

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARMENT

PRESENTATION DESIGN
INDIVIDUAL DIPLOMA THESIS

May 2019

Individual Diploma Thesis

**IMPLEMENTATION OF BLOCKCHAIN-LIKE
TECHNOLOGY IN IOT DEVICES**

Charalampos Theodorou

UNIVERSITY OF CYPRUS



COMPUTER SCIENCE DEPARTMENT

May 2019

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARTMENT

Implementation of Blockchain-like technology in IoT devices

Charalampos Theodorou

Supervisor Professor
Assistant Professor Dr. Vasos Vassiliou

The Individual Diploma Thesis was submitted in partial completion of the requirements needed for acquiring an undergraduate diploma in Computer Science of the Computer Science Department of the University of Cyprus

May 2019

Gratitude

This project was done under the supervision of the Assistant Professor Dr Vasos Vassiliou of the Computer Science Department of the University of Cyprus. I owe many gratitudes to this person for providing me with many materials to study and research that proved to be fundamental for my further process on this project. During our weekly meetings with my supervisor we will discuss future plans and decide on options for the further development of my project. Without his help and dedication to this project nothing could have been achieved.

Special Thanks are owed to Dr Christiana Ioannou, researcher at the NET Research Laboratory of the Computer Science Department of the University of Cyprus. With her help and provided examples given by her I was able to understand and further my knowledge of the Contiki Operating System. I want to express my gratitude for always being available and eager to assist me in developing this project.

Furthermore, I want to say thanks to all my friends and family that stood by me throughout these four long years and especially through this last month where the hours working on this project risen exponentially and the work load was extremely high. Their moral support was exceptional and without them I wouldn't have the same results and strength to continue.

Summary

In my research I came across with many various new technologies such as the IEEE Xplore, a blockchain implementation that the Ethereum crypto was based on, the IBM's Adept, IBM's hyper-ledger, Samsung Watson, but none of them stand out like the IOTA Protocol.

During the first semester I studied this new technology, the IOTA protocol, and decided that what this technology offers is an ideal candidate for our project. Though this technology is still in experimental and developing stage and keeps changing and adapting, many people are contributing to this, developers from the IOTA foundation as well as independent developers that "smelled out" the opportunities of this technology.

The overall idea of this project was to test the abilities of a selected Technology, later chosen to be the IOTA, creating a safe environment where IoT devices can exchange data and communicate without any dangers of attacks or tampering of data.

Using this technology was quite a challenge as many new challenges arise along the way, mainly because this is a fresh technology and still evolving with no many experts out there to provide guidelines to new developers that want to experiment with it and to help cope with bugs that come along the way.

After the research part was done, I had to do some testing to this Technology. The main purpose of this whole project was to use a network of micro-sensor IoT devices to communicate with each other and to store somewhere secure their collected data and then being able to retrieve them. The programming of these devices was done in a Contiki OS environment and a Raspberry Pi was used to serve as the gateway of this network and at the same time as the connector to the IOTA network, responsible to attach the data collected from these IoT devices.

Overall the results of experimenting with this technology was quite sufficiently good, data was gathered from the sensors and published to all IoT connected devices until the information reached the gateway and then using a feature of the IOTA protocol, MAM, we were able to see the data being send and attached to the IOTA network and later on to receive them.

Contents

Chapter 1	Introduction.....	9
1.1	Introduction.....	9
1.2	Problem Definition.....	9
1.2.1	IoT Devices.....	9
1.2.2	Problems.....	10
1.3	Purpose of this Project.....	10
1.4	Methodology and Results.....	11
1.5	Chapters Overview.....	12
1.6	Table of Contents Definition.....	13
Chapter 2	The IOTA Tangle	14
2.1	Introduction.....	15
2.2	Trytes and Trits.....	15
2.3	Structure of the Tangle.....	17
2.4	Transactions.....	18
2.4.1	What is a Transaction.....	18
2.4.2	Transaction Object.....	18
2.4.3	Types of Transactions.....	20
2.4.3.1	Input Transaction.....	20
2.4.3.2	Output Transaction.....	20
2.4.3.3	Meta Transaction.....	20
2.4.4	Tip Selection.....	21
2.4.5	Own and Cumulative Weights.....	21
2.4.6	Proof-of-Work.....	22
2.4.7	Minimum Weight Magnitude.....	23
2.4.8	Bundle Construction.....	23
2.4.9	Attaching the Transaction.....	24
2.5	Seeds.....	24
2.5.1	What is a Seed.....	24
2.5.2	How a Seed is Created.....	25
2.5.3	How a Seed is Used.....	26
2.6	Addresses.....	26
2.6.1	What is an address.....	26
2.6.2	Attaching an address to the Tangle.....	27

2.6.3 Dangers when using the same address.....	27
2.7 Coordinator.....	27
2.7.1 Usage of a Coordinator.....	28
2.7.2 Milestones.....	28
2.8 Comparison with Blockchain.....	29
2.8.1 Explaining the Blockchain.....	29
2.8.2 Comparing the two technologies.....	29
2.8.3 Choosing the best candidate.....	30
Chapter 3 Private Tangle.....	32
3.1 Introduction.....	32
3.2 Components used.....	32
3.3 Difference Public-Private.....	33
3.4 Results.....	33
Chapter 4 Gateway and MAM	34
4.1 Introduction.....	34
4.2 Components used.....	34
4.3 Masked Authenticated Messaging.....	35
4.3.1 What is MAM.....	35
4.3.2 Mam Object.....	35
4.3.3 Message.....	37
4.3.4 Message Chain/Stream.....	38
4.3.5 Merkle Tree.....	38
4.4 Raspberry Pi Usage.....	39
4.5 Attaching Test.....	39
4.5.1 Establishing connection to the Public Tangle.....	40
4.5.3 MAM Example.....	41
4.6 Results.....	42
Chapter 5 Contiki wireless Sensor Network.....	43
5.1 Introduction.....	43
5.2 Components.....	44
5.3 Contiki OS.....	44
5.3.1 What is the Contiki OS.....	44
5.3.2 Features.....	44

5.3.2.1 Memory Allocation.....	45
5.3.2.2 Full IP Networking Stack.....	45
5.3.2.3 Power Awareness.....	45
5.3.2.4 Examples.....	45
5.3.2.5 Cooja Network Simulator.....	45
5.4 Tmote Sky.....	46
5.4.1 What is the Tmote Sky.....	46
5.4.2 Features.....	47
5.4.3 Use Cases.....	47
5.4.3.1 Smart Housing.....	47
5.4.3.2 Smart Cities.....	47
5.4.3.3 Emergency Situations.....	47
5.4.3.4 IOTA example.....	47
5.5 Broadcasting data.....	48
5.6 Collecting view.....	49
5.7. Attaching MAM to the Tangle.....	50
5.8 Retrieving data from the Tangle.....	51
Chapter 6 Conclusion.....	52
6.1 Introduction.....	52
6.2 Using the IOTA protocol.....	52
6.3 Modifications of Original Idea.....	52
6.4 Overall Conclusions.....	53
References.....	54
Appendix A.....	56
Appendix B.....	59
Appendix C.....	64

Chapter 1

Introduction

1.1 Introduction.....	9
1.2 Problem Definition.....	9
1.2.1 IoT Devices.....	9
1.2.2 Problems.....	10
1.3 Purpose of this Project.....	10
1.4 Methodology and Results.....	11
1.5 Chapters Overview.....	12
1.6 Table of Contents Definition.....	13

1.1 Introduction

In this chapter is described the definition of the problem that was investigated and the detailed purpose that this project was created for. It also contains a reference list of each chapter to follow, as well as a short explanation for each chapter accordingly. At the end of this chapter a table of contents definition will be provided, that contains all the terminology used in the chapters to follow.

1.2 Problem Definition

In this subchapter I will discuss the problems that I was assigned to provide a solution with this project. In order to do that I have to first explain the meaning of the IoT devices and how they communicate.

1.2.1 IoT Devices

IoT stands for Internet of Things and it is considered by many to be all the physical devices that are connected to the Internet. The term IoT has change meaning through the past years, as the development of these devices is evolving and creating many more and specific hardware for these machines. IoT devices can be considered to be anything that has a connection to the Internet and is embedded with electronics and most of the times specific sensors.

1.2.2 Problems

Communication between IoT devices can be tricky, especially if confidential data are in the exchange. It is needed to find a secure solution when exchanging data and communicating between IoT devices. When dealing with IoT devices, the processing capabilities are most of the times very small and with effect no insurance is given when communicating over the Internet, no protection is guaranteed at the end points. Another issue when exchanging information with these devices, no insurance is given when information is passed through various end points in the Internet structure, as many routes can be vulnerable and prompt for an imminent attack. With no ability to verify the data that we receive on the Internet today, and with cybercrime on the rise, this delegated and unverifiable trust has become a major obstacle for an inclusive and permission less communication. The dangers arise when using these devices, small processing capability, need to rely to 3 party devices to carry out the transfer to the other peers. When communicating with IoT devices certain problems are imminent, like dangers of attacks, malware infection, tampering with data and eventually creating invalid data that causes authenticity issues, the accuracy of certain information and eventually the trust of devices is lost and overall creating a vulnerable and unsecure environment.

1.3 Purpose of this Project

The purpose of this project is to find a viable solution in the problem of communication of IoT devices over the Internet. The need for secure communications and exchange of data across the internet rises more and more alongside with the dangers threatening this vast communication exchange. In order to create an environment where IoT devices can establish secure communications with integrity and the ability for scalability we must research all the available services provided.

As part of this project I was task to research various implementations of distributed ledger technologies and their usage for this particular scenario, the most known being the Blockchain ledger. In depth analysis must take place of each of the different technologies available in order to find the most suitable option.

Lastly, after deciding the technology best suited to our needs, a testing of this technology must take place to ensure success of our idea for secure IoT communication.

1.4 Methodology and Results

In order to proceed with the testing of secure communication of the IoT devices we must first decide the best suited available technology of distributed ledger. The first step to this was the research of various technologies, the most commonly known being the blockchain ledger. For this to happen I research the details of many technologies, of how they operate and what are their specific purposes with finally choosing the IOTA protocol for its new, ground breaking way to exchange data through Machine-to-Machine (M2M) communication ^[17]. This was done after thorough testing and reading the manuals of this software mainly located on Github repositories. This required a lot of effort in understanding the fundamentals of this new technology and its features as it is still in development and still rapidly changing as many developers and contributors still make an impact to improve this new technology.

After the research was done and the technology to operate on was chosen, then I tried to run some experiments on creating our own private network on a remote virtual server where IoT devices will post their information. This was proven to be insufficient in the end result as many issues arise during the process and was decided to operate on the public version of this. Then I chose a middle-out approach for the rest of the project in the sense that we first created the environment for our IoT devices to take sensor data and then publish them in the network with the goal to reach the gateway, our Raspberry Pi.

The next step was to program the IoT devices in the Contiki OS environment in order to have some devices that will collect information and publish them to the rest and one device that will collect all the devices and transmit them to the Raspberry Pi. It was fairly easy to program the IoT devices to broadcast sensor data throughout our local sensor network and we successfully programmed a broadcast Tmote sky and a sink Tmote sky which served as the immediate connection with the gateway (Raspberry Pi).

After this was done I created a program to test the connectivity of the Raspberry Pi to the Public network and try to attach information and later on receive them. This test used simple JSON format files and source code provided from the official repository of the IOTA Foundation on Github. This was successful and provided the confirmation that the goal to attach the sensor information was possible and that a secure environment can be created for these IoT devices to communicate.

The last stage of the project was to connect the Raspberry Pi to the network of sensors and act as the gateway of this network. This was done in order to receive the sensor data from the devices in the

network and attach real data to the Public network of IOTA and later retrieve them.

1.5 Chapter Overview

In chapter two we explain what the IOTA foundation is, the technology developed by the IOTA, the IOTA protocol and all the aspects in detail about this new evolving new technology. The tangle, the meaning of transactions, the creation of new transactions (the process behind this), we explain the semantics and the components of this technology in detail. And we do a comparison of the IOTA protocol between the Blockchain technology and we why choose this technology instead of the Blockchain. In chapter three we introduce the meaning of the Private Tangle, why it was an idea to use such implementation of the IOTA protocol and why it proven to not be beneficial for our case study. In chapter four we study in detail about the MAM feature of the IOTA protocol and the implication it has for the future of the IOTA and IoT devices communication. In chapter five we talk about the Contiki OS environment, the sensors we used in this project and the code implemented, Tmote Sky, the sink node and the connection with the raspberry pi and we see an experiment of broadcasting data from the sensor devices and later on attaching this data to the IOTA network. Lastly, in chapter six I list all my experience with this project and the IOTA protocol, all the pros and cons of using this new technology and what I achieved these two semesters by all this.

1.6 Table of contents definitions

Name of Component	Short Explanation
DAG	Directed Acyclic Graph.
IOTAs	Units of Measurement in the tangle, referred to as “tokens”, used to give value to transactions.
Bundle of Transactions	Group of transactions .
Tips	Unapproved transactions in the tangle graph.
Height	Length of the longest oriented path to the genesis.
Depth	Length of the longest reverse-oriented path to some tip.
Consensus	General agreement between the transactions of the tangle.
PoW	Proof-of-Work for the Transactions
Private Key	Key that is used to create transactions, also referred to as an address.
Ternary	Any given three elements that when combined form a single element.
Sybil Attack	An attack where in a reputation system is subverted by forging identities in peer-to-peer networks. It is named after the subject of the book Sybil , a case study of a woman diagnosed with dissociative identity disorder.
RWMC	Reverse Walk Monte Carlo Algorithm
6LoWPAN	Internet Protocol for Low-power Wireless Personal Area Network.

