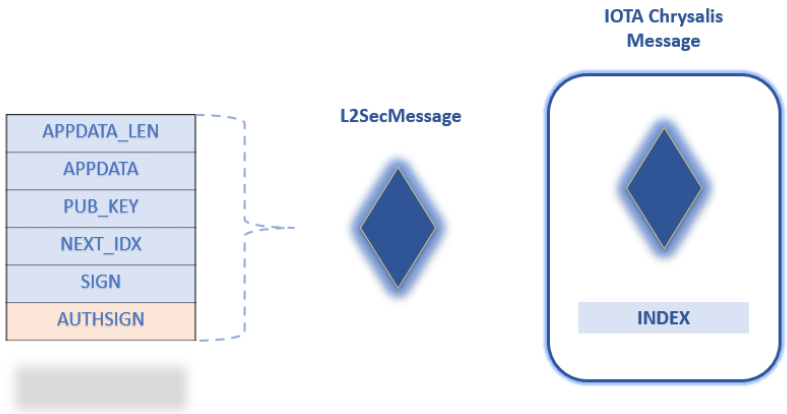


Figure 4.2: Structure of the fields that make up an L2Sec message on the left and an IOTA Chrystalis message on the right

Message Chaining

Higher level protocols or apps using L2Sec wrap their data in the APPDATA field and its length in the APPDATA LEN field. For data transfer that exceeds the maximum length of a single L2Sec message, a data sequence must

be chained. The NEXT IDX field provides the index of the next message in the stream to seek for in the Tangle. This is seen in Figure 4.3.

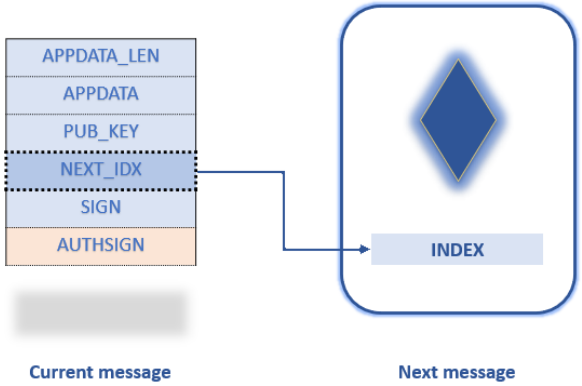


Figure 4.3: Sequencing of L2Sec messages

A single message connection allows each subscriber to only read one direction. This is done to prevent previous data (messages belonging to the same data stream) from being obtained.

Figure 4.4 depicts the INDEX and NEXT IDX creation methods. L2Sec generates a secret and public key from a random seed. The key-pair is based on the Edwards25519 curve. The public key's hash function then determines the message's index. Each L2Sec packet contains the next index (NEXT IDX). The NEXT IDX is determined using a different key-pair.

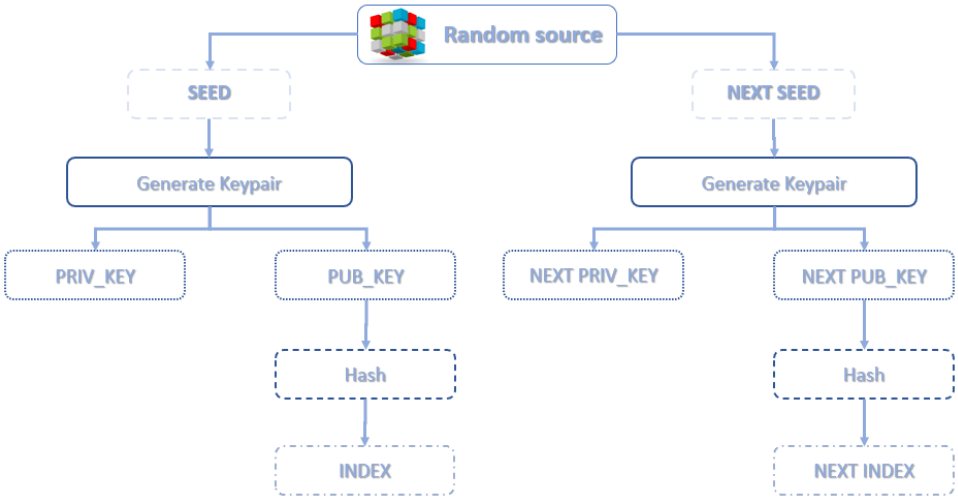


Figure 4.4: Index and Next Index generation

Data Ownership

Each message includes the field SIGN, which is derived by signing a digest h with the key pair's private key (PRIV KEY), as described in Equations 4.1 and 4.2.

$$\mathrm{h} = H(\mathrm{APPDATA\_LEN} + \mathrm{APPDATA} + \mathrm{PUB\_KEY} + \mathrm{NEXT\_IDX}) \quad (4.1)$$

$$\mathrm{SIGN} = signature(h|\mathrm{PREV\_KEY}) \quad (4.2)$$

A subscriber who wants to verify the message recalculates the hash and compares it to the signature using the public key (PUB_KEY) included in the message. It also double-checks that the PUB_KEY's hash matches the message's index. Figure 4.5 displays data verification using L2Sec message fields.

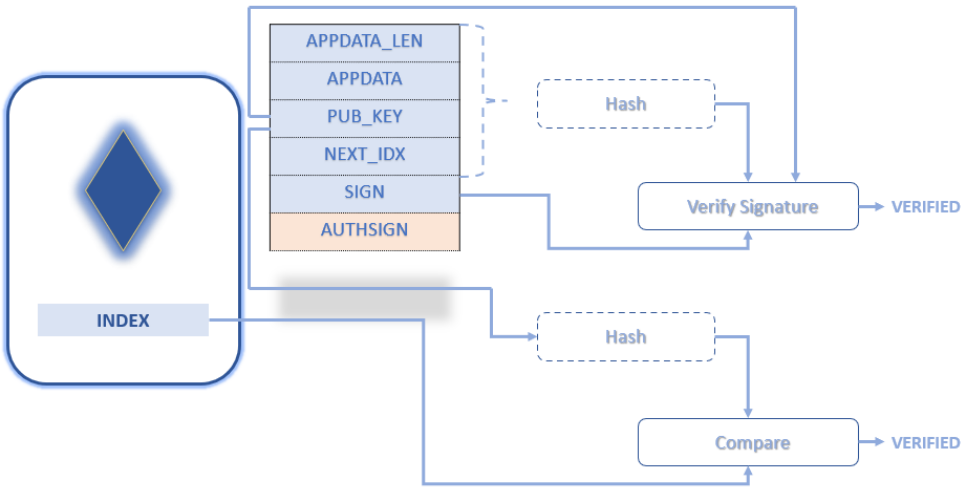


Figure 4.5: Index and Next Index generation

To divert the next message to another malicious stream, an adversary would need to find the next index's public key and insert it in its message. It is impossible for the receiver to utilize the found NEXT_IDX to attach its message chain since it does not know the key-pair used to generate the NEXT_IDX. Basically, this design enables a subscriber to confirm the data source. For integrity signature, L2Sec uses EdDSA (Edwards-curve Digital Signature Method) over the edwards25519 elliptic curve with the BLAKE2b hashing algorithm.

