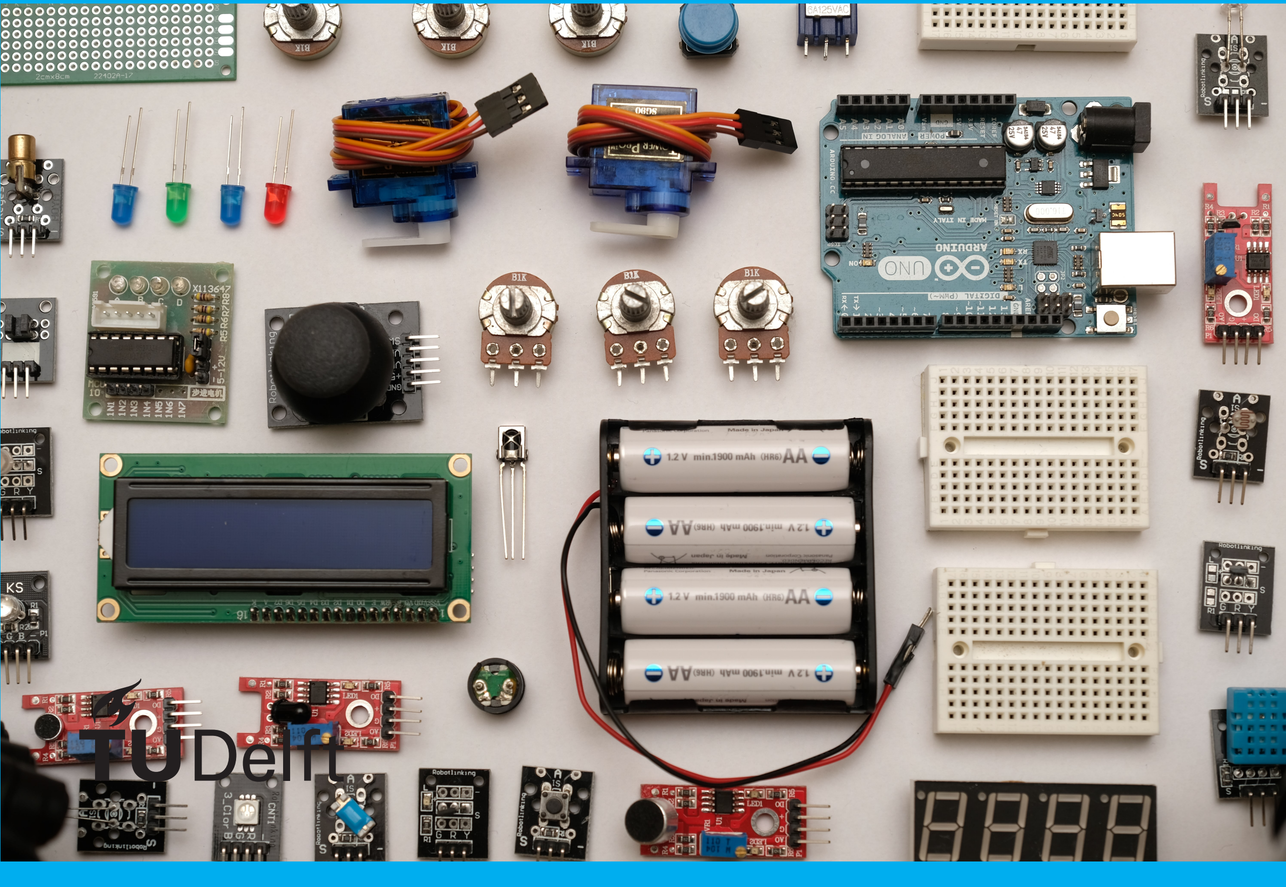
Bubblechain

An IoT Authentication system

J.R.S Da Camara



Bubblechain

An IoT Authentication system

by

J.R.S Da Camara

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 29, 2021 at 2:00 PM.

Student number:	4207858	
Project duration:	March 9, 2020 – June 29, 2021	
Thesis committee:	Dr. Z. Erkin,	TU Delft, supervisor
	Dr. K. Liang,	TU Delft
	Dr. S. Roos,	TU Delft

PREFACE

My thesis process has been a roller coaster of a ride. With lots of ups and downs. Fortunately, we have reached the end. I say we, because of all the help and support I have received along the way. First I would to thank my committee for taking their time to read my thesis. Next, I would like to thank my family, friends and girlfriend for understanding my absences for the duration of my thesis.