Chapter 2

The IOTA protocol

2.1 Introduction.....	15
2.2 Trytes and Trits.....	15
2.3 Structure of the Tangle.....	17
2.4 Transactions.....	18
2.4.1 What is a Transaction.....	18
2.4.2 Transaction Object.....	18
2.4.3 Types of Transactions.....	20
2.4.3.1 Input Transaction.....	20
2.4.3.2 Output Transaction.....	20
2.4.3.3 Meta Transaction.....	20
2.4.4 Tip Selection.....	21
2.4.5 Own and Cumulative Weights.....	21
2.4.6 Proof-of-Work.....	22
2.4.7 Minimum Weight Magnitude.....	23
2.4.8 Bundle Construction.....	23
2.4.9 Attaching the Transaction.....	24
2.5 Seeds.....	24
2.5.1 What is a Seed.....	24
2.5.2 How a Seed is Created.....	25
2.5.3 How a Seed is Used.....	26
2.6 Addresses.....	26
2.6.1 What is an address.....	26
2.6.2 Attaching an address to the Tangle.....	27
2.6.3 Dangers when using the same address.....	27
2.7 Coordinator.....	27
2.7.1 Usage of a Coordinator.....	28
2.7.2 Milestones.....	28
2.8 Comparison with Blockchain.....	29
2.8.1 Explaining the Blockchain.....	29
2.8.2 Comparing the two technologies.....	29
2.8.3 Choosing the best candidate.....	30

2.1 Introduction

In this chapter we will discuss in detail this new technology provided by the IOTA Foundation, the IOTA protocol and all its features in order to try to explain why all these can be used in my project in creating a safe and secure environment for communication between IoT devices. In the last section of this chapter I compare this new technology with the Blockchain technology listing their differences and similarities and why I chose this implementation for my project.

The IOTA protocol is a distributed ledger Technology, which in short is a database that is shared among connected devices being synchronized and share digital information in a commonly established platform. With the advent of distributed ledger technologies, the option to distribute and synchronize ledgers of data and money in a secure ^[17], decentralized and permission-less environments are now possible. By removing the need for trusted third-parties as the gatekeepers and arbiters of truth, enormous efficiency gains, innovation opportunities and new value propositions emerge with the use of this new, cutting-edge technology.

IOTA is specific developed for IoT or Machine-To-Machine (M2M) communication. IoT devices will generate a huge amount of data which can be stored on the Tangle. These IoT devices can broadcast data using different types of wireless technology, such as WiFi, BlueTooth, Lora, ZigBee, etc.

2.2 Trytes and Trits

The IOTA foundation created and adopted the Tryte Alphabet System for the IOTA distributed ledger. The Tryte Alphabet is a balanced ternary system which is shown in figure 2.1 in form of a table of each Tryte character and its corresponding Trits and Decimal values. Messages exchanged in the IOTA ledger use these Trits, which are in the form of $\{-1,0,1\}$ and when combined in 3s forma Tryte which can represent 27 states. These Trytes were created in order to make the Trits more human readable. The Tryte alphabet consists of the 26 letters of the latin alphabet plus the number 9, totalling at 27 characters in total. IOTA uses this alphabet when creating seeds, address, hashes and keys.

Tryte-encoded character	Trits	Decimal Number
9	0, 0, 0	0
A	1, 0, 0	1
B	-1, 1, 0	2
C	0, 1, 0	3
D	1, 1, 0	4
E	-1, -1, 1	5
F	0, -1, 1	6
G	1, -1, 1	7
H	-1, 0, 1	8
I	0, 0, 1	9
J	1, 0, 1	10
K	-1, 1, 1	11
L	0, 1, 1	12
M	1, 1, 1	13
N	-1, -1, -1	-13
O	0, -1, -1	-12
P	1, -1, -1	-11
Q	-1, 0, -1	-10
R	0, 0, -1	-9
S	1, 0, -1	-8
T	-1, 1, -1	-7
U	0, 1, -1	-6
V	1, 1, -1	-5
W	-1, -1, 0	-4
X	0, -1, 0	-3
Y	1, -1, 0	-2
Z	-1, 0, 0	-1

Figure 2.1 table showing the equivalence of Trytes, Trits and Decimal values..

2.3 Structure of the Tangle

A tangle is a data structure based on a DAG which stands for Directed Acyclic Graph. Directed means the graph is pointing to one direction only, starting from the root transaction, the “Genesis” of the tangle, and moving through time to the next ones. Acyclic means the graph, which is slowly created with the insertion of new transactions every second, is non circular and every new transaction points to previously non pointed transactions. The vertices in the DAG represent transactions and the edges represent approvals to these transactions. When a new transaction is received, it is added as a new vertex in the DAG. Approving a transaction implies that its history was verified and found to be valid. This makes sure there are no double-spends or new illegitimate transactions created.

The “Genesis” as mentioned before, is the first transaction in the Tangle. All the IOTA tokens are created in this genesis transaction and no new ones will ever be created, these tokens are constantly distributed through transactions. All subsequent transaction in the tangle approve the “Genesis” transaction directly or indirectly.

After being approved by a large number of newer transactions, a transaction becomes part of the consensus of the tangle and is nearly impossible to alter. This is achieved by requiring each new transaction to perform a proof-of-work computation, that will be later explained in detail. This computation is a small and easy process for any transaction that will prove very expensive to spam or fork the Tangle after consensus has been confirmed.

In order for someone to connect and attach information and by extension Transactions to the public Tangle, first the publisher must connect to Full-Node of the Tangle Network. Full-Node is in a sense a public server where anyone with the link can create a connection and start sending Transactions through that Full-Node.

In figure 2.2 is shown an example of the structure of the Tangle. All vertices represent the transactions of the Tangle and the edges represent the connections between the transactions (directed to previous transactions). These edges are said to approve the transactions connected to. The ‘a’ vertex is the Genesis of the Tangle network, the first transaction to arrive at the tangle and is also the creation transaction, the root of the Tangle. The blue vertices represent the unapproved transactions of the Tangle, no validation has yet happened to these transactions to safely confirm they are secure. The grey vertices represent the tips of the Tangle, which are the newly added transaction that zero transactions are connected to them.

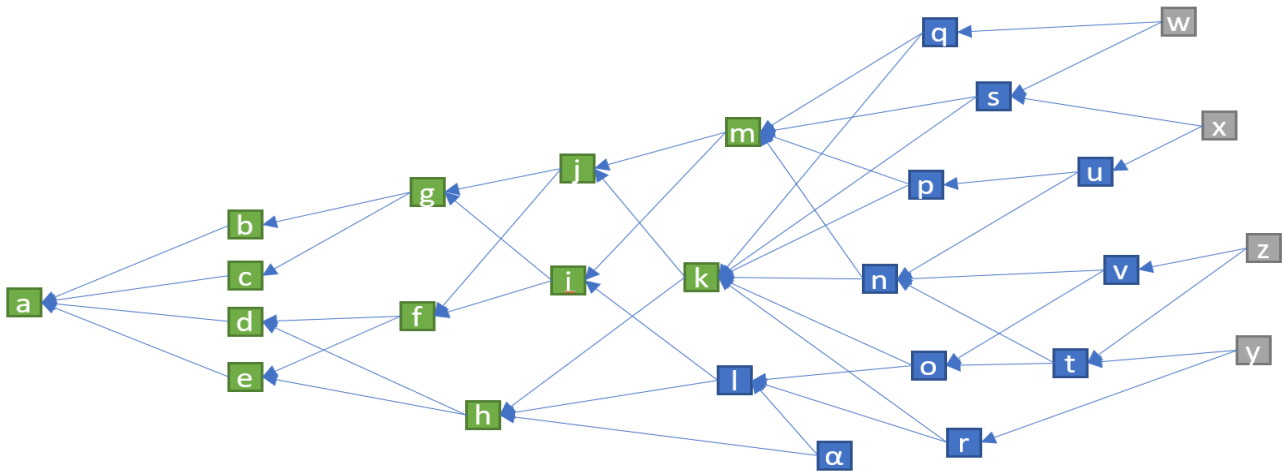


Figure 2.2 shows an example of the tangle's structure and all squares represent Transactions of the Tangle.

2.4 Transactions

In this subchapter we will explain in detail the meaning of Transactions and their function within the Tangle. We will also discuss how a Transactions is created, what is consists of and how a Transactions is attached to the Tangle.

2.4.1 What is a Transaction

Transactions in the Tangle, are in a group of transactions called “bundle of transactions” and each of these transactions require its own Proof-of-Work before being approved. Transactions is the way information is stored in the Tangle. Transactions can either contain data, messages, or even be used to send or receive IOTA's, the cryptocurrency of this network, but this won't be in any way used for this project.

2.4.2 Transaction Object

In this subchapter we will examine the Transaction Object of the format of every transaction in the Tangle and explain the importance and usage of every attribute of the object ^[5].

Hash: Uniquely identifies the transaction on the Tangle. This is generated by taking a hash of the trits of the address of the transaction.

signatureMessage Fragment: Holds either a signature or a message, which may be fragmented over multiple transactions in a bundle. If the value in the value attribute is less than zero, then this attribute contains a fragment of the signature authorising the spending of the IOTAs. If the value in the value attribute is greater than zero, then this value is a string message attached to the transaction. If the value in the value attribute equals to zero, then this value could be either a signature or a message fragment, depending on the previous transaction.

value: The amount of IOTAs spend for a transaction, the IOTAs that are being transferred. If this value is negative, then the address of the transaction is spending IOTA's, if it is positive, then the address of the Transaction is receiving IOTAs. If it is zero, then this transaction is being used to carry metadata, a signature fragment or a message fragment, instead of transferring IOTAs.

address: The address that is associated with this transaction. If the value attribute is greater than zero, then this attribute contains the recipient's address. If value attribute is less or equal to zero, then this attribute contains the sender's address.

timestamp: Unix timestamp when the transaction was issued.

currentIndex: The transaction's position in the bundle. If currentIndex equals to zero, then the transaction is called the "tail transaction" and it is the first transaction in the bundle. If currentIndex equals to the lastIndex, then the transaction is called the "head transaction" and it is the last transaction in the bundle.

lastIndex: The last transaction's position in the bundle. This is the total number of transactions in a bundle. This value is attached to every transaction to make it easier to traverse and verify bundles.

bundle: A hash value that identifies which transactions belongs to the same bundle. This value is created by combining all metadata from all transactions in the bundle.

tag: User-defined tag to easily find a transaction. Used to classify a transaction and helps when searching transactions using a Tangle explorer. It is optional and if no tag is declared then the attribute's value contains all 9's.

attachment Timestamp: The timestamp after the Proof-Of-Work is completed.

nonce: This attribute contains the solution for the Proof-of-Work. Nonce is required for the transaction to be accepted by the network.

peristance: This attribute indicates whether the transaction is pending or confirmed.

2.4.3 Types of Transactions

In a transaction bundle there can be up to three different types of transactions, output transactions, input transactions and meta transactions and are explained furthermore.

2.4.3.1 Input Transactions

These Transactions, as the name implies, are the Transactions that arrive at the receiver end with a variable size of value (IOTAs). For Input Transactions with negative value means that the complete balance of IOTAs from the sending address is spent, which makes them unavailable and invalid because no IOTAs can be validated as they were all probably used in previously approved Transactions. For Input Transactions with value greater than zero are the Transactions where previously unspent/ not used IOTAs are transferred to the recipient address

2.4.3.2 Output Transactions

These are the Transactions where IOTAs are send to one or multiple addresses, (will be discussed in later subchapters, for now let's say it's the physical address of a person issuing transactions with another person). These transactions are easily recognized because the transaction value is always greater than 0 and the address does not belong to the sender.

2.4.3.3 Meta Transactions

Zero value Transactions are called meta Transactions. These Transactions are mainly used to exchange messages within addresses that contain data. Meta Transactions will be later referred to as "messages" and will be more focussed in a different chapter for their usage in this Project and their ability to publish sensor data in the Tangle without the use of IOTAs.

2.4.4 Tip Selection

This is the process whereby you traverse the Tangle in a random walk from the genesis towards the first unapproved Transaction (Tip) it reaches. This process is done twice to randomly select two Tips which will be validated by the new Transaction, with the use of the Random Walk Monte Carlo algorithm. This algorithm's goal is to generate fair samples from some difficult distribution and is used when choosing the two unapproved Transactions and to safely determine if a Transaction is approved. The random walk is biased towards Transactions with more cumulative weight, or more Transactions approving them, this can be thought of as walking towards the heaviest branch. This selection method has the result of bigger branches of approved Transactions getting bigger and smaller branches shrinking and eventually disappearing, in the sense that no new Transactions will approve them. An example of this process of Tip Selection is shown in Figure 2.3.

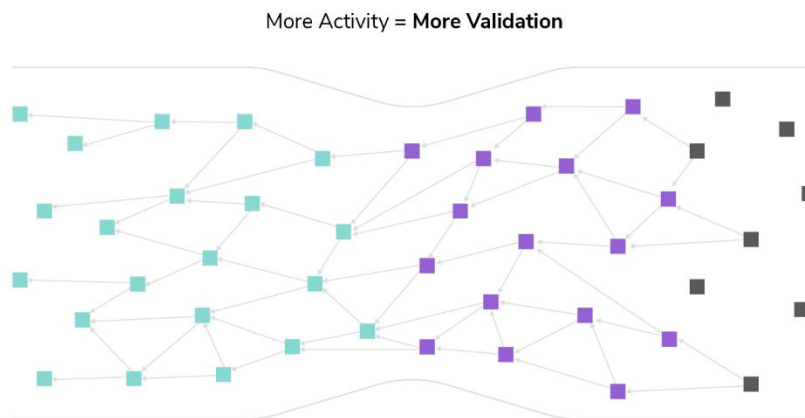


Figure 2.3 shows the approved transactions (navy blue), the yet approved transactions (purple) and the tips being selected (black). This figure shows that more activity, meaning more transactions incoming, the more validation will happen to the existing transactions making it a self-growing secureness and dependability in the Tangle.

2.4.5 Own and Cumulative Weights

Every Transaction has an initial weight called the “own” weight can have values such as 1,3,9 and etc.(3^n). This “own” weight is determined by the effort put by its issuing address.

The cumulative weight of a Transaction is the Transaction's “own” weight plus the sum of all weights of all transactions that directly or indirectly approve this transaction. It is a very important metric when Transactions are on their way for network approval. A transaction with a larger cumulative weight is considered more important than a smaller cumulative weight one, when selecting Transactions. Each new Transaction added to the tangle increases the ancestors cumulative weight by the

weight of that Transactions, meaning that older Transactions grow in importance over time. The use of cumulative weights avoids spamming, duplication of Transactions and other attack styles. It is assumed that no entity can generate an abundance of Transactions with “acceptable” cumulative weights in a short period of time, meaning that this strengthens this already secure process.

2.4.6 Proof-of-Work

The last step when issuing a new Transaction is the process of the Proof-of-Work. Once the bundle is constructed, signed and the tips are added to the bundle, the Proof-of-Work has to be done for each Transaction in the bundle. Every Transactions in a bundle requires a nonce, which is the result of the Proof-of-Work, in order to be accepted by the tangle network. The main purpose of the Proof-of-Work is to prevent spam, duplicated Transactions and Sybil-attacks.