Authentication

The SIGN field does not give author authentication since the key-pair used is just for message chaining. L2Sec needs an extra key-pair tied with the electronic identification of the IoT device to confirm and validate the data source's identity. So L2Sec adds an Authentication Signature (AUTHSIGN) field to identify and authenticate the sender. Using the private key, this

signature authenticates all other fields of the L2Sec transaction. The private key and its public key certificate might be kept in a hardware secure device. To verify the source of the content, a subscriber must use the source's public key, which has been validated by a trustworthy Certification Authority. Figure 4.6 depicts the whole L2Sec message.

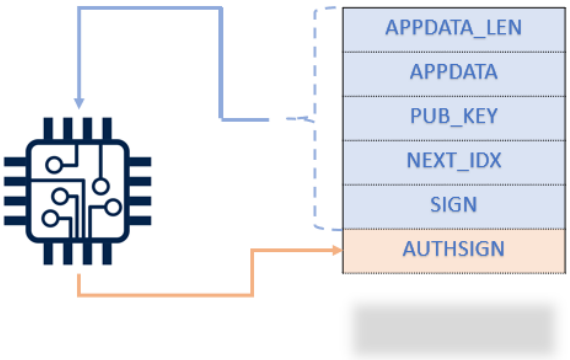


Figure 4.6: Message L2Sec with Authentication Signature produced by a Hardware Secure Element

Encryption

To protect the data in the Tangle, every L2Sec message is encrypted as seen in Figure 4.7. Using a symmetric cryptographic key and a nonce as an initialization vector, the encryption is carried out. The encryption key is pre-shared (PSK) between the message author and the subscribers. The XSalsa20 cipher is used to encrypt 's entire L2Sec message.

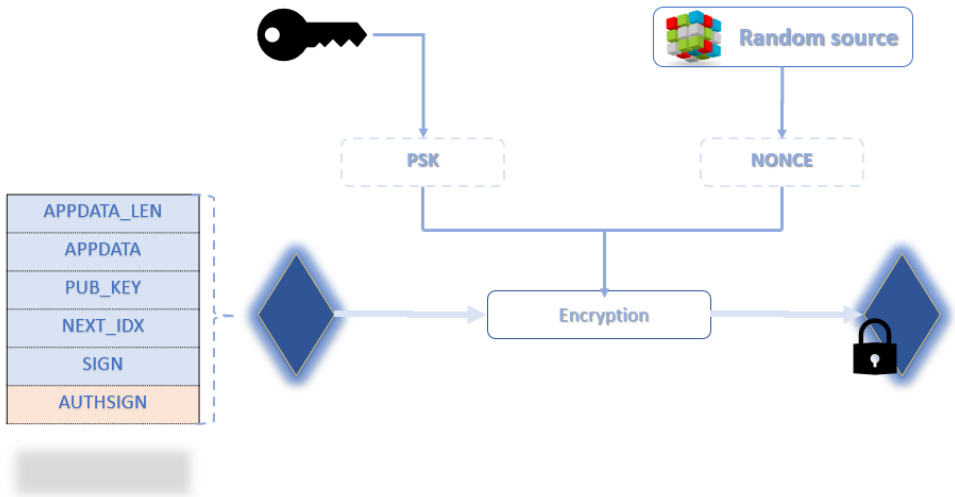


Figure 4.7: Encryption of an L2Sec message

The L2Sec message is now encrypted and ready to be despatched and

anchored to the Tangle. The final message is encased in a Chrysalis indexation payload and consists of the encrypted L2Sec message together with the nonce used for encryption. The preceding theoretical material was provided to provide sufficient background knowledge to comprehend the software solution described in the following chapter and to appreciate why, in light of modern IoT technology requirements, it is advantageous to integrate DLT technology with traditional IoT.[13][15][17]

Chapter 5

Methodological contribution

The practical section is based on software produced by STMicroelectronics; at the moment, this is the only software that enables complete interaction with IOTA Tangle using constrained devices based on STM32 controllers.

Prior to this, the majority of current projects were developed for less versatile controllers, such as the ESP32 and ESP8266, and as a result, their software did not incorporate all of the capabilities provided by the protocol for connecting with the IOTA Tangle.

This section discusses how the Tangle communication protocol is implemented practically, at the level of written code.

What steps should a developer take to effectively construct a client capable of communicating with the proposed distributed network on the basis of any STM32 controller? This question will be addressed in the chapter below within the scope of the methodological contribution.

5.1 Hardware

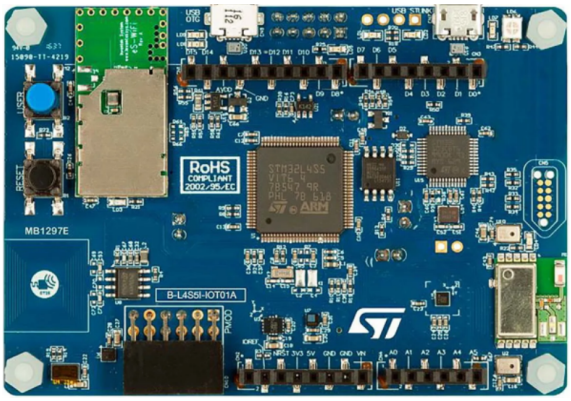


Figure 5.1: B-L4S5I-IOT01A Discovery kit for IoT node [13]

Key Product on board:

- Ultra-low-power STM32L4+ Series STM32L4S5VIT6 microcontroller based on the Arm® Cortex®-M4 core with 2 Mbytes of Flash memory and 640 Kbytes of RAM in the LQFP100 package
- Bluetooth® 4.1 module (SPBTLE-RF) from STMicroelectronics
- 802.11 b/g/n compliant Wi-Fi® module (ISM43362-M3G-L44) from Inventek Systems
- Dynamic NFC tag based on ST25DV04K with its printed NFC antenna
- 2 digital omnidirectional microphones (MP34DT01) from STMicroelectronics
- Capacitive digital sensor for relative humidity and temperature (HTS221) from STMicroelectronics
- High-performance 3-axis magnetometer (LIS3MDL) from STMicroelectronics
- 3D accelerometer and 3D gyroscope (LSM6DSL) from STMicroelectronics
- 260-1260 hPa absolute digital output barometer (LPS22HB) from STMicroelectronics
- Time-of-flight and gesture-detection sensor (VL53L0X) from STMicroelectronics
- Highly secure solution (STSAFE-A110) from STMicroelectronics
- 2 push-buttons (user and reset) Flexible power-supply options: ST-LINK, USB VBUS, or external sources
- On-board ST-LINK/V2-1 debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port

5.2 Software

This section describes the X-CUBE-IOTA1 software extension for the STM32Cube.