I would like to thank all of the Master students, Sharif, Daphne, Martin among others, which I had worked with on the 6th floor of the VMB building for making the process enjoyable. Thanks for all the moments, cakes, support throughout my thesis.

Next, I would like to thank Oguzhan "Ozzy" Ersoy for his patience, assistance and motivation during the process. You are a great person in general, a hell of a friend.

Last, but not least I would like to thank Zekeriya "Zeki" Erkin. You were a wonderful supervisor, super helpful and motivating, which was especially helpful in the midst of this pandemic. Glad to have had you as my supervisor. Please stay as kinds as you are mean to your future master students.

J.R.S Da Camara
Delft, June 2021

Abstract

Authentication of domains has been a crucial part of the growth of web browsing, especially for e-commerce and secure browsing. However, the digital space has expanded from web domains to include devices such as smart cars, smart houses, and other IoT devices. The future for these devices is to communicate autonomously with one another, also known as Machine-to-Machine(M2M) communication. For M2M communication, IoT devices need to be able to authenticate each other. However, IoT devices cannot take over traditional authentication mechanisms. This problem is due to the expected growth in the number of IoT devices and the varying resource capabilities of these devices. In this work, we introduce Bubblechain, a generic, decentralised, and scalable IoT authentication system. In contrast to existing works, Bubblechain can update and remove identities of IoT devices that allows them to authenticate each other in a dynamic setting. Bubblechain also creates a higher trust level for devices that belong to the same Bubble. A Bubble refers to a group of devices that belong together, such as a home or office setting. The system also provides the possibility to authenticate devices from different Bubbles. The authentication process in Bubblechain is autonomous and decentralised.

CONTENTS

LIST OF FIGURES

List of Tables

1 Introduction

In 2014, Google acquired Nest Labs for $3.2 billion [1]. Nest Labs merged with Google Home to form Google Nest. Google Nest sells smart devices such as thermostats, doorbells, locks, and alarm systems. Google is not the only tech company to invest in the Internet of Things (IoT). Similarly, Samsung has a SmartThings division, which has 62 million active users [2].

One thing that the IoT divisions of these companies have in common is that they focus on Smart Homes. The definition of a *Smart Home* is the integration of technology and services through home networking for a better quality of living [3]. To increase the quality of living, it might be helpful if devices could communicate autonomously. For example, a person's smart car communicates with the person's thermostat to regulate the temperature when the person left the house or is driving home. This form of communication is referred to as Machine-to-Machine (M2M) communication.

Examples of IoT devices within a Smart Home are smart speakers, smart fridge, and smart meters. Besides Smart Homes, IoT can be found in other sectors as well, e.g. Smart Cars, with Tesla as a notable example. In addition, IoT devices have branched out to many sectors such as healthcare, retail, finance, agriculture, and energy. With the applications for IoT devices growing, the number of devices is also expected to grow. In 2014, The technology research firm Gartner predicted that there would be 26 billion IoT by the year 2020 [4].

With all these types of devices, the data generated can be sensitive. Think of the location of a smart car. This information can inform an adversary of a person's location. With all the different types of devices in mind, this question begs to be asked: "What about the security of these devices?".

1.1. IoT Security

The incorporation of IoT devices has led to certain security incidents. For example, the data generated by smart meters can help determine which home appliances are in use [5]. Based on the appliances in use, an adversary can interpret the user's habits. The moment a user turns on or off their lights could indicate their sleeping pattern.

Smart cars have also faced security incidents. Vulnerabilities on the internet connected entertainment systems have made it possible to remotely take over the manoeuvrability of a Jeep [6]. Smart cars have also faced the problem of theft via their keyless entry systems [7, 8]. Smart cars could face privacy concerns in the event of compromised location data. Achieving security is not as simple as adding encryption. Other aspects, such as accessibility and authentication, should also be taken into account. In this thesis, we focus on the aspect of authentication.