When the Proof-of-Work begins its process it firstly gets the Minimum Weight Magnitude (explained later on). An IOTA transaction data is then encoded and stored in a string of 2673 trytes, called the transactionObjectTrytes. The last 81 trytes of the transactionObjectTrytes is reserved for the nonce. Then the execution of Proof-of-Work begins, using the transactionObjectTrytes and the Minimum Weight Magnitude as input, and then outputs the nonce which is 81 trytes in size as mentioned before and copied in the reserved spot in the transactionObjectTrytes and then the transactionObjectWithNonceTrytes is created.

Next the validation of the Proof-of-Work takes place. The transactionObjectWithNonceTrytes is then converted into trits and a CheckHash object is created and initialized, that holds the Curl hash algorithm result. This algorithm then outputs the result of the process into the CheckHash object. Then the Proof-of-Work must check whether the number of 0’s at the end of the CheckHash value are at least the Minimum Weight Magnitude. If that’s the case then the nonce of the Proof-of-Work is valid, if not then the process must be repeated until a valid result is arrived. A valid nonce is required in order for any Transaction to be accepted by the tangle network.

2.4.7 Minimum Weight Magnitude

The Minimum Weight Magnitude is the true difficulty of the Proof-of-Work process. Its value is the amount of PoW you have to do per Transaction in the bundle. In detail, this is the number of trailing zeros, which is the sequence of zeros in a representation of a number, where the Proof-of-Work must

be done until no other digits after the zeros follow, each time removing through a process a digit. Mainly its value is set to be 14, which is proven to be the golden line between protecting the Transactions in the tangle and the computational power needed to calculate the Proof-of-Work, the time required to finish. Bigger values mean more time to finish, which goes up exponentially, and ensure bigger percentage of protection while smaller values mean a lot less time to calculate and less protection insurance from attackers.

2.4.8 Bundle Construction

This is the first step when creating a new Transaction for the Tangle. This consists of the construction of the bundle of Transactions and the signature process for the input transactions with the sending address's private key.

IOTA uses a bundle which consists of multiple Transactions containing credits to the receiving addresses, outputs, and debits from the spending address. In IOTA there are two main types of transactions (explained in previous subchapter), one where a value is transferred and thus, have to sign inputs, and one where no value is transferred to an address, e.g. a message.

In this process of constructing a Bundle of Transactions, a BundleHash is created to ensure the safety when transferring the information in the bundle. In order to create a BundleHash the address, value, tags, timestamps, correctIndex and lastIndex attributes of each Transaction in the bundle are used. All the trytes of these values are calculated in a sum creating then the BundleHash. Then this sum goes through a process in order to create a normalized version of this hash that is used to create or validate a signature on the Transactions in the Bundle. This normalized hash contains no tryte value of 'M' and all the weights of the trytes are evenly distributed, hence the normalization.

2.4.9 Attaching the Transaction

The process of creating a new Transaction is split in three sub-processes, the construction of the bundle, the Tip selection and the Proof-of-Work.

First a bundle of Transactions is constructed, meaning that all Transactions that are relevant between them or carry fragments of the same message are combined together to create this bundle and essentially create a large Transaction Object. Then this bundle is signed with a private key, called an address

(will be explained in detail later), by our network default gateway where in our case study is the Raspberry Pi. The second step to this process is the Tip Selection and as explained before this process chooses two other unconfirmed Transactions, called Tips, using the Random Walk Monte Carlo algorithm to get the best candidates for approving. Lastly, the Proof-of-Work for all Transactions in the Bundle is done. This process does some calculations on the Transactions to decide whether these Transactions are not tampered with and are suitable to be attached to the Tangle. This is done by creating a hash of the whole bundle in order to add security to our attach of data. When all these steps take place, then the Transaction is considered valid and is attached to the Tangle and is now considered part of the whole network and will probably instantly used to approve new Transactions to the Tangle.

2.5 Seeds

In this part, we will explain the meaning of Seeds in the Tangle, what is a seed and how a seed is created. Also before explaining the aspect of a seed, we will make a reference to the IOTA wallet, which will be needed in the explanation.

An IOTA wallet is a way to access your IOTA funds, create Transactions and view the history of a seed's Transactions. The latest API officially provided from the IOTA foundation is the IOTA Trinity wallet.

2.5.1 What is a Seed

A seed can be considered as the equivalent of a username and a password or a unique access key, an identification process for every access to the IOTA network. A seed is created once per "account" and grants access to an individual's IOTA funds. A seed is 81 trytes long, trytes as mentioned before is the alphabet the IOTA protocol uses and consists of all 26 characters from the Latin alphabet and the number '9'. Every seed creates addresses that belong to that specific seed, considered in a sense to be the root of all addresses created for any Transaction a user will create or already created in the past.

Lengthy seed is what makes it so secure, for someone to access our IOTA wallet they need to know every single character in the seed, so as long as the seed is stored safely, the possibility of someone accessing your seed and with extension your funds and transactions, it is incomprehensibly small.

2.5.2 How a Seed is created

There are three official ways in creating an IOTA seed and any other option is considered an attack opportunity and a possible vulnerability for the Tangle Transactions.

The first way is to follow the step by step instruction guide in IOTA's new API, the IOTA Trinity wallet, which is available for IOS and Desktop environments.

Another way to create a seed is to create random 81 characters using command in both the Linux terminal or the Mac console, which this prerequisite some system's programming knowledge.

The command to create a seed in Linux and Mac are shown below:

Linux terminal: "cat /dev/urandom |tr -dc A-Z9|head -c\${1:-81}"

Mac console: "cat /dev/urandom |LC_ALL=C tr -dc 'A-Z9' | fold -w 81 |head -n 1"

And lastly, there is the option to manually create the characters by simply writing it down. This is recommended to be avoided as it is very dangerous and can create a seed that is already in use and can provide a back door to your whole funds.

When creating a seed, it is considered a hazard when using random generators, IOTA seed generators provided online or any other tool from sketchy web sites as it can prove to be traps or scums to access an individual's funds ^[8].

2.5.3 How a seed is used

Before establishing the usage of a seed we must clarify that during any transaction, whether receiving IOTAs or transmitting IOTAs to others, it is never directly used. When creating a transaction, for example to transmit some IOTAs to another wallet, the sender's seed is used to create an address to make that transaction.

2.6 Addresses

In this part, we will explain the meaning of addresses in the Tangle, how they operate, how they are valuable in the secure exchange of transactions, list the steps required to attach a new address, what happens when you do and we must avoid when using addresses.

2.6.1 What is an address

Before attaching an address to the Tangle we must explain what an address actually is. In order to generate an address, the seed is used. IOTA deals these generated addresses in a deterministic fashion. A deterministic system is a system in which no randomness is involved in the development of future states of the system, thus a deterministic model will always produce the same output from a given initial state. In this case the initial state is considered to be the seed and the deterministic process is that the same set of addresses are always generated in the same order. This accommodates the ease of no needing to store a private key file on a device because only the seed is needed which every time is used to reproduce these addresses, or keys.

Now will see the procedure to generate an IOTA address. Every address created has a corresponding key index number starting from 0, indicating their chronological order being created.

First as mentioned above, we use the seed by converting it to trits in order to generate any address.

Then a portion of the seed needs to be created, let's refer to it as "sub-seed", by combining the seed and the address's key index number. This sub-seed will be the root of that address, and each subsequent address can be accessed by having this root while all previously created addresses will not be accessible by this particular sub-seed, root.

The last part of creating an address is to add an address checksum. This checksum is an additional 9 trytes being added to the address that are used to validate the integrity and validity of each address. The final size of an address, including the checksum is 90 trytes long, 81 trytes for the address and 9 trytes for the checksum. In order to create this checksum we convert the 81 trytes of the address to trits and then hash this to create a new string value. This new hashed value is then converted back to trytes and the last 9 trytes of this string is considered to be the checksum and then lastly this is appended to the original address to create the final form of every address.

2.6.2 Attaching an address to the Tangle

When attaching an address to the Tangle is the same as attaching a Transaction Bundle to the Tangle. First a zero value transaction is created that references that address, then the Tip selection happens, selecting and approving two previously unapproved transactions from the Tangle and then finally doing the Proof-of-Work for that transaction.

2.6.3 Dangers when using the same address

No limit to the number of transactions an address can receive, but as soon as you've used funds from that address to make a transaction, this address should not be used anymore.

Every time you send IOTAs, meaning an output transaction that sends IOTAs to another individual, a part of the private key of that specific address is revealed because by receiving a Transactions you also receive that Transactions address. That address contains the sub-seed that created that address and every time we use the same address a portion of that sub-seed is revealed, meaning that the more we use the same address the more of its sub-seed will be shown creating an opportunity for malicious activity because the more percentage of the sub-seed is shown the more possible it is for someone to brute force and get the full private key, instantly gaining access to all funds on that specific address.

2.7 Coordinator

In this subchapter we will discuss about the coordinator and how is used to ensure the security of the tangle, what are milestones and what responsibility they have to this.

2.7.1 Usage of a Coordinator

In its infancy stage (early stage) the Tangle network has a small number of nodes. An attacker can relatively easily create many nodes on the network and thus creating many malicious transactions. In this case, if we were to use the RWMC method, there is a big chance that malicious transactions are selected. To protect the Tangle network against 34% attacks in its infancy stage a temporary protection mechanism is used called the Coordinator or COO for short. The COO is a special node run by IOTA foundation that is used to directly or indirectly validate the transactions (this will be done privately configured in our private tangle server).

Using the Coordinator does not mean that the network is centralized, the network is still 100% decentralized because every node on the network is verified by the Coordinator and they are not breaking consensus rules by creating IOTAs out of thin air or approving double-spending Transactions.

The Coordinator's process will be shut down as soon as the network reaches a certain number of confirmed transactions per second (ctps). When this number is reached many transactions are approving transactions all the time.

2.7.2 Milestones

Coordinator creates special Transactions approximately every 1 minute that are called milestones, and as every other Transaction these milestones must approve two other Transactions in order to enter the Tangle. Milestones are said to set the general direction for the Tangle's growth. Every node in the Tangle contains the field "latestMilestone", whose value, as the name implies, is the hash of the latest milestone received by the Coordinator. Each milestone creates a sub-tangle of Coordinator approved Transactions and every Transaction that is directly or indirectly referenced by a milestone is considered approved by the Tangle. This process allows for some monitoring to our tangle in its early stages where it is more vulnerable, where the number of incoming Transactions is still very small. In addition, these milestones cannot be faked because they are signed with a hash from the Coordinator.

2.8 Comparison with Blockchain

In this part we will do a short reference to the Blockchain implementation and do a comparison between these two technologies, listing their similarities and differences and why the IOTA protocol is the best candidate for our particular project.

2.8.1 Explaining the Blockchain

A Blockchain is decentralized ledger that can be considered as a database that is shared across a network of computers. Once a record has been added, a block has been added and given value, to the chain of blocks and it is very difficult to change, providing immutability to all the information that is stored ^[9]. To ensure all the copies of the database are the same, the network of connected peers does constant checks with the help of authorised parties. Blockchain was originally designed to serve as a secure holding place for the Transactions of the crypto-currency Bitcoin. Blockchain monitors and verifies Bitcoin transactions by calling upon a decentralized network of remote nodes, controlled by miners, that create the order in which transactions occur, take place and are valid. This network's algorithm ensures that every Transaction that takes place in the Blockchain is unique across all the network.

2.8.2 Comparing the two technologies

Now we will do a comparison of these two technologies, the IOTA protocol and the Blockchain ledger, listing their differences and similarities in their usage and fundamental components.

Differences:

IOTA protocol uses the DAG structure called Tangle, which is not the same as the blocks used in the ledger of the Blockchain. Blockchain's Transactions are grouped into blocks and stored in sequential chains, while the Tangle is a Directed Acyclic Graph with a constant stream of individual Transactions entangled together with the main purpose of scalability of its network of Transactions, this is shown in Figure 2.4. The IOTA tangle has no need for mining, in order to gather its tokens and attach Transactions, while Blockchain needs people to act as miners for the process of its Transactions' Proof-of-Work [1]. A blockchain has an inherent transaction rate limit, because all participants agree on the longest chain, and discard forks and side branches. The Tangle, on the other hand, allows different branches of the DAG to eventually merge, resulting in a much faster overall throughput.





TITLE	Blockchain	IOTA
Scalable	Less Scalability	Highly Scalable
Structure	Chain of Blocks (linear)	DAG
System	Decentralized	Distributed
Data Integrity	Ensures data integrity	Ensures data integrity
Data Validation	With every new block in the chain of blocks	With every new Transaction and extra val- idation through the Milestones
Authenticity		
Immutability		

Figure 2.4 Shows the table of comparison between Blockchain and IOTA.

Similarities:

Both of them use this concept of Transactions, with Blockchain storing them in blocks and the Tangle storing them in a Transactions bundle. Both technologies are affected by time, meaning the longer they both run the more Transactions they receive meaning the more secure both become over time. In both cases, tampering with data and eventually changing an already saved Transactions' values is very difficult to do ^[1].

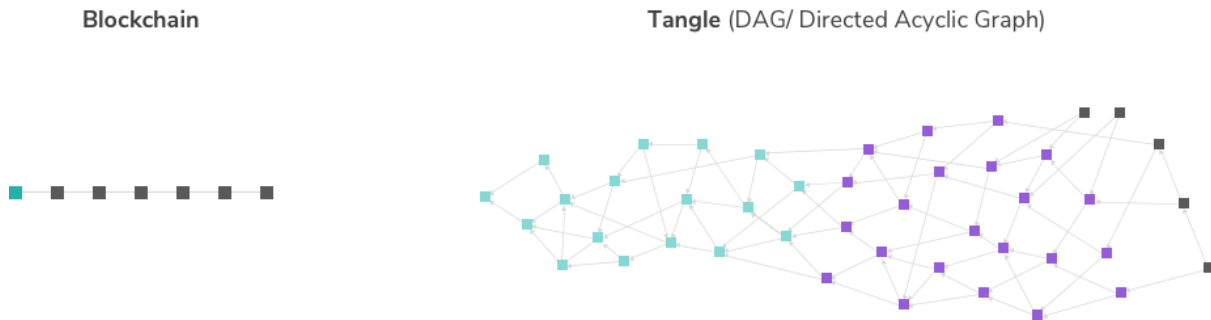


Figure 2.5 shows the structure of the Blockchain ledger and IOTA's Tangle.