The X-CUBE-IOTA1 extension software package for STM32Cube comprises middleware to allow the IOTA Distributed Ledger Technology (DLT) functionality.

The X-CUBE-IOTA1 software package augments the capability of STM32Cube with the following important features:

- Middleware libraries containing:

  - Wi-Fi administration
  - Transport-level security (MbedTLS)
  - IOTA Client API for Tangle interaction
- Complete driver for developing applications that access motion and environmental sensors
- Example to illustrate how to construct and send to the Tangle an encrypted authenticated stream based on L2Sec
- STM32Cube facilitates simple porting across several MCU families

The software extension enables the IOTA DLT on an STM32 microcontroller by adding the necessary middleware.

5.3 Chrysalis – IOTA version 1.5

Chrysalis is the name of the protocol upon which the X-CUBE-IOTA1 extension software package is based.

It is essential to comprehend that the IOTA protocol has its own growth path, which begins with the usage of a centralized node - "Coordinator" - for transaction validation and ends with a completely decentralized network. Consequently, it is possible that the version that worked a year ago is no longer supported. Nevertheless, it is important to note that the transfer to Chrysalis was the final major transformation.

The transition from IOTA 1 to IOTA 1.5 is not simple. Developers must restructure their apps and move their tokens in order to use Chrysalis. But at the same time I t will make the adoption of IOTA 2.0 much easier. So Chrysalis is an interim upgrade before going on to Coordicide, a completely decentralized network.

According to the IOTA Foundation, the new Chrysalis protocol update is 60 percent more energy-efficient than IOTA's initial implementation. This is why Chrysalis is already so appealing to developers. To accomplish these advantages, the Foundation modified its protocol to use atomic transactions, which record state changes more effectively than "account-based models" that need the updating of the whole account balance. These atomic transactions may be as short as 275 bytes per transaction, as opposed to the initial 3500 bytes. A refined method for selecting "tips" expedites transaction validation and synchronization. According to the IOTA Foundation, its systems can complete 600 million transactions using the same amount of energy as one Bitcoin transaction.

Consider the substantial enhancements made between versions 1.0 and 1.5. (Chrysalis):

- IOTA 1.0 has an unorthodox implementation strategy based on trinary representation: each IOTA element is defined using trits = -1, 0, 1 rather than bits and trytes of 3 trits rather than bytes of 8 bits. A tryte is represented as an integer between -13 and 13, encoded with the letters A through Z and the number 9.
- IOTA 1.5 (Chrysalis) substitutes a binary structure for the trinary transaction architecture.

The primary enhancements added by Chrysalis are:

- Reusable addresses: the introduction of the Ed25519 signature scheme, which replaced the Winternitz one-time signature system (W-OTS), enables users to securely transfer tokens many times from the same address;
- No more bundles: IOTA 1.0 uses bundles to make transfers. Bundles are groups of transactions linked by a root reference (trunk). With IOTA 1.5, the former bundle format is gone, replaced with Atomic transactions. The Tangle vertex is represented by the Message, a container for payloads;
- UTXO model: Originally, IOTA 1.0 tracked individual IOTA tokens using an account-based model: each IOTA address stored a number of tokens, and the total number of tokens from all IOTA addresses equaled the entire supply. (This is the same concept as Bitcoin.) Instead, IOTA 1.5 employs the unspent transaction output model, or UTXO, which is based on the concept of monitoring unspent token amounts through a data structure called output.
- Up to 8 Parents: In IOTA 1.0, you could only reference two parent transactions. Chrysalis introduces a higher number of referred parent nodes (up to 8). At least two distinct parents should be used at the same time to get the greatest outcomes.

Now that we have a broad understanding of the software implementation of the IOTA protocol, let's move on to its practical examination.[17][18][**?**]

5.4 Initial use of software

I used STM32CubeIDE, thus all descriptions will pertain to this environment. **Build and run the project** The project is structured such that there are three different build configurations:

- L2SEC to run the respective example application relying on STASAFE security services and libsodium crypto support;
- CRYPTOLIB to include the cryptographic middleware by ST;
- SODIUM to include the cryptographic middleware by libsodium;

I used L2Sec since a portion of the theory is based on this protocol.

In the .cproject file are the next configurations:

```
<configuration configurationName="SODIUM" />
<configuration configurationName="CRYPTOLIB" />
<configuration configurationName="L2SEC" />
<configuration configurationName="Debug">
```

To build the project successfully, the L2Sec configuration must be set to "active":