1.2. IoT AUTHENTICATION

As IoT is becoming a more integral part of our lives, ordinary objects are equipped with electronic devices to facilitate our lives. With the goal of achieving widespread M2M communication, verifying if communication is carried out with the correct device is crucial for our privacy. Verifying the identity of a user or process is called authentication. With the number of devices, a centralised approach to authentication would not be feasible and would have the drawback of having a single point of failure. Therefore we discuss two state-of-the-art solutions for decentralised IoT authentication.

1.2.1. BUBBLES OF TRUST

Bubbles of Trust is a decentralised Ethereum-based authentication system for IoT devices [9]. The system groups devices, under the term Bubble. Where the Bubble acts as a trust barrier for the group. Within each Bubble, one device is appointed to manage identities, while the others use these identities to authenticate each other. Figure 1.1 depicts a high-level overview of the system.

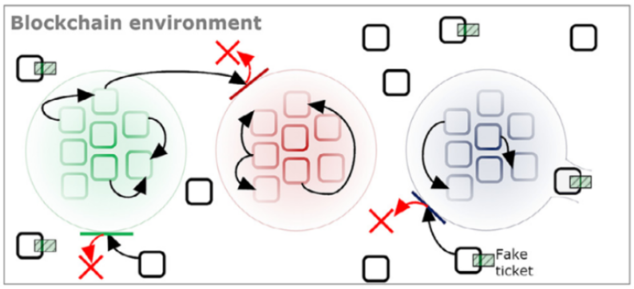


Figure 1.1: High-Level overview of the Bubbles of Trust system [9].

The communication is done via transactions, while a smart contract handles the authentication of identities within the system. The system is resilient against sybil, message replay,and denial of services (DoS) attacks. Sybil attacks are when an adversary subverts the reputation system by creating additional identities to gain an advantage. As the name already says, message replay attacks are attacks that reiterate previous messages. DoS attacks are when an adversary seeks to make a resource unavailable, either temporary or indefinitely. In Chapter 2, we explain this scheme in detail.

1.2.2. Decentralized Identity and Trust Management Framework for IoT

Decentralized Identity and Trust Management Framework is an Identity Management System (IdMS) for IoT devices [10]. The framework is based on a Web of Trust, allowing arbitrary identities to be trusted. These identities are self-sovereign identities (SSI). While the IdMS utilises an IOTA Tangle to access and store data. The goal of this framework is to create trust between identities. Which will help facilitate M2M communication between IoT devices.

Figure 1.2 shows a high-level overview of the system. The framework sought out to achieve its goal by setting objectives of scalability, permissionless, interoperability, security, automatic trust calculation, and privacy. The problem with SSI is that they are not worth much if they cannot show whether they are valid. Therefore the framework incorporates claims about each identity. Claims are statements about an identity, either from the owner or other parties. Next to claims, the system has attestations that are personal opinions about a claim. A trust score is calculated for each identity based on the identity's claims and attestations.

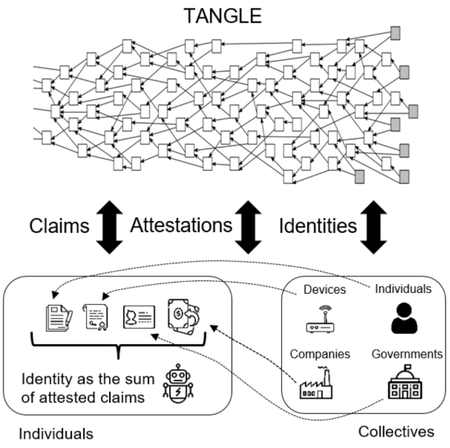


Figure 1.2: High level overview of the Decentralized Identity and Trust Management Framework for Internet of Things [10].

The IdMS operates at a low cost of entry, making the system accessible to everyone. The system is also capable of removing or reducing the trust of identities of corrupted devices. At the same time, new devices can start with a list of already trusted devices.

On the other hand, the low cost of creating identities makes it so that malicious actors can create more identities to raise their score, and carry out a Sybil attack. The low cost also makes it tempting for corrupted identities to be disregarded for new identities. The action of dropping corrupted identities for new ones is called White-washing.