2.8.3 Choosing the best candidate

It's clear enough that for the purposes of this project, the best candidate suitable to act as the main structure of our information is the IOTA protocol. The IOTA is considered by many to be the natural extension of Blockchain, as it is very similar in spirit but overcomes some of its fundamental limits. The Tangle does not have a built in maximum throughput, providing a never ending limit to scaling and the number of Transactions it can serve. In IOTA, as mentioned before, the component of miners does not exist therefore there is no incentives to slow the network down because of conflicts of interest and to raise fees as all Transactions send the full input amount to their destination address, while in the Blockchain ledger the miners get a percentage fee for their job to do this.

Blockchain technology promised a compelling vision: decentralized networks allowing open innovation and peer-to-peer transactions without intermediaries or fees. Ultimately, they were never built to execute it in full, due to inherent technical flaws in their design. As blockchain adoption has increased over the last decade, early adopters have been hit with sluggish transaction times and skyrocketing fees. As financial rewards for validating blockchain transactions became increasingly competitive, their networks have also become increasingly centralised around a few powerful actors. But the need for decentralized and permissionless systems remains, and has only increased in recent years.

By solving the inefficiencies of the Blockchain, IOTA, ^[9] based on the revolutionary distributed

ledger technology, the Tangle, is the missing link for the Internet of Everything and Web 3.0. Powering a secure, scalable and feeless transaction settlement layer, IOTA empowers machines and humans to participate in flourishing new permissionless economies, the most important being the Machine economy and by extension the Machine-to-Machine communication.

Chapter 3

The Private Tangle

3.1 Introduction.....	32
3.2 Components used.....	32
3.3 Difference Public-Private.....	33
3.4 Results.....	33

3.1 Introduction

In this chapter we will discuss the idea behind trying to create our own private Tangle and the components required to do this. We will also explain the differences of a private and public Tangle, listing the pros and cons of each implementation. And lastly, we will explain why this was not an applicant idea in the results sub-chapter.

3.2 Components used

In order to create our own private Tangle and use it to attach Transactions with messages critical for this project we used various implementations from the IOTA Foundation and community of IOTA provided mainly from the main IOTA repository in Github, the IOTA ledger ^[5]. In addition, we used a virtual private server provided by the IT department of the Computer Science department of the University of Cyprus in order to create a node that will store our Tangle. Lastly, we used a workstation to configure the connectivity with the server in order to program it and install the IOTA instance of our private Tangle implementation.

3.3 Difference Public-Private

The idea behind creating a Private Tangle was to create a secure structure, the Tangle, in order to be singly used for our selfish purposes to test the capabilities of this technology and the benefits of using this on IoT devices' communication.

The main difference of the private with the public tangle was that no other person or an unauthorized machine was able to access this and by extensions intervene with the data and Transactions being attached on the network. This was thought to be better for testing and monitoring and how such network is developing with new information constantly arriving where authenticity of data is ensured.

3.4 Results

Though this was originally thought to be the best suitable option for testing this technology it was later proven that was not sufficient to test the full capabilities of this technology and how it will benefit our purposes. The way the IOTA Public works is that basically, the more Transactions you have every second the more secure it becomes and the more accuracy of approving new Transactions is ensured, thus providing a constantly scalable network of Transactions.

Chapter 4

MAM and the Gateway

4.1 Introduction.....	34
4.2 Components used.....	34
4.3 Masked Authenticated Messaging.....	35
4.3.1 What is MAM.....	35
4.3.2 Mam Object.....	35
4.3.3 Message.....	37
4.3.4 Message Chain/Stream.....	38
4.3.5 Merkle Tree.....	38
4.4 Raspberry Pi Usage.....	39
4.5 Attaching Test.....	39
4.5.1 Establishing connection to the Public Tangle.....	40
4.5.3 MAM Example.....	41
4.6 Results.....	42

4.1 Introduction

In this chapter we will discuss the part about connecting to our private Tangle and sending specialized data as transactions on the network. We will also discuss about Mask Authenticated Messaging and their importance for this project. And lastly, we will see how the raspberry pi is used in this part of the project and see an example of uploading data and receiving them from the Tangle.

4.2 Components Used

In this part of my project I required a device that will act as a gateway of my local interconnected micro sensor network and at the same time as the sending device of Transactions to our private Tangle. The perfect option for this device was the Raspberry Pi. In order to connect the Raspberry Pi it is required to have a connection to the Internet, where in this case we use a Raspberry Pi WiPi wire-

less adapter, also some peripherals like a mouse, a keyboard, a monitor in order to program the device to our needs. Lastly, we used a feature provided from the IOTA Foundation, the Masked Authenticated Message that will be explained in detail in later subchapters.

4.3 Masked Authenticated Messaging

In this part of the report we will explain the meaning of Masked Authenticated Messaging, the MAM Object, discuss what is a Message and a Message Chain/Stream and how a merkle tree is created and how it helps in this process.

4.3.1 What is MAM

MAM stands for Masked Authenticated Messaging and it is a module built on top of IOTA protocol that makes it possible to send fully encrypted messages from authenticated parties or devices [8]. Masked means the message sent is encrypted, authenticated means the message sent is confirmed to be coming from a trusted device and messaging means that there is continuous message stream created in the Tangle that will carry on until the device stops publishing data into the Tangle.

IOTA MAM makes it possible for sensors and other devices to encrypt entire message streams and securely store these messages in the Tangle each on a separated address. Only authorized parties will be able to read and reconstruct the entire message stream. In essence it works a lot like a radio where only those with the right frequency can listen in. In MAM only those with the right channel ID (root) gains access to the message.

4.3.2 MAM Object

Below all the attributes of the Masked Authenticated Message Object is listed and explained in short form.

side_key: Used to encrypt and decrypt the message. When the mode attribute is set to “encrypted”, then the message is encrypted by this side_key. To decrypt the encrypted message, the same side_key is used. It is also used to encrypt the PoW nonce, the signature, the number of siblings and the siblings

mode: The publisher of the MAM object can choose between three channel modes, public, private and restricted. In Public mode, anyone has access to the message by simply having the address of the message. In this case the address is equal to the Channel ID of the message. In Private mode, only the publisher of the message has access to the message. In this mode, the address of the message is the hash of the message's Channel ID which can not be deducted. And lastly, the restricted mode, where anyone with the side_key can have access to the message, assuming that person already has the hash of the message's Channel ID. In this mode only someone that has the correct hash of the Channel's ID and the side_key gains access to the message and that is why we chose this mode for our message communication.

next_root: Using this next_root, we gain access to the next message in the message chain. In private and restricted mode the next address (next message) is calculated with the hash of this next_root. In public mode the next address is simply the value inside this attribute.

security: The publisher can choose between security level 1, 2 or 3 and this is the number of times the hashing process happens in the seed.

start: Start refers to the first leaf key index number in the Merkle tree. A leaf is the hash of the address of each message.

count: The number of leaves in the Merkle tree.

next_count: When a message is created, always 2 Merkle trees are created, the current Merkle tree and the next Merkle tree, accordingly one for the current message and the next message. Next_count is the number of keys used in the next Merkle tree.

index: Within the Merkle tree, the index number is used to process each leaf.

seed: It is the seed used for identification of the user and in the message chain, the Merkle trees always use the same seed.

payload: Also called masked payload. Consists of the actual message, for example sensor data, signature and other information.

root: The addresses are hashed. These hashed addresses are the leaves and each non-leaf node is a hash of its children. This results in a single hash called the Merkle ROOT. Each root represents an address where a message can be attached to the Tangle.

address: The address is used to attached the message to the Tangle. In private and restricted mode, the address is calculated as the hash(Root) while in public mode is simply the root.

4.3.3 Message

A Message is the format we will use when exchanging information from the network of sensors to the Tangle. This message is stored in the transaction object in the `signatureMessageFragment` attribute. When issuing a new data Transactions, meta Transaction, a zero value Transaction is created with a `currentIndex` value equal to zero and a `signatureMessageFragment` value containing the actual message. Subsequent Transactions with `currentIndex` value greater than zero, the `signatureMessageFragment` contains the subsequent parts of the message, this will be explained in detail below ^[11].

The `signatureMessageFragment` attribute always has a fix size of 2187 trytes, that are translated to approximately 1.27 Kbytes for every message in every data Transactions, as shown in Figure 4.4.

2187trytes converted to bytes:

- Bytes = (trytes x 3 x $\ln(3)$ / $\ln(2)$) / 8
- Bytes = (2187 x 3 x $\ln(3)$ / $\ln(2)$) / 8 = ~ 1300 bytes = ~ 1.27 Kbytes.

Figure 4.4 Shows the process to calculate the size of every Message in the `signatureMessageFragment` attribute of a data Transactions. The number 3 stands for the 3s value of the Trits and the division by 8 is the translation from bits to bytes.

If the `signatureMessageFragment` is empty the field contains all 9's and is considered the end of the Message Stream and that Transaction containing this value is the last Transaction for that exchange of data. If the signature or message is longer than 2187 trytes, the signature of message are fragmented and stored over multiple transactions in the transaction bundle. Each single transaction inside a transaction Bundle consists of 2674 trytes \approx 1.55Kbytes. If you have a message that is 20Kbytes in size, this message is fragmented and stored 20/1.27 (Size of `signatureMessageFragment` field where the message in Trytes is stored) that equals to 16 transactions inside the transaction bundle. Total transaction bundle size is 16 x 1.55(size of transaction) which is approximately 25Kbytes. For each of these 16 transactions a different PoW has to be done and a different nonce for every PoW is calculated. In order to convert ASCII characters to Trytes we use the IOTA javascript library (`iota.util.toTrytes`). This library uses a Base64 encoder that encodes any data in the `signatureMessageFragment` field. To restore the message to its original format the `iota.util.fromTrytes` is used in order to convert trytes to its Base64 representation, then use Base64 decoder to convert the data to its original format. If a message is fragmented and stored in multiple transactions we can

restore the raw tryte message by concatenating all fragments, starting with the transaction with currentIndex 0 inside the transaction bundle and decoding them all together.

4.3.4 Message Chain/Stream

Before getting into detail of how the Masked Authenticated Messaging works we need to explain the meaning of message chains in order to understand the works of Masked Authenticated Streams.

In a MAM stream or message chain, every message holds a reference to the next message.

A Message stream only flows in one direction. A subscriber with a channel ID has no access to the upstream messages, only to the messages being received in his end. In a Masked Authenticated Messaging stream the message is encrypted (masked) and the message also contains a signature, where this signature authenticates that the publisher indeed created this message. Any MAM exchange contains a channel ID which is also called the root of the message. This root is used when attaching the message to the Tangle and used when searching for a message stream already in the Tangle. When such messages are attached to the Tangle, the transaction containing the messages, the Meta Transaction, does not need to be approved by other Transactions.

4.3.5 Merkle Tree

This tree structure is the key element of Masked Authenticated Messaging. It is a hash tree in which each leaf node is a hash of a block of data and each non-leaf node is a hash of its children. This results in a single hash called the Merkle root. If every node has two children, the tree is called a binary hash tree and an example of this is shown in figure 4.1.

Below, the Figure 4.1 is an idea of counter structure of this implementation, an unsuitable option for this. The example says to examine the scenario where all hashes of the message's parts are appended creating a bus topology of connected hashes and then hash them all together to get one single hashed root (instead of a merrkle root).

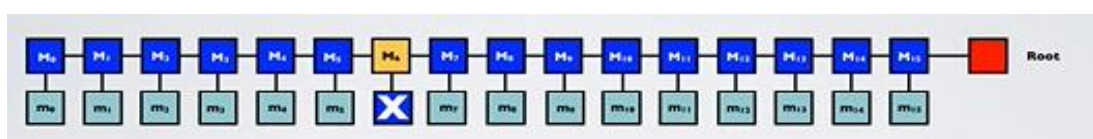


Figure 4.1 Shows a design of an example of a different structure rather than the one used in a Merkle Tree.

Let's say that Bob gets the root hash from a trusted source, who we shall name Alice. If Alice wants to prove to Bob that the message part "M6" is not tampered with, she needs to send message M6 and all other hashed messages to Bob. Bob then hashes message M6, appends all hashes to a single string and hash this string to get one root hash. Bob compares this new root hash with the trusted source root hash to check if message M6 is not tampered with. In this example, Alice has to provide 15 hashed values and the Message M6 to Bob to prove that message M6 is not tampered with.

This example ^[8] rises some serious issues for the structure, where n is the number of parts of a message, as the sender must resend the whole message again in order to prove authenticity of the whole message, hence resenting n parts. While in the Merkle Tree structure if the recipient suspects a part of message was tampered with then the receiver must sent only $2\sqrt{n}$ parts of the message, the height of the tree. This is explained below with an example of this structure and in Figure 4.2.

If Alice wants to prove that the message M6 ('X') is not tampered with, then she needs to send message M6 and 4 hash values (nodes in blue) to Bob. With these information, Bob calculates the root hash value and compares it with the trusted root hash received before to make sure that message M6 is not tampered with.

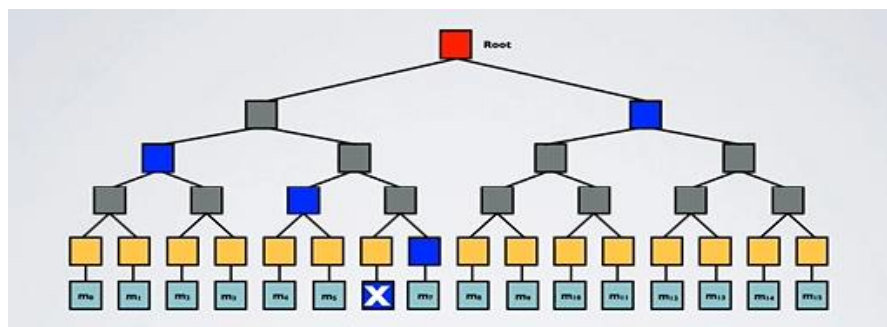


Figure 4.2 Above is an example of a merkle tree in a binary form with 16 parts (fragments) of a message. If it is suspected that only a single part was tampered with then only that part is affected.

Using a Merkle Tree provides integrity and validity of data send with MAM method by using a small amount of data that a trusted authority has to maintain. This also means that little memory is required and subsequently very little disk space is needed.

4.4 Raspberry pie Usage

The main purpose of usage of the Raspberry Pi in this project is to act the device that will send all the data collected from the sensors to the Tangle via the MAM method. The raspberry is connected with

the ending point of the network of sensors and collects all the data from it. In order for the Raspberry pi to be attach Transactions to the Public Tangle, we first must create a connection to a public node. In further sub-chapter it will be discussed the process of this connectivity and we will see an example of a MAM generated Transaction being published to the Tangle and later on being received.