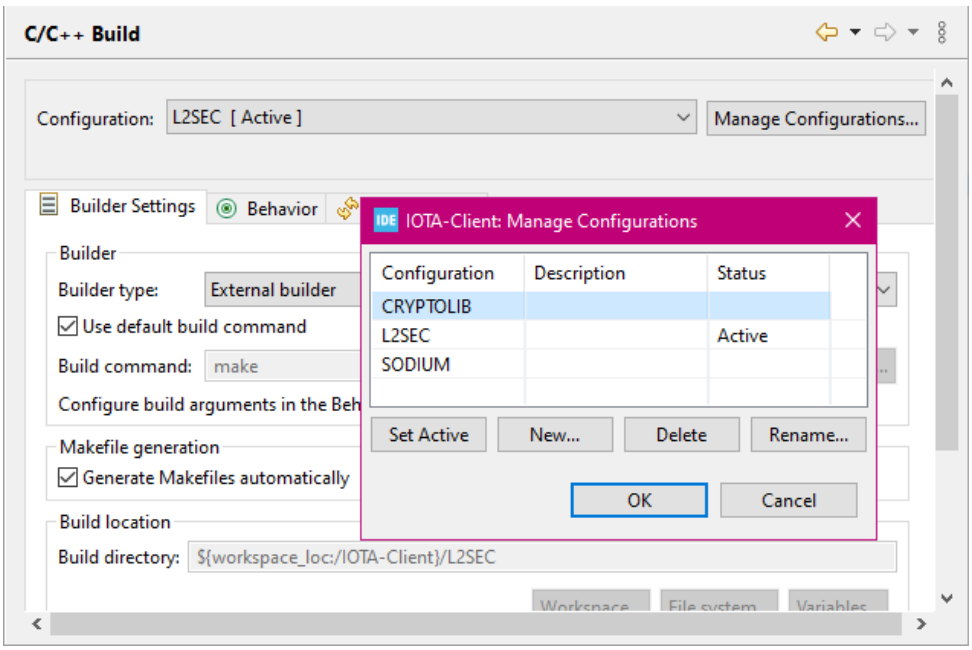


Figure 5.2: Build configurations

The "Build configurations" box in "Run configurations" will be set to "Select Automatically" by default; with this option, uploaded programs will not function properly. It is crucial to set it to L2SEC.

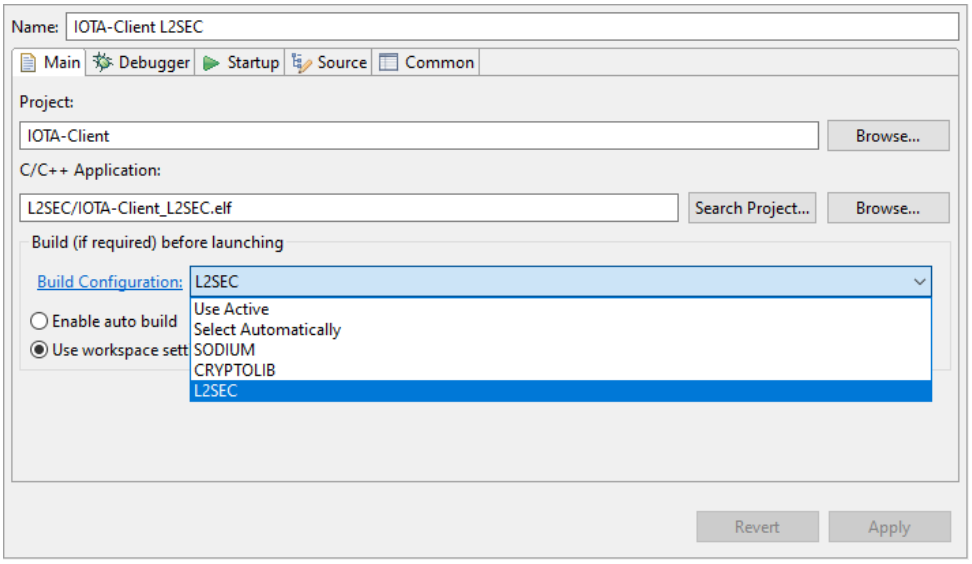


Figure 5.3: Run configurations

In debug configurations, ST-LINK S/N SWD interface is chosen.

5.5 Example analysis

Here I provide sample code from the X-CUBE-IOTA1 software package, to see how the described theory is applied in practice. In particular, the formation of the structure of the message and the inclusion of a payload in it. The function that we will consider is responsible for sending encrypted messages to the Mainnet network.

Function

```
1 int send_enc_data(void); // send encrypted data
```

utilize the next major steps and functions to send data to the Mainnet network:

1. Data encryption/decryption function;
2. Defining client endpoint configuration, such as next:
   host = NODE_HOST, port = NODE_HOST_PORT, use_tls = NODE_USE_TLS
   Which are defined for Mainnet:

```
1 #define NODE_HOST "chrysalis-nodes.iota.org"
2 #define NODE_HOST_PORT 443
3 #define NODE_USE_TLS true
4
```

3. Creation of the message object, which is correspond to structure of the message described in theoretical part:

```
typedef struct {
  uint64_t network_id;
    UT_array* parents;
    payload_t payload_type;
    void* payload;
    uint64_t nonce;
} core_message_t;

```

4. Creation of indexation payload:

```
indexation_create(Payload_type, data, data_length);

```

   Payload type for the Indexation payload is "2";

5. Sending the message itself (main part of this function):

```
send_core_message(client_endpoint_config, msg, &msg_res);

```

Let's investigate the inner workings of the *send_core_message()* function.

Three major events take place in this function:

1. Getting tips from the Mainnet network.
   To get tips an HTTP request is sent with the command "/api/v1/tips";
   Parents field is also fulfilled from the received tips structure.
2. Sending the HTTP request with the JSON data commands.
3. Getting as a respond "Message ID" saved in &msg_res structure , which can be used for getting message from the Tangle.

Chapter 6

Practical contribution

Remotely located devices may need to be managed. As a result, as part of the application level extension, I built methods that enable the transmission of messages containing instructions to the Tangle; these messages are downloaded by the designated client upon request. The command is consequently processed and executed. Figure 6.1 illustrates the description.

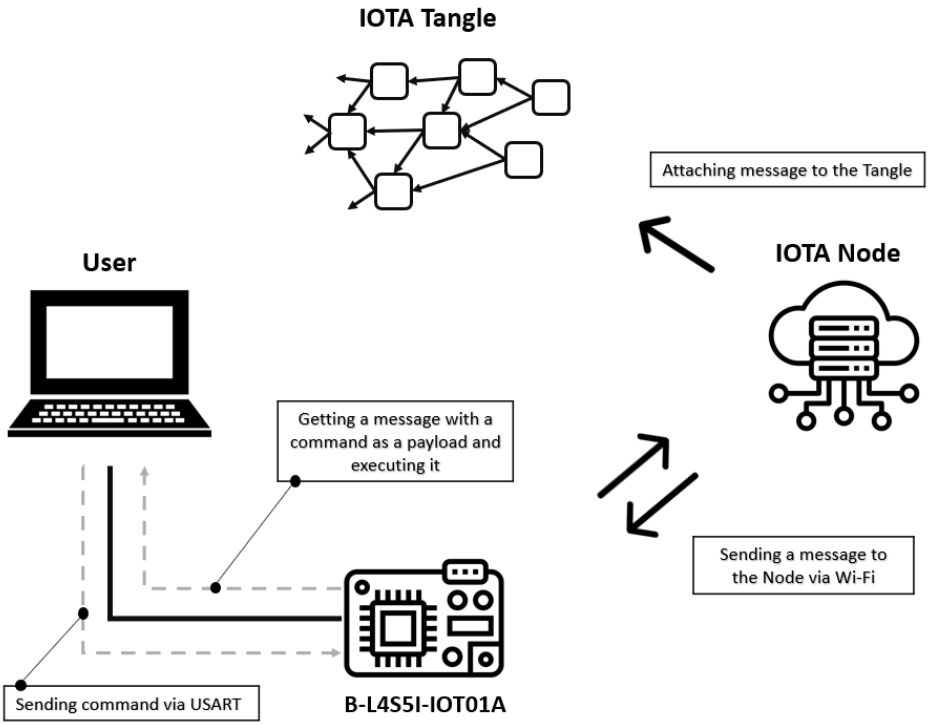


Figure 6.1: Proposed solution

After uploading project and connecting the device to WiFi menu will be displayed on the terminal. The menu will give the user a choice, whether to send message with an encrypted or decrypted (common) payload. In each of the cases entered command will then form part of the indexation payload and will be forwarded to the Mainnet network as a structured message, where it will be connected to the Tangle. In response to a client's request to attach