1.3. REQUIREMENTS FOR IOT AUTHENTICATION

In this thesis, we aim for a decentralised IoT authentication system that is **scalable**, **generic**, **has different trust levels**, and **incorporates the life-cycle of devices**, as we argue below:

The expected number of IoT devices in the billions makes scalability a must. A centralised solution could not cope with the expected number of devices, making decentralisation the only option. Decentralisation has the benefit of being more fault-tolerant than a centralised system. In the event of a system failure, a different entity can maintain the system running. Additionally, decentralised solutions can provide transparency, which can increase trust.

With the expected growth in number of IoT devices, there will also be an increase in types of devices. Devices come in all shapes and sizes, such as smartphones, smart meters, smart cars, or smart fridges. The resource capabilities of these devices may vary. However, as M2M communication is the goal, these different types of devices should all be able connect.

M2M communication can help with the mission of having technology improve our quality of life. The devices of a person working together can excel in the areas of efficiency and performance. The previous example of setting the home temperature can save resources while increasing the person's quality of life, despite the sensitivity of the generated data. Devices should be able to communicate while being able differentiate between devices that belong together and foreign devices. Essentially, creating determining which devices should be trusted or not.

With time, devices can either break or be replaced by a newer version. For example, the lifespan for a smartphone is about two and a half years [11]. Therefore, an IoT system needs to be able to remove, add or update devices.

1.4. RESEARCH QUESTION

Given the interest in and the relevance of the topic, this thesis dives deeper into IoT authentication. With the previously stated requirements, the research question (RQ) for this thesis is:

> **"How can we design a decentralised generic authentication mechanism for IoT devices that incorporates different trust levels for these devices and also incorporate the life-cycle of devices?"**.

The research question can be divided into the following sub-questions :

1. How can we apply an authentication technique that would be suited for M2M communication?
2. How can the a IoT authentication mechanism in question incorporate different trust levels for these devices?
3. How to make this IoT authentication mechanism decentralised?
4. How can individual devices authenticate identities?

5. How can the IoT authentication mechanism be scalable, yet still be able to incorporate the life-cycle of devices?

1.5. CONTRIBUTION

In this thesis, we present Bubblechain. Bubblechain is a decentralised IoT authentication system that utilises the Ethereum blockchain to authenticate devices. Bubblechain compresses the identities into a Merkle Tree structure and uses Elliptic Curve Cryptography for communication between devices. In summary, Bubblechain provides:

- A scalable IoT authentication system that scales based on the number of owners instead of devices.
- Bubblechain also can establish communication with all devices while having a trusted barrier for devices that belongs to the same owner.
- Bubblechain also considers the life-cycle of devices, making it possible to add, remove, and update identities.
- Within the Bubblechain system, the only moment that all devices need to be online is when managing identities.

1.6. OUTLINE

The structure of this thesis is composed in the following way. The first part discusses the determined requirements based on existing solutions. Chapter 2 presents the security requirements for each IoT layer, a taxonomy of IoT authentication, and related work. Chapter 3 discusses the cryptographic preliminaries that are used, as well as blockchain technology, with a focus on the Ethereum blockchain. Chapter 4 covers Bubblechain, our decentralised IoT authentication protocol. Chapter 5 analyses our proposed solution with regards to security and performance. Additionally, we provide a comparison to a closely related work. Finally, we discuss the design choices and obtained results, provide an outlook for the future of Bubblechain, and conclude in Chapter 6.

2

IoT Authentication

In this chapter, we present state-of-the-art IoT authentication. First, we elaborate on layers for IoT and their security requirements. Secondly, we explain a taxonomy on IoT authentication schemes. Third, we elaborate on existing works. Finally, we present a closely related work, Bubbles of Trust, in-depth.

2.1. IoT Layers

In this section, we elaborate on the fundamental architecture model for IoT space. According to literature, the basic model consists of three layers: perception, network and application layer [12–15]. We finalise the section by summarising the security requirements for each layer in Table 2.1.

The perception layer refers to the physical layer that senses the environment to perceive the physical properties, with examples such as temperature, speed, location. The perception layer consists of sensors that are characterised by limited processing power and storage capacity [16]. Due to these limitations, the perception layer is susceptible to attacks, such as Denial of Service (DoS), Distributed Denial of Service (DDoS), Sybil attacks [17]. DoS is a type of attack that shuts down the system or network and prevents authorised users from accessing it. DDoS is a large scale variant of DoS attacks. Sybil attacks are when an attacker can deploy fake identities using fake nodes. The perception layer has the following security requirements: lightweight encryption, authentication, key agreement, and data confidentiality.