Figure 4.3 shows the Raspberry Pi 2 that was used in this part of the project.

4.5 Attaching Test

In this sub-chapter we will see how the Raspberry Pi can attach data to the Public Tangle. Testing this method by first connecting the Raspberry Pi to a known public node ^[7] of the Public Tangle and attach the MAM examples. In this case, for the MAM representation of data we used randomly generated numbers in Json format ^[4] in order to be easily tested and implemented. As mentioned in previous sub-chapters, when sending MAM we create meta Transactions, meaning no IOTAs are spend at the process. Below we will see the process to publish and later on receive these MAM from the Tangle.

4.5.1 Establishing connection to the Public Tangle:

The first step to this experiment is to connect to known public node ^[7]. This comes with its own risks, as no guarantees are given for any node. Many nodes are unstable in their functionality and may not operate all the time as efficient as someone might think. For this experiment I used a public node from the availability list of the Tangle's Node website.

4.5.2 MAM example:

After ensuring the connection of the Raspberry Pi with a known node of the Tangle, we now need to create the data and encapsulate in Transactions using MAM. For this particular example, we create randomly generated numbers in Json format every 15 seconds and publish them to the Public Tangle [10]. The mode of the MAM object of these was chosen to be restricted, meaning we had to add a side key to the object in order only the people with the side key and of course the root of any address of Transaction can access any data published to the Tangle.

```
pi@raspberrypi:~/SendMAM/dht11-raspi3-master/dht11-raspi3-master $ sudo node mam_publish.js
json= { data: 45, dateTime: '28/04/2019 08:34:48' }
Root: CUEBKYYQACHCMQOQBUDMNJBKAXFUWALTAWELBQA9NABFHQLBUMZJG99YDMXFXT9NHMH9DVSQPEDMBWNQJ
Address: NVDBAXLDTFVXKLN0U0UHQSXV9UTLSNGR09BEFXTFEESZHHZZXN9ACVR9QPRLDHBBBKYYXVUIWKPMIQPDO
json= { data: 20, dateTime: '28/04/2019 08:35:31' }
Root: BLTVHKJATGUTOXZJWA0GGTKHHKBMSG9MQFCMFVQPRBQLG9WTLXIFOCZXPAYZUTGZBICGXNYLVRKCEJM
Address: HOEDDMVE099XKEFAMJLFFEA9SNQBXJTJVIGQI9GMHNXIHSOU00XQKPMBTLSC0KNZHMWTPRRJM9VIVTRWM
dateTime: 28/04/2019 08:34:48, data: 45, root: CUEBKYYQACHCMQOQBUDMNJBKAXFUWALTAWELBQA9NABFHQLBUMZJG99YDMXFXT9NHMH9DVSQPEDMBWNQJ
json= { data: 75, dateTime: '28/04/2019 08:36:01' }
Root: TWNVEEPPQ09GDNERNSSGEG9U0HIRKQUFQMYTFFGINMWFKHQBQUWYNXAPGDNHURNOGRWEDRHIKCDHOXHVVK
Address: JITFOFUQUBLMLE0UH9SYHFDKYRVOEIMMEGLMYIQBPHTW0YZHMNTQ09LFILEIOYLM0GJMLJPB9UWNUQVS
dateTime: 28/04/2019 08:35:31, data: 20, root: BLTVHKJATGUTOXZJWA0GGTKHHKBMSG9MQFCMFVQPRBQLG9WTLXIFOCZXPAYZUTGZBICGXNYLVRKCEJM
json= { data: 28, dateTime: '28/04/2019 08:36:31' }
Root: PAKYPCRLORYWDEQCDFZVJVQNCVMPJCJYHIOEMFJNS9SVNXLSD0ZDQLQSOHIURZSRSMWIFKDAZYMNJSXHWZRZ
Address: K9TAASDECREJHQDXCIPTJPPEXVARSGECFGSABDDQCSRJONOSLWMMGAJDBBVVYMTNDZKPGMSC9RGWIXANGK
dateTime: 28/04/2019 08:36:01, data: 75, root: TWNVEEPPQ09GDNERNSSGEG9U0HIRKQUFQMYTFFGINMWFKHQBQUWYNXAPGDNHURNOGRWEDRHIKCDHOXHVVK
json= { data: 51, dateTime: '28/04/2019 08:37:01' }
Root: 09IPRBZSQZIOK9CUMEQYRYUB9ZKDQLANLFRRVZNUHFFHVZRICBGINPYJKPTJPBDSULFMKVHAXVURMLTP
Address: PZNFGEYLVJYLHBLIMBYLLYLWADF0CPFPFBYCCMKUUFUFRAMTDNKSNIJQZCGTQCLKWSFWNNA9RVQ9T0YPB
dateTime: 28/04/2019 08:36:31, data: 28, root: PAKYPCRLORYWDEQCDFZVJVQNCVMPJCJYHIOEMFJNS9SVNXLSD0ZDQLQSOHIURZSRSMWIFKDAZYMNJSXHWZRZ
json= { data: 21, dateTime: '28/04/2019 08:37:31' }
Root: 9WB0L9VYLZRYQ9JSZ9GFRFXLDILBJXP9IZXVRDVCVUUDXPWKHJBMJD90MIL9GNPBCVJPZQZWFE00EZAMX
Address: BJGDZKDYTWSHIT0Z0WMLPPY90APZARPDUBNUATGGDAOHGNYAVAZDWQDJIKLOTQJAKMZNQ9BZREZNAXRQ
dateTime: 28/04/2019 08:37:01, data: 51, root: 09IPRBZSQZIOK9CUMEQYRYUB9ZKDQLANLFRRVZNUHFFHVZRICBGINPYJKPTJPBDSULFMKVHAXVURMLTP
^C
pi@raspberrypi:~/SendMAM/dht11-raspi3-master/dht11-raspi3-master $
```

Figure 4.4 shows the Json file and its contents that are being published to the public node in the Tangle.

In Figure 4.4 we see an example of a MAM being published to the Tangle [12]. As we can see, a root is created for every MAM object and an address for that Transaction containing the MAM object. By using any root of MAM we can then retrieve the particular MAM data and its subsequent MAM data. The lines containing the format “dateTime: ###/###/#### \$:\$:\$\$, data: ##, root: #####” is an indication that a MAM object was created with these fields as data, that are stored in the signature-MessageFragment of the Transaction, and the root confirmation that this MAM was validated and attached to the Tangle. The code for this process was written in Javascript and can be found in appendix A.

A terminal window titled 'pi@raspberrypi: ~/SendMAM/dht11-raspi3-master/dht11-raspi3-master'. The window shows the execution of 'sudo node mam_receive.js' followed by a long alphanumeric string. The output displays five lines of data, each with a timestamp and a data value. The data values are 45, 20, 75, 28, and 51. The terminal window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'.

```
pi@raspberrypi: ~/SendMAM/dht11-raspi3-master/dht11-raspi3-master
File Edit Tabs Help
pi@raspberrypi:~/SendMAM/dht11-raspi3-master/dht11-raspi3-master $ sudo node mam_receive.js CUEBKYQACHCMQOQBUDMNJBKAXFUWALTAWELBQA9NABF
HQLBUMZJG99YDMXFXTH9NHMH9DVSQPEDMBWNQJ
dateTime: 28/04/2019 08:34:48, data: 45
dateTime: 28/04/2019 08:35:31, data: 20
dateTime: 28/04/2019 08:36:01, data: 75
dateTime: 28/04/2019 08:36:31, data: 28
dateTime: 28/04/2019 08:37:01, data: 51
pi@raspberrypi:~/SendMAM/dht11-raspi3-master/dht11-raspi3-master $
```

Figure 4.5 shows the data being received to the Raspberry Pi from the public Tangle.

Now, in Figure 4.5 we can see the data that were previously published to the Tangle being received back to the Raspberry Pi ^[12]. This is done by again, connecting to the same public node that the publishing happens and then by simply providing a root of a MAM object we can get all the subsequent successfully published information from the Tangle, including the MAM of that root. As we can see the root being used is the first root of the MAMs being created, this means that all MAMs should be received, but the last created data shown in Figure 4.4 with data=21 is not received because no confirmation happen that this was indeed attached successfully to the Tangle. The code for this process was written in Javascript and can be found in appendix A.

4.6 Results

The testing of this particular example is considered a success. We managed to connect to a node ^[7] and successfully attach information that later on was received to our local component. The data that were attached used the highest method of encryption, the restriction mode of the MAM object, were only people with the side key and the root of the MAM object can access them, meaning only us. As we can see five different address are created in order to send these five Json ^[4] format data. This is done due to the issue explained in previous examples, of the dangers of attaching more than once with the same address, part of the root will be revealed. This successful experiment will be our basis for later sending actual sensor data from the sensor network through our Raspberry Pi to the Tangle.

Chapter 5

The Contiki wireless sensor network

5.1 Introduction.....	43
5.2 Components.....	44
5.3 Contiki OS.....	44
5.3.1 What is the Contiki OS.....	44
5.3.2 Features.....	44
5.3.2.1 Memory Allocation.....	45
5.3.2.2 Full IP Networking Stack.....	45
5.3.2.3 Power Awareness.....	45
5.3.2.4 Examples.....	45
5.3.2.5 Cooja Network Simulator.....	45
5.4 Tmote Sky.....	46
5.4.1 What is the Tmote Sky.....	46
5.4.2 Features.....	47
5.4.3 Use Cases.....	47
5.4.3.1 Smart Housing.....	47
5.4.3.2 Smart Cities.....	47
5.4.3.3 Emergency Situations.....	47
5.4.3.4 IOTA example.....	47
5.5 Broadcasting data.....	48
5.6 Collecting view.....	49
5.7 Attaching MAM to the Tangle.....	50
5.8 Retrieving data from the Tangle.....	51

5.1 Introduction

In this chapter we will make a reference to the Contiki Operating system ^[18] and how it helped in the progress of my project. We will discuss about the components used in this part of the project, the local network of sensors, discuss about the Tmote Sky and how is the best choice when testing for this implementation and how we connect these components with the raspberry pi and the configuration for acting as a gateway. We will also make some tests with broadcasting and collecting data across the network and provide the results at the end. After the collection of data is done, a connection is established to a node of the Tangle and we publish the data and later retrieve them.

5.2 Components

I used an instance of Contiki Operating System, called Instant Contiki, on my personal laptop to create the environment for programming the IoT devices, the micro sensors. In order to create a network of connected IoT devices we used the sensor devices called Telos B nodes that act as the Tmote Sky that were programmed in the Contiki OS environment. We lastly used the raspberry pi as the gateway of this network and we connected it with the sink node of the sensor network.

5.3 Contiki OS

In this subchapter we discuss about the Contiki Operating System, its basic features provided to all its users.

5.3.1 What is the Contiki OS

Contiki is an open source operating system, that can be freely used both in commercial and non-commercial systems, designed for the Internet of Things that we used when programming our micro sensors. In this implementation we used an instance of this OS, the Instant Contiki that provides an entire development environment in a single download that can access by using any Linux emulator platform. This OS is a powerful toolbox for building complex wireless systems that provides powerful low-power Internet communication that fully supports IPv6 and IPv4 ^[18]. With Contiki OS development is relatively easy and fast, Contiki applications are written in standard C programming language and by using the Cooja simulator Contiki networks can be emulated before burned into hardware, to provide a testing environment. In addition to all these, the Contiki developer community provides support for all its features and issues that may arise.

5.3.2 Features

Below there are listed the various features provided from the Contiki Operating System, how they are beneficial and why they are used in our project.

5.3.2.1 Memory Allocation

Designed for tiny systems, having only a few kilobytes of memory available. Highly memory efficient and provides a set of mechanisms for memory allocation.

5.3.2.2 Full IP Networking Stack

Contiki OS provides a full IP network stack, with standard IP protocols such as UDP, TCP and HTTP, in addition to the new low-power standards like 6lowpan, RPL and CoAP. Contiki IPv6 stack, developed by and contributed to Contiki by Cisco is fully certified under the IPv6 Ready Logo Program.

5.3.2.3 Power Awareness

Contiki OS is designed to operate in extremely low-power systems, systems that may need to run for years on a pair of simple AA batteries. Contiki provides mechanisms for estimating the system power consumption and for understanding where the power was spending to assist development and testing.

5.3.2.4 Examples

There are plenty of ready examples in the Contiki source code tree, found in Instant Contiki, to help get started with our own code. Examples of how to program network cores and how to interact with the platform hardware, broadcasting data or collecting data examples.

5.3.2.5 Cooja Network Simulator

Contiki devices often make up large wireless networks. Developing and debugging software for such networks could be tricky and dangerous. Cooja, the Contiki network Simulator, makes it tremendously easier by providing a simulation environment that allows developers to both see their applications run in large-scale networks or in extreme detail on fully emulated hardware devices and to monitor the transfer of data and do analytics on the exchange of data of interconnected devices.

5.4 Tmote Sky

In this subchapter we explain what the Tmote Sky is, its features, why it is used in my research and what use cases we derive from this.

5.4.1 What is the Tmote Sky

Tmote Sky is a low power wireless sensor that offers high data rate sensor network applications requiring ultra low power and having high reliability and ease of development makes it ideal for our implementation for testing for an IoT network of communicating devices ^[20].

In this project we used two MTM-CM5000-MSP TelosB sensors serving as Tmote sky sensors in our network, one broadcasting sensor data (temperature, humidity and uv light) and the other collecting the data and serving as the sink node, connecting as the end node with the raspberry pi that servers as the gateway of our network and the connector component with our Private Tangle.



Figure 5.1 shows the Tmote Sky used in this project.

5.4.2 Features

- Inter-operability with many different IEEE 802.15.4 devices.
- Integrated on-board antenna with 50m range indoors and 125m range outdoors.
- Optional Integrated humidity, temperature and light sensors.
- Ultra low current consumption.
- Fast wakeup from sleep.
- Programming data fed via USB and data collected via USB too.
- TinyOS support, also the implementation of communication and mesh networking.
- User and Reset button ^[13].

5.4.3 Use Cases

Here we list the various applications that this network of sensors can be used for,

5.4.3.1 Smart Housing

In a smart house, sensors can collect information about the temperature of a room, the humidity of a room, the motions in a room or just about anything and create a data that will be fragmented in the message and sent to the Tangle for storage, future usage or even instantaneously usage by receiving them. The attaching part ensures the authenticity of these information and provides a secure path to and from the Tangle.

5.4.3.2 Smart Cities

In smart cities, millions of these devices can operate at the same time, sending information to the Tangle and creating a network of sensor information so an AI decision making system can collect this approved information and make the appropriate actions.