a message to a node, the node will complete a Proof of Work, verify the message, and then allocate and return a 64-digit hexadecimal number as the message identification (to B-L4S5I). This identification will then be used to receive a message from the network and execute the command it contains.

6.1 Implemented commands and functions

Defined commands:

```
{"use_l2sec", "use_enc_msg", "use_msg"}
```

use_l2sec: command to send L2Sec Stream.
The parameter is a number, from 1 to 255.

use_enc_msg: command to send an encrypted message. Parameter - a string from 1 to 127 characters long, not including end of the string character.

use_msg: command to send a standard message.
Parameter - a string from 1 to 127 characters long, not including end of the string character.

Each command also involves the transfer of a parameter, which is entered through the command's trailing space. The inclusion of a parameter is required for the command validity check to be completed successfully prior to its execution.

Further are given examples of the correct input:

use_l2sec 5
use_enc_msg Hello world
use_msg Hello world

Function:

```
/* Send message with user input request*/
void send_user_in_req_message(void);
```

includes next executable steps:

1. Request user to input the payload
2. Sending the message to the network (for more details view 5).

The actual function is shown below.

```
void send_user_in_req_message(void)
{
    char* id_input_request = "Payload:"; // question
    char* in_payload;
    size_t t_max_length = MSG_ID_LEN + 1;  // max_length

```

```
    in_payload = (char*) malloc(t_max_length);
    serial_get_ascii(id_input_request, in_payload, t_max_length);

    iota_client_conf_t ctx = {.host = NODE_HOST, .port =
    NODE_HOST_PORT, .use_tls = NODE_USE_TLS};

    printf("Sending data message to the Tangle...\n");
    res_send_message_t res;
    memset(&res, 0, sizeof(res_send_message_t));

    /*FEL\xF0\x9F\xA6\x8B - Index; \xF0\x9F\xA6\x8B - Butterfly
     symbol*/
    int ret = send_indexation_msg(&ctx, "FEL\xF0\x9F\xA6\x8B",
    in_payload, &res);
    if (ret == 0) {
    if (!res.is_error) {
      printf("message: https://explorer.iota.org/mainnet/message
    /%s\n", res.u.msg_id);

    } else {
      printf("Node response: %s\n", res.u.error->msg);
      res_err_free(res.u.error);
    }
    } else {
    printf("send_indexation_msg API failed\n");
    }
    return;
}
```

Function:

```
/* Send encrypted message with user input request*/
int send_user_in_req_msg_enc(void);
```

is similar to *send message* function 6.1 with the difference that the message entered by the user passes through the encryption function and only after that, in encrypted form, it becomes part of the indexation payload.

Encryption function placement (functions have been simplified to emphasize order):

```
    serial_get_ascii(id_input_request, in_payload_enc,
    t_max_length);
    /* After user input */
    /* Data encryption */
    xor_encrypt_decrypt(in_payload_enc, encrypted);
    /* Before indexation create */
    idx = indexation_create("FEL\xF0\x9F\xA6\x8B", (byte_t *)
    encrypted);
    send_core_message(idx);
```

Now that the available commands and rules for inputting them as well as the functions for sending a message have been discussed, let's move on to the primary function responsible for receiving a message from the network, examining the message, and executing the matching command if it was discovered.

Function:

```
int get_message_in_idx(void)
```

is responsible for:

1. User input of the message ID
2. Receiving the message
3. Unwrapping message
4. Check whether the payload is a command or a standard message
5. Parse the command
6. If the command was valid, the function intended to provide access will be executed.

The actual function is shown below.

```
int get_message_in_idx(void)
{
    char cmd_buffer[128] = {0};
    char cmd_decrypted[100] = {0};
    /* User input request message */
    char* id_input_request = "Message id:";
    size_t t_max_length = MSG_ID_LEN + 1;
    char* out_destination;
    out_destination = (char*) malloc(t_max_length);
    is_cmd_e is_cmd_ret = 0;
    /* Get user input and save it in out_destination pointer*/
    serial_get_ascii(id_input_request, out_destination,
    t_max_length);

    printf("%s -- %d", out_destination, strlen(out_destination));

    iota_client_conf_t ctx = {.host = NODE_HOST, .port =
    NODE_HOST_PORT, .use_tls = NODE_USE_TLS};

    res_message_t *msg = res_message_new();
    if (msg) {
        if (get_message_by_id(&ctx, out_destination, msg) == 0) {
            if (msg->is_error) {
                printf("API response: %s\n", msg->u.error->msg);
            } else {
                switch (msg->u.msg->type) {
                    case MSG_PAYLOAD_TRANSACTION:
                        printf("it's a transaction message\n");
                        dump_tx_payload(msg->u.msg->payload);
                        break;
                    case MSG_PAYLOAD_INDEXATION:
                    printf("it's an indexation message\n");

                    /*Unwrap the message and get the payload*/
                    dump_indexation_payload_cmd(msg->u.msg->
```

```
34                      payload, cmd_buffer);
35
36                       /* Check if command, in case of standard
37                       message */
38                      is_cmd_ret = is_cmd_payload(cmd_buffer);
39
40                       /* Check if command, in case of encrypted
41                       message */
42                      is_cmd_enc_ret = is_cmd_enc_payload
43                      (cmd_buffer, cmd_decrypted);
44
45                      /* If command was found in message then call
46                      an according function*/
47                      if(CMD_MSG == is_cmd_ret)
48                      {
49                      printf("Msg_cmd: %s\n", cmd_buffer);
50                      call_cmd_function(cmd_buffer);
51                      }
52
53                      /* If command was found in encrypted message
54                      then call an according function */
55                      else if(CMD_MSG == is_cmd_enc_ret)
56                      {
57                      printf("Msg_cmd_enc: %s\n", cmd_decrypted);
58                      call_cmd_function(cmd_decrypted);
59                      }
60                      else
61                      {
62                      printf("data: %s\n", "Not a command");
63                      dump_indexation_payload(msg->u.msg->payload);
64                      }
65                      break;
66                      case MSG_PAYLOAD_MILESTONE:
67                           printf("it's a milestone message\n");
68                           dump_milestone_payload(msg->u.msg->
69                           payload);
70                           break;
71                      case MSG_PAYLOAD_UNKNOW:
72                      default:
73                           printf("Unknow message\n");
74                           break;
75                  }
76              }
77          } else {
78          printf("get_message_by_id API failed\n");
79          }
80          res_message_free(msg);
81      } else {
82      printf("new message response failed\n");
83      }
84
85      return 0;
86  }
```