The network layer is in charge of getting the data from the perception layer and transmitting it to the application layer [17]. Several different network technologies are used for the transmission of data. WiFi, Bluetooth, 4G are examples of such technologies. With the responsibility of transmitting data, the layer is targeted by attacks such as Man-in-the-Middle (MITM) attacks, DoS, and eavesdropping. Communication security, routing security, authentication, key management, and intrusion detection are security requirements for the network layer.

The application layer handles the delivery of application-specific services to the user [17]. In other words, the application layer is responsible for providing services. The problem arises that "traditional" application layer protocol does not perform well within IoT, several issues arise at the application layer [18]. Issues such as data accessibility and authentication, data privacy, and dealing with big data. The authentication and accessibility of data could be impacted by fake or illegal users, making the need for permission and access control [17]. The fact that IoT connects to different manufacturers leads to the application of different authentication schemes.

Table 2.1: Security requirements for each IoT layer.

Layer	Security Requirments
Perception	Lightweight encryption
	Authentication
	Key Agreement
	Data Confidentiality
Network	Communication Security
	Routing security
	Authentication
	Key Management
	Intrusion Detection
Application	Authentication
	Data Privacy
	Permission and Access Control

2.2. Taxonomy on IoT authentication schemes

In this section, we discuss the taxonomy of IoT devices based on their similarities and characteristics [17]. The taxonomy can be categorised in six categories: IoT layers, authentication factor, architecture, procedure, token-based, and hardware-based [17]. Figure 2.1 depicts a general overview of the taxonomy.

Authentication factor refers to the authentication of a device, either in the form of identity or contextual [17]. Identity-based authentication is when one party presents information to authenticate itself. Identity-based authentication schemes can use one or more cryptographic techniques such as hashing, symmetric or asymmetric cryptographic algorithms. Context-based authentication can either be physical or behavioural. Example of physical authentication can be fingerprints or retinal scans. An example of Behavioural-based authentication is analysis of keystroke dynamics.

The architecture categorisation refers to a system's architecture, which can either be distributed or centralised [17]. Centralized architecture is when one party distributes and manages the credentials used for authentication. For a distributed system, the distribution and management of credentials are done between communicating parties. Distributed and centralised architectures can be divided further into flat or hierarchical schemes. Hierarchical means there is a multi-level architecture. Whereas in a flat system, the entities involved are on the same level.

Procedure regards the way the parties involve authenticate each other. There are three options for an authentication procedure, which are one-way, two-way, or three-

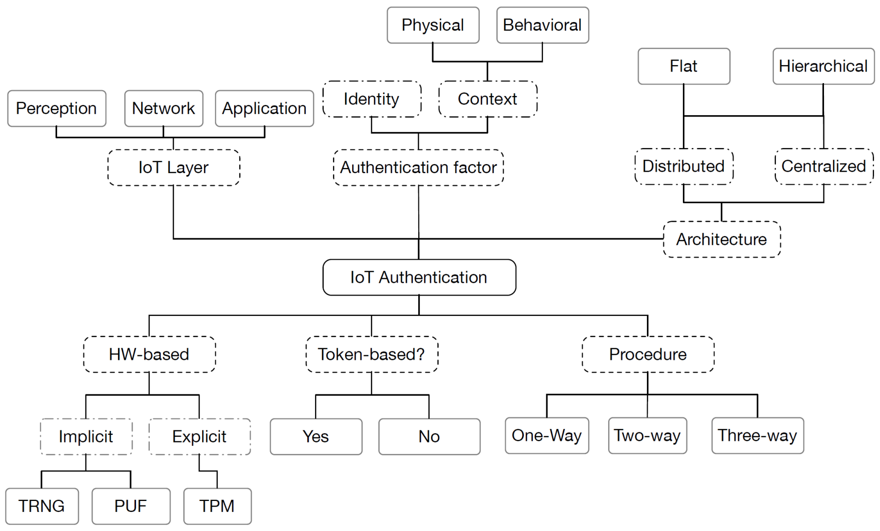


Figure 2.1: Taxonomy of IoT Authentication Schemes [17].

way [17]. The one-way procedure is that only one party authenticates. Mutual authentication or two-way authentication is when both entities authenticate to each other. Three-way authentication is where a central authority authenticates the parties and help to authenticate themselves mutually.