5.4.3.3 Emergency Situations

In emergency situations, e.g. fire, these IoT devices are capable of delivering critical information to fire fighters during an emergency, monitor the occupancy, smoke, light and fire. Capable of tracking emergency crew inside the building and displaying the details inside the fire fighter's mask.

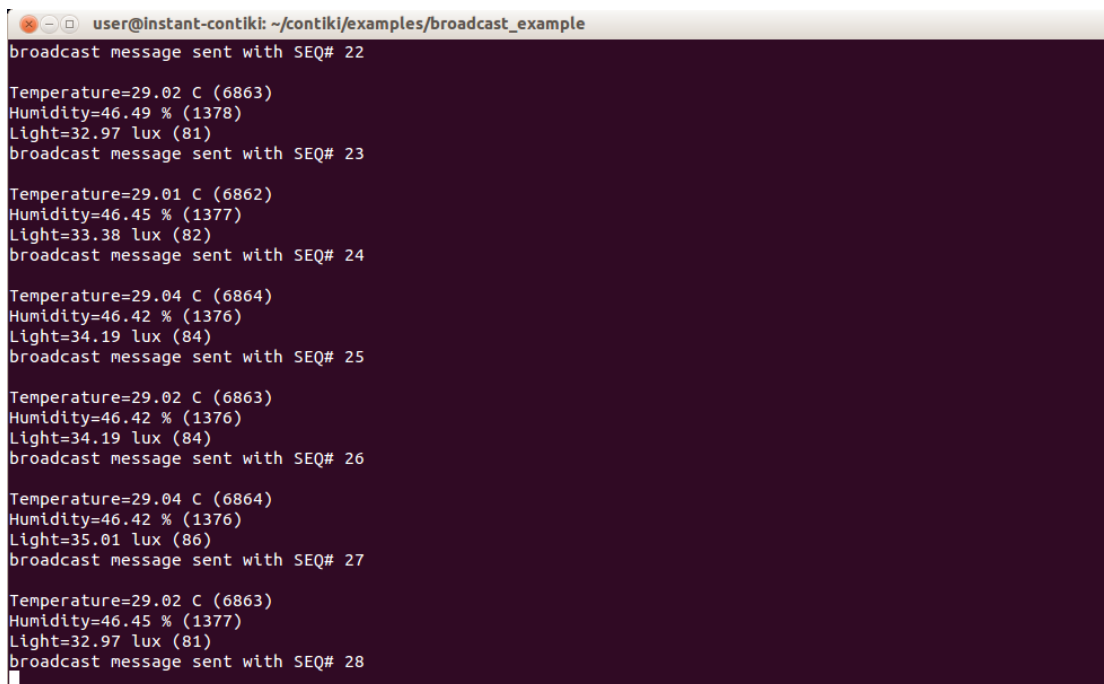
5.4.3.4 IOTA example

In a general example now, using the IOTA Tangle as the backbone of any distribution of data being send from various sensors and received from IoT devices we ensure a safe and secure path for transmitting these messages as well as high authenticity rate that these messages are indeed valid and trusted.

5.5 Broadcasting Data

Here we will explain the process I took when setting up one of the Telos B nodes to operate as the Broadcasting node of the network and how it collects the sensor data from the environment.

Using the examples of code inside the Instant Contiki OS environment ^[21] I programmed one of the devices to gather sensor data and broadcast this data across the network. The broadcasting IoT device acquires data from its sensors every 4 seconds and transmits the data on channel 26 for any device that listens to the same channel. The code for this is found on Appendix B.

A terminal window titled 'user@instant-contiki: ~/contiki/examples/broadcast_example' displays a series of broadcast messages. Each message consists of a confirmation line followed by three sensor readings: Temperature, Humidity, and Light. The data is shown for sequence numbers 22 through 28.

```
user@instant-contiki: ~/contiki/examples/broadcast_example
broadcast message sent with SEQ# 22
Temperature=29.02 C (6863)
Humidity=46.49 % (1378)
Light=32.97 lux (81)
broadcast message sent with SEQ# 23
Temperature=29.01 C (6862)
Humidity=46.45 % (1377)
Light=33.38 lux (82)
broadcast message sent with SEQ# 24
Temperature=29.04 C (6864)
Humidity=46.42 % (1376)
Light=34.19 lux (84)
broadcast message sent with SEQ# 25
Temperature=29.02 C (6863)
Humidity=46.42 % (1376)
Light=34.19 lux (84)
broadcast message sent with SEQ# 26
Temperature=29.04 C (6864)
Humidity=46.42 % (1376)
Light=35.01 lux (86)
broadcast message sent with SEQ# 27
Temperature=29.02 C (6863)
Humidity=46.45 % (1377)
Light=32.97 lux (81)
broadcast message sent with SEQ# 28
```

Figure 5.3 shows the sensor data the broadcasting device has gathered and the confirmation message that this data is being pushed to the network.

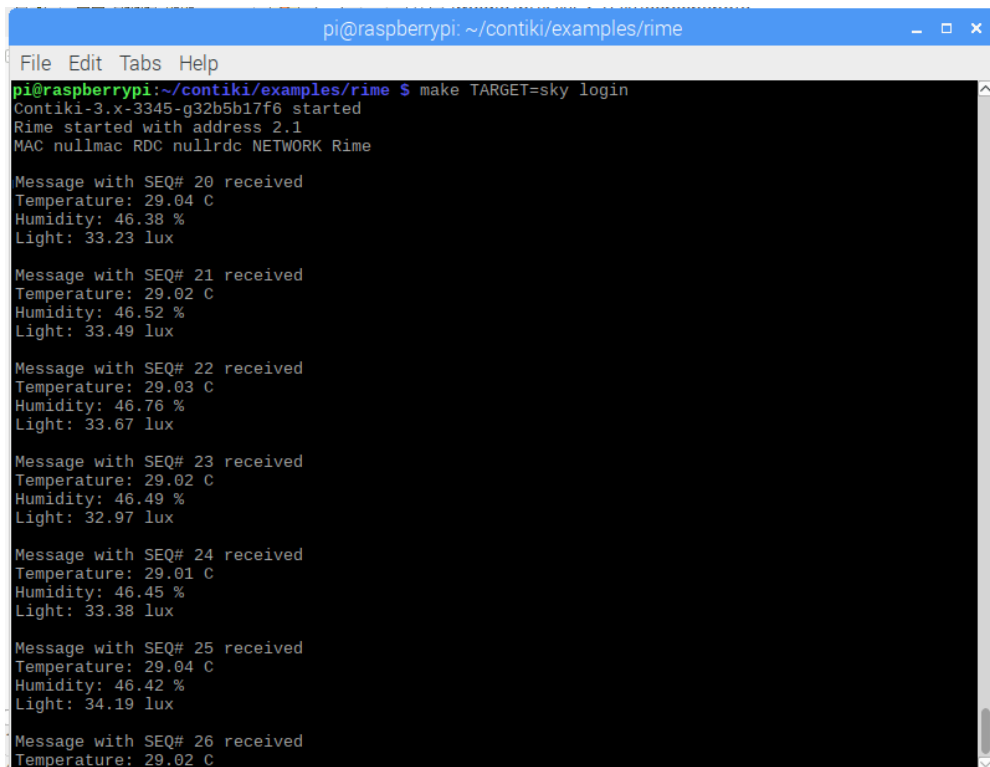
5.6 Collecting View

Here we will explain the process I took when setting up one of the Telos B nodes to operate as the collecting node of the network and what it does with the information received.

In a more mature network, we will have many more devices acting as T-mote sky and broadcast data between them, collected from all devices. But still we will need one of them to act as the sink node of the network, that will be connected with the gateway in order to send the data to the Tangle.

Using the examples of code inside the Instant Contiki OS environment ^[21] I programmed one of the devices to act as the sink node of the network and I connected it with the Raspberry pi. The sink node is set up to listen on broadcast channel 26 and when a message is received by any device, the message is shown on screen. This is done by connecting the IoT device to the Raspberry pi and running the command “make TARGET=sky login” in order to operate the sink node from the terminal. The code for this device is found in appendix B.

The sensor data collected is saved in a text file in the local file system of the Raspberry Pi to be later processed by the Javascript file responsible to create the MAM objects and sending them to a Public Node to be attached to the Tangle.



```
pi@raspberrypi: ~/contiki/examples/rime
File Edit Tabs Help
pi@raspberrypi:~/contiki/examples/rime $ make TARGET=sky login
Contiki-3.x-3345-g32b5b17f6 started
Rime started with address 2.1
MAC nullmac RDC nullrdc NETWORK Rime

Message with SEQ# 20 received
Temperature: 29.04 C
Humidity: 46.38 %
Light: 33.23 lux

Message with SEQ# 21 received
Temperature: 29.02 C
Humidity: 46.52 %
Light: 33.49 lux

Message with SEQ# 22 received
Temperature: 29.03 C
Humidity: 46.76 %
Light: 33.67 lux

Message with SEQ# 23 received
Temperature: 29.02 C
Humidity: 46.49 %
Light: 32.97 lux

Message with SEQ# 24 received
Temperature: 29.01 C
Humidity: 46.45 %
Light: 33.38 lux

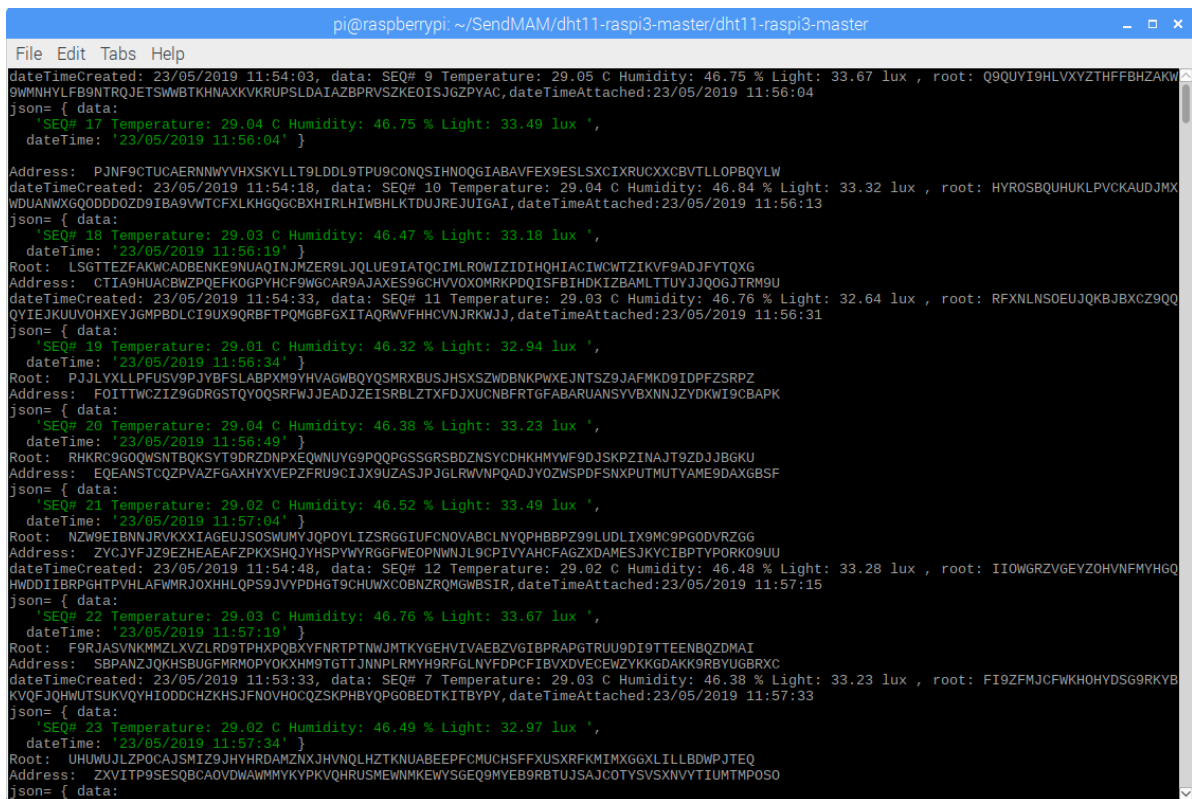
Message with SEQ# 25 received
Temperature: 29.04 C
Humidity: 46.42 %
Light: 34.19 lux

Message with SEQ# 26 received
Temperature: 29.02 C
```

Figure 5.4 shows the sink node gathering the sensor data that was broadcast from the other device.

5.7 Attaching MAM to the Tangle

In this part the Raspberry Pi creates a connection with a Public Node and attaches the data to the Tangle gathered by the sink node of our sensor network. The process of this is first to create a connection with a Public Node of the Tangle ^[7], to see if the selected one is available for attachment. Then, we access the information of the sensor devices which are located in a text file and we create a Json format for every sensor information we have stored. A mam object is created with this Json format as its signatureMessageFragment. For every mam object created, a different address is created (for reasons mentioned in previous chapters – dangers of addresses). When the mam object is created it is send to the Public Node for attachment ^[12]. The node sends feedback which will either be the time it was attached or some error message informing about the node's unavailability to attach. The code for this is found in Appendix C.