Parsing and calling the function is in:

```
1 call_cmd_function(cmd_buffer); //line 50 and 59
```

This function is responsible for deciding which command function should be executed.

```
static void call_cmd_function(char* in_cmd_buffer)
{
  valid_out_e res = WRONG_INPUT;
  cmd_pack_t cmd_f = {0, 0};
  size_t t_str_lim = 128;  // max_length
  cmd_f.supply_str = (char*) malloc(t_str_lim);

  /* Parsing function  */
  /* cmd_f is the structure with data about command type */
  /*and user parameter included to the command via terminal*/
  res = parse_cmd(in_cmd_buffer, &cmd_f);

  if(CORRECT_INPUT == res)
  {
    switch(cmd_f.cmd_func)
    {
      case L2SEC_FINIT:
        /*L2Sec finit stream*/
        send_l2sec_protected_stream(cmd_f.supply_val);
        serial_press_any();
        break;
      case ENCRYPTED_MSG:
        /*Sending encrypted message*/
        send_enc_data_user_in(cmd_f.supply_str);
        serial_press_any();
        break;
      case MSG:
        /*Sending message*/
        send_data_message_user_in(cmd_f.supply_str);
        serial_press_any();
        break;
      case NONE:
        break;
      default:
        break;
    }
  }
  else{
    printf("Wrong input.");
  }
  return;
}
```

Additional data types:

1. Enumeration type *cmd_func_e* used as the return type of the parse function to indicate user commands.

```
typedef enum
{
  NONE = 0,
  L2SEC_FINIT = 1,
  ENCRYPTED_MSG,
  MSG
}cmd_func_e;
```

2. Structure type *cmd_pack_t* is used to transfer the data needed to execute the commands to the place of their use.

```
typedef struct
{
  cmd_func_e cmd_func;
  uint16_t supply_val;
  char* supply_str;
}cmd_pack_t;
```

3. Enumeration type *cmd_func_e* used as a return type to determine the validity of function output.

```
typedef enum
{
  WRONG_INPUT = 0,
  CORRECT_INPUT
}valid_out_e;
```

And finally, the parsing function is presented below.
All explanations are provided in the comment sections in the code.

```
static valid_out_e parse_cmd(char* in_cmd_str, cmd_pack_t*
    out_cmd_f_type)
{
valid_out_e ret = WRONG_INPUT;
valid_out_e check_digit_ret = WRONG_INPUT;
uint16_t out_int_digit;
uint8_t temp_ret_1 = 0;

size_t str_len = 0;
int cmp_ret = 0;
char* cmd;
char* str_supply;

const char ch = ' ';

temp_ret_1 = strlen(in_cmd_str);

if(0 != temp_ret_1) /*If string is not empty*/
{
  /*Check if space is included into the string*/
  if(NULL != strchr(in_cmd_str, ch))
  {
    /* Extracting command  from the common string
     * containing cmd and parameter */
    cmd = strtok(in_cmd_str, " ");
    for(uint32_t i = 0; i < CMD_NUM; i++)
      {
        /* Comparing input with existing commands */
        cmp_ret = strcmp(cmd, cmd_set[i]);
        /* If input is the same as existing cmd then True */
        if(0 == cmp_ret)
        {
          printf("in parse");

```

```
34            /* Extracting parameter  from the common string
35             * containing cmd and parameter */
36            str_supply = strtok(NULL, " ");
37            switch(i)
38            {
39              case 0:
40                /* For L2Sec func check if parameter is string of
41                 * digits and it is in range from 1 to 25.
42                 * Func convert str of char to int number and
43                 * return it through parameter if input was valid.
    */
44                check_digit_ret = check_l2sec_digit(str_supply,
45                &out_int_digit);
46
47                if((CORRECT_INPUT == check_digit_ret))
48                {
49                  out_cmd_f_type->cmd_func = L2SEC_FINIT;
50                  out_cmd_f_type->supply_val = out_int_digit;
51
52                  ret = CORRECT_INPUT;
53                }
54                break;
55              case 1:
56                /*String lenght is between 1 and 127*/
57                str_len = strlen(str_supply);
58                if(MIN_STR_LEN<= str_len && MAX_STR_LEN >=str_len)
59                {
60                  out_cmd_f_type->cmd_func = ENCRYPTED_MSG;
61                  out_cmd_f_type->supply_str = str_supply;
62
63                  ret = CORRECT_INPUT;
64                }
65                break;
66              case 2:
67                /*String lenght is between 1 and 127*/
68                str_len = strlen(str_supply);
69                if(MIN_STR_LEN<= str_len && MAX_STR_LEN >=str_len)
70                {
71                  out_cmd_f_type->cmd_func = MSG;
72                  out_cmd_f_type->supply_str = str_supply;
73
74                  ret = CORRECT_INPUT;
75                }
76                break;
77              default:
78                break;
79            }
80
81            break;
82          }
83        }
84    }
85  }
86  return ret;
87  }
```

Using command *use_l2sec X* as an example, let's examine how it works.

Of the menu options, I choose *Send data message*, entered the command *use_l2sec X*.

And got next output to the terminal:

message: https://explorer.iota.org/mainnet/message/543cf0258254a242b887504ff722f410e4daa0aec00f1894ab8c0c6f762a1a69

Message can be found on the following link [1], by entering 543cf0258254a242b887504ff722f410e4daa0aec00f1894ab8c0c6f762a1a69 to the search.