Token-based authentication is when a user or device uses the token for identification [17]. A token is a piece of data created by a server. On the contrary, a non-token based authentication, which can use techniques such as credentials for authentication.

Hardware-based authentication is when the use of physical characteristics of the hardware or the hardware itself is required [17]. The hardware can either be implicit or explicit. Implicit hardware-based authentication uses physical characteristics; an example is a physical unclonable function (PUF) [17]. A PUF uses the differences between physical characteristic to create a digital fingerprint. Explicit hardware-based refers to the hardware itself. An example is a chip that stores and processes the keys used for authentication.

2.3. PRIOR ART OF IOT AUTHENTICATION SCHEMES

In this section, we explain several state-of-the-art IoT authentication schemes. The setting of the schemes can vary with examples such as Generic, Smart Homes, Transportation, Military, Wireless Sensors Networks (WSN), and Smart Grids. The choosen setting influences the requirements for the scheme and IoT devices. For example, a faster process might be a vital requirement for IoT that involves transportation. Given that an error can lead to the loss of human life. However, that might not be the case for a smart grid, which can periodically send energy consumption.

We categorised the discussed schemes according to their architecture. The architecture can either be distributed or centralised. Distributed authentication means the distribution and management of authentication credentials are done among various parties. For a centralised architecture, this task is done by a single identity.

Both centralised and distributed architectures can be further devived into hierarchical and flat schemes. Flat schemes that the entities are on the same level. Hierarchical schemes divide the entities into different levels. After discussing the different schemes, we summarise them in Table 2.2 based on architecture. We also note their corresponding setting.

2.3.1. Flat and Distributed Architecture

In this subsection, we briefly discuss four systems that use the flat and distributed architecture, their setting, and their strengths and weaknesses. The setting for the first scheme is WSN. WSN is a set of networks that combine distributed information collection, information transmission and information processing technology [19]. The authors proposed a scheme that consists of two main steps: the initialisation step and the authentication step [19]. The initialisation step is when the user gets a lightweight public/private key-pair. The authentication step is when two nodes use their key-pair to authenticate each other. The strength of this system is resiliency against DoS and MITM attacks. While the scheme has the weakness that authentication must occur many times in a multi-server environment.

The second work is a two-stage mutual authentication protocol; the stages are the enrollment stage and the authentication stage [20]. The enrollment stage handles the identification of devices. While in the authentication stage, several handshake messages are exchanged between the end device and server to generate a session key. The setting for this protocol is WSN. The scheme's weaknesses are inefficient storage of certificates and the susceptibility to node capture attacks. The strengths are the ability to resist malicious users and DoS attacks.

The third work is an identity and trust framework for IoT devices, based on Distributed Ledger Technology for Self-Sovereign Identity [10]. The setting for this protocol is generic. The framework utilises a Web of Trust to enable automatic and trust rating for arbitrary identities. The strengths are the low cost for creating identities and the ability to white-list a set of identities. However, malicious identities can influence the trust score, and malicious identities can be abandoned for a new one, known as whitewashing.

The last work that we discuss with this architecture is consisted of three tiers and uses each of these protocols for different purposes, these are Diffie-Hellman for key agreement protocol, Rivest–Shamir–Adleman (RSA) public-key cryptosystem and Advanced Encryption Standard (AES) to achieve confidentiality and HMAC for message integrity [21]. The scheme solutions focuses on Smart Grids. The weakness of the system is that the authors lack analysis for location privacy. On the other hand, a strength is that the scheme is resistant to modification, replay, and message analysis attacks.

2.3.2. Hierarchical and Distributed Architecture

In this subsection, we will briefly cover four schemes with this architecture, their scenario, and their strengths and weaknesses. The first authentication scheme is named speaker-to-microphone, which is a lightweight device authentication protocol for IoT

[22]. This scheme is applicable for generic IoT devices and achieves distance authentication between wireless IoT devices. The system's strengths are that it has a low error rate and is resistant to audio replay attacks, changing distance, and similar attacks. However, a weakness is that location privacy is not analysed.