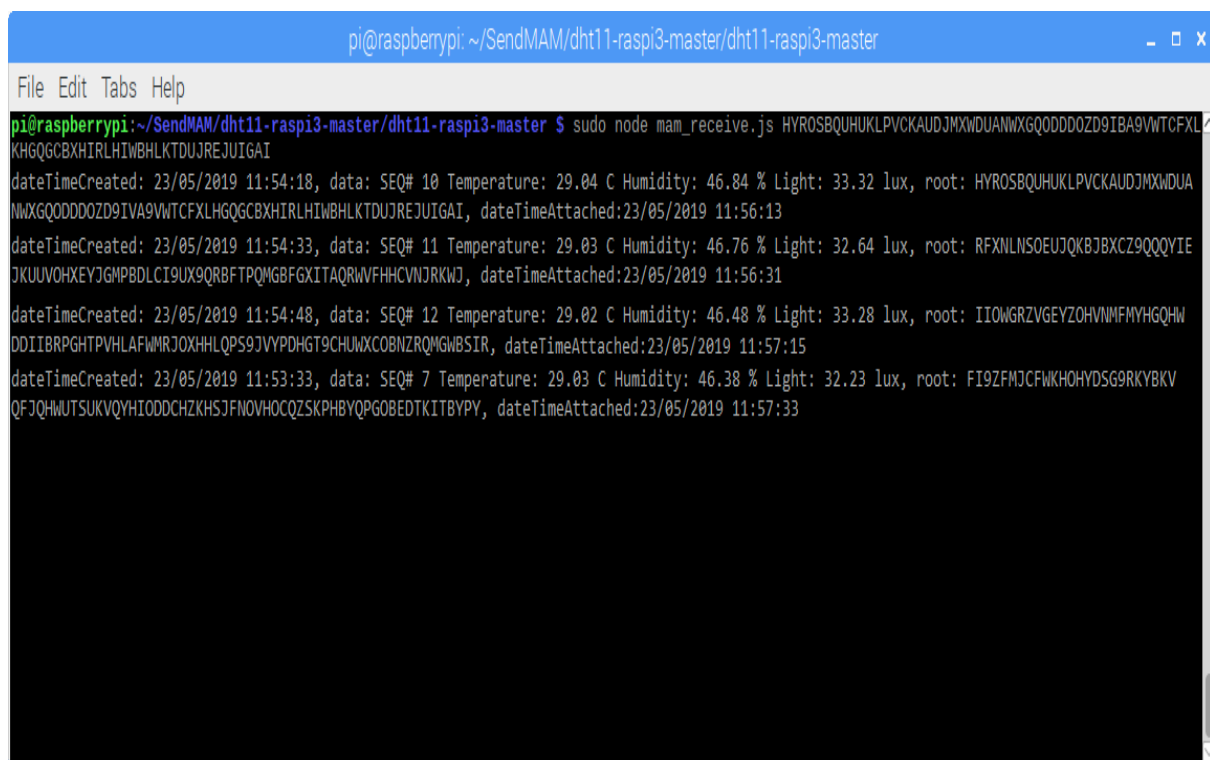


```
pi@raspberrypi: ~/SendMAM/dht11-raspi3-master/dht11-raspi3-master
File Edit Tabs Help
dateTimeCreated: 23/05/2019 11:54:03, data: SEQ# 9 Temperature: 29.05 C Humidity: 46.75 % Light: 33.67 lux , root: Q9QUYI9HLVXYZTHFFBHZAKW
9WMNHYLF89NTRQJETSMBTKHNAKKVRUPSLDAIAZBPRVSKZE0ISJ6ZPYAC, dateTimeAttached:23/05/2019 11:56:04
json= { data:
  'SEQ# 17 Temperature: 29.04 C Humidity: 46.75 % Light: 33.49 lux ',
  dateTime: '23/05/2019 11:56:04' }
Address: PJNF9CTUCAERNMMVYHXSXYLLT9LDDL9TPU9CONQSIHNOQGIABAVFEX9ESLSXCIXRUCXXCBVTLL0PBQYVW
dateTimeCreated: 23/05/2019 11:54:18, data: SEQ# 10 Temperature: 29.04 C Humidity: 46.84 % Light: 33.32 lux , root: HYROSBUHUHLKPVCKAUDJMX
WOUANWXGQDDDD0ZD9IBA9VWTCFXLKHGQCBXHIRLHIWBLKTDUJREJUI6AI, dateTimeAttached:23/05/2019 11:56:13
json= { data:
  'SEQ# 18 Temperature: 29.03 C Humidity: 46.47 % Light: 33.18 lux ',
  dateTime: '23/05/2019 11:56:19' }
Root: LSGTTEZFAKCADBENKE9NUAQINJMZER9LJQLUE9IATQCIMLR0WIZIDIHQHIACIWCWTZIKVF9ADJFYTXG
Address: CTIA9HUACBWZPQEFK06PYHCF9WGCAR9AJAXES9GCHVVOXOMRKPQDISFBIHDKIZBAMLTUUVJJQ0GJTRM9U
dateTimeCreated: 23/05/2019 11:54:33, data: SEQ# 11 Temperature: 29.03 C Humidity: 46.76 % Light: 32.64 lux , root: RFXNLNS0EUJQKBJBXCZ9QQ
QYTEJKUUV0HXYVJGMPB0LCI9UX9QRBFTPQMGBFGXITAQRWVFHHCNVJRKWJJ, dateTimeAttached:23/05/2019 11:56:31
json= { data:
  'SEQ# 19 Temperature: 29.01 C Humidity: 46.32 % Light: 32.94 lux ',
  dateTime: '23/05/2019 11:56:34' }
Root: PJJLVXLLPFUS9P9JYBFLAPXPM9YHVAGWBQVQSMRKBUSJHSXSZWBKBPWKEJNTSZ9JAFMKD9IDPFZSRPZ
Address: FOITTW0ZIZ9GDR6STQVQSRFWJJEADJZEISRBLZTFD9JXUCNBFRTGFABARUANSVYVBXNNJZYDKWI9CBAPK
json= { data:
  'SEQ# 20 Temperature: 29.04 C Humidity: 46.38 % Light: 33.23 lux ',
  dateTime: '23/05/2019 11:56:49' }
Root: RHKRC9G0QWNTBQKSYT9DRZDNPEQWNUY69PQPGSSGRSBDZNSYCDHKHMYWF9DJSKPZINAJT9ZDJJBKGU
Address: EQEANSTCQZPVAFZGAXHYVPEZFRU9CIJX9UZASJPJGLRWVNPQADJYQZWSPDFSNXPUTMYAME9DAXGBSF
json= { data:
  'SEQ# 21 Temperature: 29.02 C Humidity: 46.52 % Light: 33.49 lux ',
  dateTime: '23/05/2019 11:57:04' }
Root: NZW9EIBNNJRVKXXIAGEUJSOSWUMYJQPOYLIZSRGGIUFCONOVABCLNVQPHBBPZ99LUDLIX9MC9PG0DVRZGG
Address: ZYCJVFJZ9EZHEAEAFZPKXSHQJYHSPYVYRGFWGFEOPNWNJL9CPIVYAHCFAGZXDAMESJKYCIBPTYPOK09UU
dateTimeCreated: 23/05/2019 11:54:48, data: SEQ# 12 Temperature: 29.02 C Humidity: 46.48 % Light: 33.28 lux , root: IIOWGRZVGEY2OHVNFMYHGQ
HWDDIIBRPGHTPVHLAFWMRJ0XHLQPS9JYVDPHGT9CHUWX08NZRQMGWBSIR, dateTimeAttached:23/05/2019 11:57:15
json= { data:
  'SEQ# 22 Temperature: 29.03 C Humidity: 46.76 % Light: 33.67 lux ',
  dateTime: '23/05/2019 11:57:19' }
Root: F9RJASVNMKMLXVZLRD9TPHPQBXYFNRTPTNWJMTKYGEHVIVAE8ZVGI8PRAPGTRU9D9I7TEENBQZDMAI
Address: SBPANZJQKHSBUGFMROPYOKXHM9TGTJNNPLRM9YH9RFLNVDFPCFIBVXDVECEWZYKKGDAKK9BYUGBRXC
dateTimeCreated: 23/05/2019 11:53:33, data: SEQ# 7 Temperature: 29.03 C Humidity: 46.38 % Light: 33.23 lux , root: FI9ZFMJCFWKHOHYDSG9RKYB
KVQFJQHWUTSUKVQYHIODDCHKHSJFNOVHOCQZSKPHBVQPG0BEDTKITBYPV, dateTimeAttached:23/05/2019 11:57:33
json= { data:
  'SEQ# 23 Temperature: 29.02 C Humidity: 46.49 % Light: 32.97 lux ',
  dateTime: '23/05/2019 11:57:34' }
Root: UHUWJLZPOCAJSMIZ9JHYHRDAMZNXJHVNLQHZTKNUABEEPFMUCHSFFXUSXRFKIMXGGXLLILBDWPJTEQ
Address: ZKVITP9SESQBCA0VDWAWMMYKYPKVQHRUSMEWNKWEYSGEQ9MYEB9RBTUJSAJCOTYSVSNVYTIUMTPOSO
json= { data:
```

Figure 5.5 shows the data being attached to the Public Node through the Raspberry Pi.

5.8 Retrieving data from the Tangle

After publishing the data to the Tangle, for each mam object that was created an address and a root was issued that were different for every single mam. Now, this root can be used to retrieve the message published in that specific mam object and all subsequent attached messages ^[12]. Through the Raspberry Pi, we connect to the Public Node ^[7] that the data was published and we send the root Tryte number and the node responds after some time with the information that was published.



```
pi@raspberrypi: ~/SendMAM/dht11-raspi3-master/dht11-raspi3-master
File Edit Tabs Help
pi@raspberrypi:~/SendMAM/dht11-raspi3-master/dht11-raspi3-master $ sudo node mam_receive.js HYROSBQUHUKLPVCKAUDJMXWDUANWXGQODDDOZD9IBA9VWTCFXL
KHGQGCGBXHIRLHIWBLKTDUJREJUIGAI
dateTimeCreated: 23/05/2019 11:54:18, data: SEQ# 10 Temperature: 29.04 C Humidity: 46.84 % Light: 33.32 lux, root: HYROSBQUHUKLPVCKAUDJMXWDUA
NMXGQODDDOZD9IVA9VWTCFXLHGQGCGBXHIRLHIWBLKTDUJREJUIGAI, dateTimeAttached:23/05/2019 11:56:13
dateTimeCreated: 23/05/2019 11:54:33, data: SEQ# 11 Temperature: 29.03 C Humidity: 46.76 % Light: 32.64 lux, root: RFXNLNSOEUJQKBJBXCZ9QQQYIE
JKUUVQHXEYJGMPBDLCI9UX9QRBFTPQMGFBGXITAQRWVFHHCNVNJRKWD, dateTimeAttached:23/05/2019 11:56:31
dateTimeCreated: 23/05/2019 11:54:48, data: SEQ# 12 Temperature: 29.02 C Humidity: 46.48 % Light: 33.28 lux, root: IIOWGRZVGEYZOHVNMFMVYHGQHW
DDIIBRPGHTPVHLAFWMRJQXHHLQPS9JVYPDHGT9CHUMXC0BNZRQMGWBSIR, dateTimeAttached:23/05/2019 11:57:15
dateTimeCreated: 23/05/2019 11:53:33, data: SEQ# 7 Temperature: 29.03 C Humidity: 46.38 % Light: 32.23 lux, root: FI9ZFMJCFWKHOHYDSG9RKYBKV
QFJQHWUTSUKVQYHIODDCHZKHSJFNOVHOCQZSKPHBVQPG0BEDTKITBYPY, dateTimeAttached:23/05/2019 11:57:33
```

Figure 5.6 shows the data being retrieved from the Public Node to the Raspberry Pi.

Chapter 6

Conclusion

6.1 Introduction.....	52
6.2 Using the IOTA protocol.....	52
6.3 Modifications of Original Idea.....	52
6.4 Overall Conclusions.....	53

6.1 Introduction

In this chapter I created a synopsis, of all conclusions in my experience when working with this new technology, the IOTA protocol. The difficulties I faced experimenting with this technology and how I chose to proceed in facing them. Further I listed an overall conclusion with pros and cons with my experience with this project.

6.2 Using the IOTA protocol

Understanding this new technology, the IOTA protocol, took many hours of research and digging through all the APIs provided by the IOTA Foundation ^[16]. This new technology as mentioned before in previous chapters is a new, still in development, platform were many people contribute to ensure its future stability and usage. The option to work and experiment with this new technology was beneficial in understanding the future of IoT communication.

6.3 Modifications of Original Idea

The original option for this project was to create and operate on our own private Tangle in order to ensure that no other people or machines intervene in the structure of the Tangle. This was not applicable, as in order for the Tangle idea to work as intended a vast amount of Transactions are need and a time must pass to stabilize the function of the Tangle. This time required to pass is directly dependant to the amount of Transactions that happen to the network, the more valid and approved

Transactions are attached, the faster the network will stabilize and ensure better performance for future Transactions. We replaced the part of creating and using our own private Tangle, with simply attaching information through MAM to the Public Tangle, where we used the restricted mode that ensures only us, using our own private side key, can access these messages and by extension the Transactions containing these messages and then receive them back to our local machines.

6.4 Overall Conclusions

My conclusions of my whole experience of using this new technology, the IOTA protocol, are mostly positive. I understand how this structure created by the IOTA Foundation, the Tangle, and how it will be beneficial for the not so distant future of IoT communication. Though this new technology is still being created and not completely finished ^[17], in the sense that still people face small issues in creating and attaching Transactions, people still keep using it and the number of users and Transactions being attached is exponentially rising. The community of IOTA developers and freelancer developers contribute to fixing these little bugs and performance issues people are facing to ensure that future versions will be fully operational. Constantly updated to solve previous versions issues ^[6]. This development must be monitored carefully for future releases and versions that will fit specific purposes and scenarios because it has the potential of being the future method of IoT devices communication. Already many different implementations are being provided to the public via open source software ^[5] that are optimized for specific scenarios. In our particular project, I researched the possibility of creating our own private Tangle to store data collected from sensors but this didn't prove beneficial, as I mentioned before, in order for the Tangle to operate as it should, to provide a secure environment and authentic data and Transactions, a vast number of devices are needed to constantly attach new Transactions and approving previously unapproved ones. More Transactions attached instantly means more chance of successfully approved Transactions, thus creating a never ending cycle of constantly improved and ensured security for future Transactions. The only downfall for using such technology, is that an expert knowledge is required in order to understand how this works and how it can be manipulated to fit specific needs.