Then, upon receiving the message by Message ID, the associated function was performed (sending L2Sec messages), resulting in the following output(L2Sec protected Stream execution):

```
Channel: >> 5 << message(s) will be sent

Channel: sending message # 1 (hex: 1) ...

Application data message to L2Sec protect and send:
'SENSOR DATA (IOTA L2Sec app payload #1) — Thu May 19 08:12:41
    2022
{"Device":"B-L4S5I-IOT01A","time":1652947961,"temp":32.78,"humi
    ":27.46}'

STSAFE: Random - Generate -> OK
STSAFE: Random - Generate -> OK
STSAFE: Authentication Signature - Sign -> OK
STSAFE: PSK - Unwrap -> OK
STSAFE: Random - Generate -> OK
Cleaning PSK from RAM... -> OK

Sent message - ID:
7e376cadf4dc47e68b4523ef6c38ac493f2b118aa2069faae98857a3a993c438
Sent message - index:
551E547DD3C9F7837E94105FC266BEA72514D6A76E2525A30939B2D31585DEAC

Index of the first message in the channel: 551
    E547DD3C9F7837E94105FC266BEA72514D6A76E2525A30939B2D31585DEAC


Channel: sending message # 2 (hex: 2) ...\vs
```

Messages can be found on [1], by entering Index of the first message in the channel: 551E547DD3C9F7837E94105FC266BEA72514D6A76E2525A30939B2D31585DEAC

6.2 Additional system evolution

In addition to the already proposed application layer extensions, I prepared the basis for porting the IOTA project to another platform and implemented BLE (Bluetooth low energy) communication between the two devices.

The idea is that in the future it would be possible to use more than one STM controller to interact with Tangle and by means of BLE control access of another device to send messages to the IOTA network.

The second device is the STM32 Discovery kit B-L475EIOT01A2C, which has a different CPU than the prior kit, making the process of adding IOTA software to this device more difficult.

Next, I give a quick, efficient and proven way to organize BLE communication between devices B-4S5I and B-L475 (STM32CubeIDE is used).

This method is suitable for any STM32 devices with a built-in SPBTLE-RF BLE module or those that use a separate one.

Communication will work so that by pressing a button on one device, it will toggle on another. The application works bidirectionally.

Steps to be taken to configure BLE for B-L475:

1. In STM32CUbeIDE go to Help > Manage Embedded Software Packages; Click the STMicroelectronics tab; find the X-CUBE-BLE1 package, choose the last version, and install it;
2. Then File > New > STM32 Project; select your board in the Board selector; create the project and open CubeMX (.ioc file);
3. Go to Software Packs > Select Components and choose configurations as shown in Figure 6.2.
4. In Pinout & Configuration tab choose the BLE1 package in the Software Packes tab;
5. Choose both Wireless BlueNRG-MS and Device BLE1 Applications;
6. Communication with the BLE module is going on via SPI, therefore figure out what SPI is connected on your board to this module in User Manual. As well as find out what USART is connected to the Virtual-COM port. USART is used in the application to provide user feedback about the connection and communication process.
7. Enable and configure SPI and USART in Pinout & Connectivity > Connectivity.
8. HCI_TL_iNTERFACE (HCI stands for Host-Controller Interface) and BSP configurations are provided in Figures 6.3 and 6.4 accordingly.

9. Then click on the Device Configuration Tool Code Generation button on the Instrument panel.

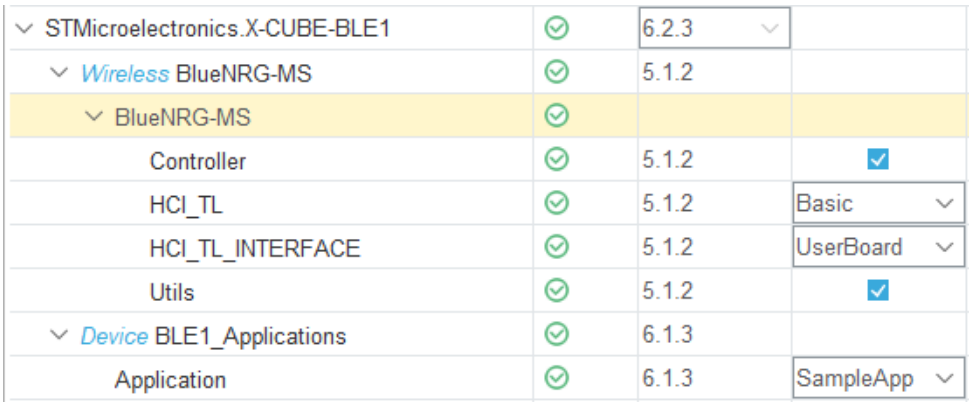


Figure 6.2: BLE1 configurations

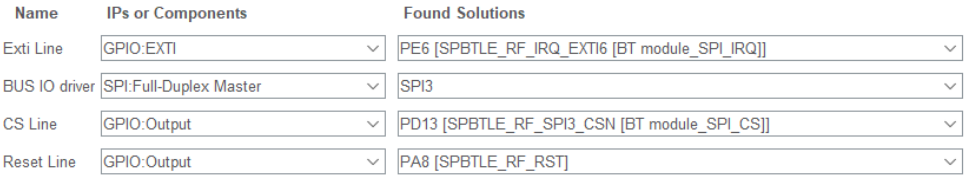


Figure 6.3: HCI_TL_INTERFACE configurations

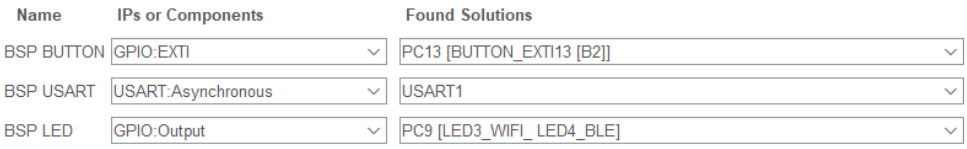


Figure 6.4: BCD configurations

All the processes must be repeated for the second board. For B-L4S5I, the only variation between these stages will be the selection of a different board during project creation.

Whether the device is a server or a client is determined by defining SERVER_ROLE or not accordingly.

Therefore use at the BlueNRG_MS/App/app_bluenrg_ms.c:

```
1 #define SERVER_ROLE
```

or comment it.

Now load the project first to the Client board and restart it. And do the same for the Server board. When LED 4 goes out, then the devices are connected.

Chapter 7

Practical part summary and discussion

The objective of the practical section was to serve as the foundation for remote device control through a decentralized network. In such a manner that one device transmits messages containing payloads in the form of commands to the IOTA network, another device may download and execute these messages. In my thesis, I examined this concept on a single device. However, the functionality I developed may be readily incorporated into other devices. Therefore, this system might be enhanced by including Bluetooth Low Energy communication across devices, with each item able to connect to Tangle. For instance, while sending a command to Tangle for execution, the master device sends a Message-ID (through Bluetooth) as well to the device that should download and execute the message.