The second work we discuss uses X.509 digital certificates to ensure the authentication of devices [23]. This scheme aims to be generically applicable. The system strength is the use of X.509 certificates. The use of X.509 certificates makes the system easy to integrate, because these certificates are already in use. The system's weakness is the inability to scale to the expected number of IoT devices.

The third work that has distributed and hierarchical architecture setting is for generic IoT devices. The scheme is a multi-tier authentication [24]. In the first stage, a server in the cloud verifies a users credentials. In the second stage, a user would pursue a predetermined sequence on the virtual screen. The system's benefit is the resistance to replay attack. However, the possibility of a DoS attack is not considered.

The last scheme is on a vehicular ad-hoc network (VANET) [25]. In other words, the chosen scenario for this system is transportation. The authors propose a threshold authentication protocol to support secure and privacy-preserving communications in VANETs. The protocol uses a group signature scheme for achieving threshold authentication, anonymity, efficient cancellation and traceability within the VANET. A limitation of this system is the absence of analysis of the communication overhead. Simultaneously, the benefits are efficient computation cost and the resistance against a replay attack.

2.3.3. FLAT AND CENTRALISED ARCHITECTURE

In this subsection, we discussed five different Flat and Centralised architecture schemes, their benefits and drawbacks and their settings. The first protocol is for performing authentication between a user and a server, and not between end devices [26]. The protocol is a two-step verification of IoT devices. The first authentication step is a password or a shared secret key, whilst the second is the use of a PUF. The benefits are low computation cost, consideration for impersonation and physical attacks. In contrast, the drawbacks are no consideration for machine-learning attacks and no consideration for environmental variations [17].

The second work is a new hardware-based approach, using PUFs as a fingerprint to authenticate IoT device [27]. The setting is generic IoT applications. A disadvantage of the system is that it may not be suitable for a scenario where the environmental conditions fluctuate because of the usage of PUFs [17]. The main advantage of this work is that it takes machine learning attacks into account.

The third work is a vehicular authentication scheme. The scheme connects wirelessly and physically to Electronic Vehicle (EV) [28]. Wireless in the form of wireless connecting to a server and physical by a charging cable. Both connections verify the identity of the EV. One benefit is that resilient against substitution attacks because of both connection type. However, there is no evaluation of the message and the verification delay.

The fourth work is design for smart grids. The authors discuss each home equipped with a smart meter that sends data in intervals to a hub. This hub collects and sends the data to a control centre. The control centre prepares a bill and sends it to the customer. The communication between hub and smart-meter is when the scheme comes into play.

Which is a lightweight authentication scheme with efficient communication and computation overhead [29]. The efficiency of communication and computation is a benefit for such systems and the resiliency against message modification attacks, message analysis, and message injection. However, a drawback is the lack of consideration for routing attacks.

The last work that we discuss concerning this architecture is also geared for smart grids. The authors proposed an efficient and robust approach to authenticate aggregate power usage data. The architecture is also fault-tolerant and is based on signatures [30]. The benefits of this scheme are fault-tolerant architecture and efficiency for communication and computation. Whilst having the drawback that attacks against such a system are not considered.

2.3.4. Hierarchical and Centralised Architecture

The last discussed authentication architecture is hierarchical and distributed. We highlighted three works that utilise this architecture. The first scheme is a PUF-based authentication scheme for smart homes. The work provides mutual authentication between end devices and gateway by storing the Challenge-Response pairs from the PUF [31]. The authenticate devices then can generate a session key for secure communication. This scheme's benefits are the resistance to replay attacks and efficiency in terms of computation and communication. The drawback of the scheme is the lack of consideration of machine-learning attacks and environmental variations.

The second work utilises identity-based aggregate signatures to authenticate in VANETs [32]. The benefits of this system resistance to movement tracking, replay, and message modification attacks. However, location privacy is not considered.

The third work is about a PUF-based authentication scheme that uses zero-knowledge proof of knowledge of discrete algorithm [33]. The protocol requires for each authentication that the user inputs a password. The limitations of the system an attacker could use a fake user to ask for the password. Only devices are authenticated. At the same time, one strength is resistance to the cloning and copying of devices.