References

- [1] Analyst, I. (2018, July 09). The tangle vs blockchain: A comparison between IOTA and bitcoin. Retrieved from <https://www.ig.com/no/markedsnhyheter-og-analyse/trading-opportunities/the-tangle-vs-blockchain--a-comparison-between-iota-and-bitcoin-180709>
- [2] CETIC. (n.d.). Retrieved from <https://github.com/cetic>
- [3] Contiki-Os. (2018, November 03). Contiki-os/contiki. Retrieved from <https://github.com/contiki-os/contiki>
- [4] Introducing JSON. (n.d.). Retrieved from <https://www.json.org/>
- [5] IOTA. (n.d.). Retrieved from <https://github.com/iotaedger/>
- [6] IOTA Discord Server! (n.d.). Retrieved from <https://discordapp.com/invite/fNGZXvh>
- [7] IOTA nodes list. (n.d.). Retrieved from <https://iota.dance/>
- [8] IOTA tutorials. (n.d.). Retrieved from https://www.youtube.com/channel/UCG5_CT_KjexxjbgNE4IVGkg
- [9] IOTA » Blockchain. (n.d.). Retrieved from <https://blockchain.wtf/cryptocurrency-blockchain/iota/>
- [10] Iotaedger. (2018, November 14). Iotaedger/MAM. Retrieved from <https://github.com/iotaedger/MAM>
- [11] Lie, R. (n.d.). IOTA. Retrieved from https://www.mobilefish.com/developer/iota/iota_quick-guide_raspi_mam.html
- [12] Lie, R. (2018, July 16). Robertlie/dht11-raspi3. Retrieved from <https://github.com/robertlie/dht11-raspi3>
- [13] Tmote sky [Web log review]. (n.d.). Retrieved from <https://wirelessensornetworks.weebly.com/blog/tmote-sky>
- [14] The Contiki Operating System. (n.d.). Retrieved from <https://sourceforge.net/projects/contiki/>
- [15] The Contiki Operating System Developers' mailing list. (n.d.). Retrieved from <https://sourceforge.net/projects/contiki/lists/contiki-developers>
- [16] The IOTA Foundation. (n.d.). Retrieved from <https://www.iota.org/the-foundation/the-iota-foundation>

- [17] The Next Generation of Distributed Ledger Technology. (n.d.). Retrieved from <https://www.iota.org/>
- [18] The Open Source OS for the Internet of Things. (n.d.). Retrieved from <http://www.contiki-os.org/>
- Vidrih, M., & Vidrih, M. (2019, February 01).
- [19] IOTA or BTC? Tangle or Blockchain? What Has the Future? Retrieved from <https://medium.com/altcoin-magazine/altcoin-news-iota-and-btc-tangle-or-blockchain-what-has-the-future-67fe4b20d79a>
- [20] Tmote Sky [Web log review]. (n.d.). Retrieved from <http://tmote-sky.blogspot.com/>
- [21] Contiki tutorials. (n.d.). Retrieved from http://anrg.usc.edu/contiki/index.php/Contiki_tutorials

Appendix A

Below it is shown the code in Javascript, used for publishing random numbers to the public Tangle, using the Raspberry Pi.

```
const Mam = require('./lib/mam.client.js');
const IOTA = require('iota.lib.js');
const moment = require('moment');
const iota = new IOTA({ provider: 'https://knobbys-node-9.ddns.net:14267' });

const MODE = 'restricted'; // public, private or restricted
const SIDEKEY = 'mysecret'; // Enter only ASCII characters. Used only in restricted mode
const SECURITYLEVEL = 3; // 1, 2 or 3
const TIMEINTERVAL = 15; // seconds

// Initialise MAM State
let mamState = Mam.init(iota, undefined, SECURITYLEVEL);

// Set channel mode
if (MODE == 'restricted') {
    const key = iota.utils.toTrytes(SIDEKEY);
    mamState = Mam.changeMode(mamState, MODE, key);
} else {
    mamState = Mam.changeMode(mamState, MODE);
}

// Publish data to the tangle
const publish = async function(packet) {
    // Create MAM Payload
    const trytes = iota.utils.toTrytes(JSON.stringify(packet));
    const message = Mam.create(mamState, trytes);

    // Save new mamState
    mamState = message.state;
    console.log('Root: ', message.root);
    console.log('Address: ', message.address);

    // Attach the payload.
    await Mam.attach(message.payload, message.address);

    return message.root;
}

const generateJSON = function() {
    // Generate some random numbers simulating sensor data
    const data = Math.floor((Math.random()*89)+10);
    const dateTime = moment().utc().format('DD/MM/YYYY hh:mm:ss');
    const json = {"data": data, "dateTime": dateTime};
    return json;
}
```



```

const executeDataPublishing = async function() {
    const json = generateJSON();
    console.log("json=", json);

    const root = await publish(json);
    console.log(`dateTime: ${json.dateTime}, data: ${json.data}, root: ${root}`);
}

// Start it immediately
executeDataPublishing();

setInterval(executeDataPublishing, TIMEINTERVAL*1000);

```

Below it is shown the code in Javascript, used for receiving the messages that were previously attached to the public Tangle, using the Raspberry Pi.

```

const Mam = require('./lib/mam.client.js');
const IOTA = require('iota.lib.js');
const iota = new IOTA({ provider: 'https://knobbys-node-9.ddns.net:14267' });

const MODE = 'restricted'; // public, private or restricted
const SIDEKEY = 'mysecret'; // Enter only ASCII characters. Used only in restricted mode

let root;
let key;

// Check the arguments
const args = process.argv;
if(args.length !== 3) {
    console.log('Missing root as argument: node mam_receive.js <root>');
    process.exit();
} else if(!iota.valid.isAddress(args[2])){
    console.log('You have entered an invalid root: '+ args[2]);
    process.exit();
} else {
    root = args[2];
}

// Initialise MAM State
let mamState = Mam.init(iota);

// Set channel mode
if (MODE == 'restricted') {
    key = iota.utils.toTrytes(SIDEKEY);
    mamState = Mam.changeMode(mamState, MODE, key);
} else {
    mamState = Mam.changeMode(mamState, MODE);
}

```

```
// Receive data from the tangle
const executeDataRetrieval = async function(rootVal, keyVal) {
  let resp = await Mam.fetch(rootVal, MODE, keyVal, function(data) {
    let json = JSON.parse(iota.utils.fromTrytes(data));
    console.log(`dateTime: ${json.dateTime}, data: ${json.data}`);
  });

  executeDataRetrieval(resp.nextRoot, keyVal);
}

executeDataRetrieval(root, key);
```

Appendix B

Below it is shown the code, in a modified version of C for Contiki operations used for programming the Tmote Sky with collecting the sensor data and broadcasting them to all connected devices.

```
/*
 * Copyright (c) 2007, Swedish Institute of Computer Science.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * This file is part of the Contiki operating system.
 */

/**
 * \file
 *      Testing the broadcast layer in Rime
 * \author
 *      Adam Dunkels <adam@sics.se>
 */

#include "contiki.h"
#include "dev/light-sensor.h"
#include "dev/sht11-sensor.h"
#include <stdio.h>
#include <math.h>
#include "net/rime.h"
#include "random.h"
```

```

#include "dev/button-sensor.h"
#include "dev/leds.h"
struct sensor
{
    float frac;
    int dec;
    int val;
};
struct packet_data
{
    int SEQ_No;
    struct sensor data[3];
};
/*-----*/
PROCESS(example_broadcast_process, "Broadcast example");
AUTOSTART_PROCESSES(&example_broadcast_process);
/*-----*/

struct packet_data conv;
static void broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    printf("broadcast message received from %d.%d: '%s'\n",    from->u8[0], from->u8[1], (char *)packetbuf_dataptr());
}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;
int sequence=0;
/*-----*/
PROCESS_THREAD(example_broadcast_process, ev, data)
{
    static struct etimer et;
    static int val;
    static float s=0;
    static int dec;
    static float frac;

    PROCESS_EXITHANDLER(broadcast_close(&broadcast));

    PROCESS_BEGIN();

    broadcast_open(&broadcast, 129, &broadcast_call);

    while(1) {
etimer_set(&et, CLOCK_SECOND * 4 + random_rand() % (CLOCK_SECOND * 4));
        SENSORS_ACTIVATE(light_sensor);
        SENSORS_ACTIVATE(sht11_sensor);

        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

        val=sht11_sensor.value(SHT11_SENSOR_TEMP);
        if(val!= -1)
        {

```

```

        s=((0.01*val)-39.60);
        dec = s;
        frac=s-dec;
        printf("\nTemperature=%d.%02u C (%d)\n",dec,(unsigned int)(frac
*100),val);
        conv.SEQ_No=sequence++;
        conv.data[0].frac=frac;
        conv.data[0].val=val;
        conv.data[0].dec=dec;
    }
    val=sht11_sensor.value(SHT11_SENSOR_HUMIDITY);
    if(val!=-1)
    {
        s=((0.0405*val)-4)+((-2.8*0.000001)*(pow(val,2)));
        dec=s;
        frac=s-dec;
        printf("Humidity=%d.%02u %% (%d)\n",dec,(unsigned int)(frac*100),val);
        conv.data[1].frac=frac;
        conv.data[1].val=val;
        conv.data[1].dec=dec;
    }
    val=light_sensor.value(LIGHT_SENSOR_TOTAL_SOLAR);
    if(val!=-1)
    {
        s=(float)(val*0.4071);
        dec=s;
        frac=s-dec;
        printf("Light=%d.%02u lux (%d)\n",dec,(unsigned int)(frac*100),val);
        conv.data[2].frac=frac;
        conv.data[2].val=val;
        conv.data[2].dec=dec;
    }

    etimer_reset(&et);
    SENSORS_DEACTIVATE(light_sensor);
    SENSORS_DEACTIVATE(sht11_sensor);
    struct packet_data *sense=&conv;
    packetbuf_copyfrom(sense, sizeof(*sense));
    broadcast_send(&broadcast);
    printf("Broadcast message sent with SEQ# %d\n",conv.SEQ_No);
}

PROCESS_END();
}
/*-----*/

```

Below it is shown the code, in a modified version of C for Contiki operations used for programming the Tmote Sky with collecting the data being broadcasted from the other devices.

```
/*
 * Copyright (c) 2007, Swedish Institute of Computer Science.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of the Institute nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * This file is part of the Contiki operating system.
 */

/**
 * \file
 *      Testing the broadcast layer in Rime
 * \author
 *      Adam Dunkels <adam@sics.se>
 */

#include "contiki.h"
#include "net/rime.h"
#include "random.h"

#include "dev/button-sensor.h"

#include "dev/leds.h"

#include <stdio.h>
```

```

struct sensor
{
    float frac;
    int dec;
    int val;
};
struct packet_data
{
    int SEQ_No;
    struct sensor data[3];
};
/*-----*/
PROCESS(example_broadcast_process, "Broadcast example");
AUTOSTART_PROCESSES(&example_broadcast_process);
/*-----*/
static void
broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
    printf("broadcast message received from %d.%d: '%s'\n",
        from->u8[0], from->u8[1], (char *)packetbuf_dataptr());
    struct packet_data sense;
    packetbuf_copyto(&sense);
    printf("\nData:\n");
    printf("SEQ Number: %d\n",sense.SEQ_No);
    printf("Temperature: %d.%02u C (%d)\n",sense.data[0].dec,(unsigned
int)(sense.data[0].frac *100),sense.data[0].val);
    printf("Humidity: %d.%02u %% (%d)\n",sense.data[1].dec,(unsigned
int)(sense.data[1].frac *100),sense.data[1].val);
    printf("Light: %d.%02u lux (%d)\n",sense.data[2].dec,(unsigned
int)(sense.data[2].frac *100),sense.data[2].val);

}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;
/*-----*/
PROCESS_THREAD(example_broadcast_process, ev, data)
{
    static struct etimer et;

    PROCESS_EXITHANDLER(broadcast_close(&broadcast);)

    PROCESS_BEGIN();

    broadcast_open(&broadcast, 129, &broadcast_call);

    PROCESS_END();
}
/*-----*/

```

Appendix C

Below the code of connecting and publishing the Json format data to the public node of the tangle it is shown:

```
var fs=require('fs');
var textLines = fs.readFileSync('sensor.txt').toString().split("\n");
var lineIndex=0;
const Mam = require('./lib/mam.client.js');
const IOTA = require('iota.lib.js');
const moment = require('moment');
const iota = new IOTA({ provider: 'https://didiota.ddjros.net:14267'});

const MODE = 'restricted'; // public, private or restricted
const SIDEKEY = 'mysecret'; // Enter only ASCII characters. Used only in restricted
mode
const SECURITYLEVEL = 3; // 1, 2 or 3
const TIMEINTERVAL = 15; // seconds

// Initialise MAM State
let mamState = Mam.init(iota, undefined, SECURITYLEVEL);

// Set channel mode
if (MODE == 'restricted') {
    const key = iota.utils.toTrytes(SIDEKEY);
    mamState = Mam.changeMode(mamState, MODE, key);
} else {
    mamState = Mam.changeMode(mamState, MODE);
}

// Publish data to the tangle
const publish = async function(packet) {
    // Create MAM Payload
    const trytes = iota.utils.toTrytes(JSON.stringify(packet));
    const message = Mam.create(mamState, trytes);

    // Save new mamState
    mamState = message.state;
    console.log('Root: ', message.root);
    console.log('Address: ', message.address);

    // Attach the payload.
    await Mam.attach(message.payload, message.address);

    return message.root;
}

const generateJSON = function() {
    const dateTime = moment().utc().format('DD/MM/YYYY hh:mm:ss');
    const json = {"data":textLines[lineIndex++], "dateTime": dateTime};
    return json;
}
```



```
const executeDataPublishing = async function() {
  const json = generateJSON();
  console.log("json=",json);

  const root = await publish(json);
  const dateTime = moment().utc().format('DD/MM/YYYY hh:mm:ss');
  console.log(`dateTimeCreated: ${json.dateTime}, data: ${json.data}, root:
${root},dateTimeAttached:${dateTime}`);
}

// Start it immediately
executeDataPublishing();

setInterval(executeDataPublishing, TIMEINTERVAL*1000);
```

Below the code of retrieving this data from the public node of the Tangle it is shown:

```
const moment = require('moment');
const Mam = require('./lib/mam.client.js');
const IOTA = require('iota.lib.js');
const iota = new IOTA({ provider: 'https://didiota.ddjros.net:14267' });

const MODE = 'restricted'; // public, private or restricted
const SIDEKEY = 'mysecret'; // Enter only ASCII characters. Used only in restricted
mode

let root;
let key;

// Check the arguments
const args = process.argv;
if(args.length !==3) {
    console.log('Missing root as argument: node mam_receive.js <root>');
    process.exit();
} else if(!iota.valid.isAddress(args[2])){
    console.log('You have entered an invalid root: '+ args[2]);
    process.exit();
} else {
    root = args[2];
}

// Initialise MAM State
let mamState = Mam.init(iota);

// Set channel mode
if (MODE == 'restricted') {
    key = iota.utils.toTrytes(SIDEKEY);
    mamState = Mam.changeMode(mamState, MODE, key);
} else {
    mamState = Mam.changeMode(mamState, MODE);
}

// Receive data from the tangle
const executeDataRetrieval = async function(rootVal, keyVal) {
    let resp = await Mam.fetch(rootVal, MODE, keyVal, function(data) {
        let json = JSON.parse(iota.utils.fromTrytes(data));
        const dateTime = moment().utc().format('DD/MM/YYYY hh:mm:ss');
        console.log(`dateTimeAttached: ${json.dateTime},dataTi-
meRequested:${dateTime},data: ${json.data}`);
    });

    executeDataRetrieval(resp.nextRoot, keyVal);
}

executeDataRetrieval(root, key);
```