Chapter 8

Conclusion

The objective of this study was to investigate Internet of Things-integrable decentralized technologies like Blockchain, Smart Contracts, and others. It was important to assess whether or not a certain technology is suited for such integration and to provide the optimal solution. The foundation for developing a microcontroller-based system.

Taking a slow, bottom-up approach, I began my work with a short introduction to IoT, concentrating mostly on the problems with the present IoT architecture, to clarify why many are attempting to employ a decentralized method in the design of IoT systems today. In addition, two technologies, Blockchain and IOTA Tangle serve as the foundation for study and comparison. Starting with Blockchain because, in my opinion, it is easier to comprehend the operation of IOTA Tangle (a more sophisticated technology) if you are already familiar with the operation of another decentralized technology. Moreover, I conclude by explaining why IOTA Tangle is the ideal answer for Constrained devices.

During my study for the thesis, I discovered that STMicroelectronics has just produced software that implements the Chrysalis protocol from IOTA. This ST software package was created just for the B-L4S5I board I worked with. I discovered how this software package is structured, allowing the developer to easily and flexibly increase the user level's capabilities by supplementing it and creating add-ons for their own reasons.

Therefore, I built extra functionality in the C programming language that enables remote device control through Tangle. In such a manner that one device must transmit commands to the Tangle, another device may download and execute these commands. Currently, this capability has only been tested on one device, but flexible interaction with additional devices is assumed.

But the first step toward working with numerous devices has already been done, as I present a technique for extending the system through Bluetooth in the Additional system evolution section.

Bibliography

[1] LIJUN, Wei, Liu SHAOWEI, Wu JING a Long CHENGNIAN. Enabling Distributed and Trusted IoT Systems with Blockchain Technology. IEEE Blockchain [online]. Hong Kong, 2018, 2019 [cit. 2022-04-27]. Available from: https://blockchain.ieee.org/technicalbriefs/january-2019/enabling-distributed-and-trusted-iot-systems-with-blockchain-technology

[2] GUINARD, Dominique. THE LEDGER OF EVERY THING: What Blockchain Technology Can (and Cannot) Do for the IoT [online]. 2017, 38 [cit. 2022-05-03]. Available from: https://evrythng.com/wp-content/uploads/2018/11/The-Ledger-of-Every-Thing_-What-Blockchain-Technology-Can-and-Cannot-Do-for-the-IoT-1.pdf

[3] LEA, Perry. IoT and Edge Computing for Architects: mplementing edge and IoT systems from sensors to clouds with communication systems, analytics, and security. 2nd ed. Birmingham: Packt Publishing, 2020. ISBN 978-1839214806.

[4] SHENG-LUNG, Peng, Pal SOUVIK a Huang LIANFEN. Principles of Internet of Things (IoT) Ecosystem: Insight Paradigm. Switzerland: Springer Cham, 2020. ISBN 978-3-030-33595-3.

[5] TAMBOLI, Anand. Build Your Own IoT Platform: Develop a Fully Flexible and Scalable Internet of Things Platform in 24 Hours. Sydney: Apress, 2022. ISBN 978-1484244975.

[6] VYAS, Sonali, KUMAR SHUKLA, Vinod, Shaurya GUPTA a Ajay PRASAD, ed. Blockchain Technology: Exploring Opportunities, Challenges, and Applications [online]. Boca Raton: CRC Press, 2022 [cit. 2022-04-02]. ISBN 9780367685584. Available from: doi:https://doi.org/10.1201/9781003138082

[7] NAKAMOTO, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System [online]. 2008 [cit. 2022-03-20]. Available from: https://bitcoin.org/bitcoin.pdf

[8] POPOV, Serguei. The Tangle [online]. 2018 [cit. 2022-03-25]. Available from:

https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1/_4_3.pdf

[9] Kevin Ashton Invents the Term "The Internet of Things": Exploring the History of Information and Media through Timelines. HistoryofInformation [online]. Cambridge, 1999 [cit. 2022-03-03]. Available from: https://www.historyofinformation.com/detail.php?id=3411

[10] Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030, by vertical. Statista [online]. Cambridge: Statista, 2022 [cit. 2022-03-05]. Available from: https://www.statista.com/statistics/1194682/iot-connected-devices-vertically/

[11] Digital Signatures and Certificates. In: GeeksforGeeks [online]. 2021 [cit. 2022-03-15]. Available from: https://www.geeksforgeeks.org/digital-signatures-certificates/#: :text=A

[12] About IOTA: Data Transfer. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2022-04-06]. Available from: https://wiki.iota.org/learn/about-iota/data-transfer

[13] Vývojová sada pro procesory a mikrokontroléry, Cortex M4, 32 bitů, Cortex M4, B-L4S5I-IOT01A, Deska mikrokontroléru. In: Rs-online [online]. [cit. 2022-05-06]. Available from: https://cz.rs-online.com/web/p/vyvojove-nastroje-pro-mikrokontrolery/2044106?cm_mmc=CZ-PLA-DS3A-_-google-_-PLA_CZ_CZ_Raspberry_Pi_

[14] CARELLI, Alberto, Andrea PALMIERI, Antonio VILEI, Fabien CASTANIER a Andrea VESCO. Enabling Secure Data Exchange through the IOTA Tangle for IoT Constrained Devices. Sensors [online]. 2022 [cit. 2022-04-19]. Dostupné z: doi:https://doi.org/10.3390/s22041384

[15] STMicroelectronics. B-L4S5I-IOT01A: STM32L4+ Discovery Kit IoT Node, Low-Power Wireless, BLE, NFC, WiFi [online]. STMicroelectronics, ©2021 [cit. 2022-04-01]. Available from: https://www.st.com/en/evaluation-tools/b-l4s5i-iot01a.html

[16] IOTA Foundation. IOTA Wiki. The Complete Reference for IOTA [online]. IOTA, ©2021 [cit. 2022-03-15]. Available from: https://wiki.iota.org

[17] STMicrolectronics. X-CUBE-IOTA1 [online]. ©2021 [cit. 2022-04-01]. Available from: https://github.com/STMicroelectronics/x-cube-iota1

[18] STMicroelectronics. UM2606. Rec 4. ©2021. Available online: https://www.st.com/resource/en/user_manual/um2606-getting-started-with-the-iota-distributed-ledger-technology-software-expansion-for-stm32cube-stmicroelectronics.pdf