In Table 2.2, we summarised the discussed IoT authentication schemes. For each scheme, we indicate which architecture they uses via a checkmark. We also denote setting for the corresponding scheme. Additionally, Table 2.2 also includes the architecture of a closely related work, named Bubbles of Trust. Bubbles of Trust is elaborated in detail in the following section.

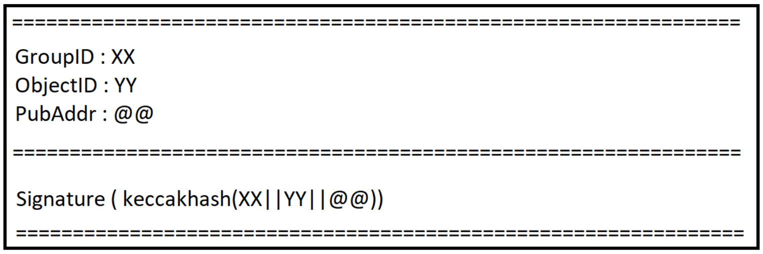
2.4. Bubbles of Trust

In this section, we covered the Bubbles of Trust protocol in-depth. Bubbles of Trust is a decentralised Ethereum-based authentication system for IoT devices [9]. The architecture of Bubbles of Trust solution is hierarchical and centralised, focused for a generic setting. The scheme uses the term Bubble to indicate a group. The goal of the protocol is to form an impenetrable barrier for devices that belong together, referred to as a Bubble. Each Bubble consists of two types of devices Master Device (MD) and Follower. Each Bubble can consist of multiple Followers, but one MD. The task of an MD is to add Followers to the Bubble.

Table 2.2: A summary of the discussed papers catergorised on their architecture and their setting

	Flat		Hierarchichal		Setting
	Centralised	Decentralised	Centralised	Decentralised	
[19]	-	✓	-	-	WSN
[20]	-	✓	-	-	WSN
[10]	-	✓	-	-	Generic
[21]	-	✓	-	-	Smart Grids
[26]	✓	-	-	-	Generic
[27]	✓	-	-	-	Generic
[28]	✓	-	-	-	Transportation
[29]	✓	-	-	-	Smart Grids
[30]	✓	-	-	-	Smart Grids
[22]	-	-	-	✓	Generic
[23]	-	-	-	✓	Generic
[24]	-	-	-	✓	Generic
[25]	-	-	-	✓	Transportation
[31]	-	-	✓	-	Smart Home
[32]	-	-	✓	-	Transportation
[33]	-	-	✓	-	Generic
[9]	-	-	✓	-	Generic

The MD adds Followers to a Bubble by creating an identity for them, called a ticket. Later, when a Follower creates an association request to another device, its ticket values are stored on the Blockchain. An identity consists of a group ID, the device's public key and identifier. The group ID is unique for every Bubble, and the device's identifier is unique within the Bubble. These values are hashed and then signed by the corresponding MD. Devices generate their Public/private key pair using Elliptic Curve Cryptography (ECC) and MDs signs using Elliptic Curve Digital Signing Algorithm (ECDSA), and uses Keccack for hashing [9]. The protocol uses ECC and ECDSA due to their smaller key and signature size. Figure 2.2 and identity with placeholder values.



```
==============================================================

GroupID : XX
ObjectID : YY
PubAddr : @@

==============================================================

Signature ( keccakhash(XX||YY||@@))
==============================================================
```

Figure 2.2: The identity structure of a Follower [9].

An association request is the first transaction of a Follower, where it submits its ticket to the Ethereum blockchain. If successful, the ticket is no longer needed to authenticate itself. The whole procedure is depicted in step in Figure 2.3. These steps are :

1. The Follower signs its ticket and sends it with a transaction.
2. The smart contract verifies that the ticket is signed by sending Follower and corresponding MD.
3. If that is the case, these values are stored.
4. Subsequent transactions will contain data, Follower and MD identifiers, signed by the Follower.
5. When the transaction is received, its signature integrity is verified.
6. If that is the case, the smart contract retrieves the public key attached to the corresponding identifiers.
7. The stored values with the sent values are compared.
8. To indicate if the device is successfully authenticated.

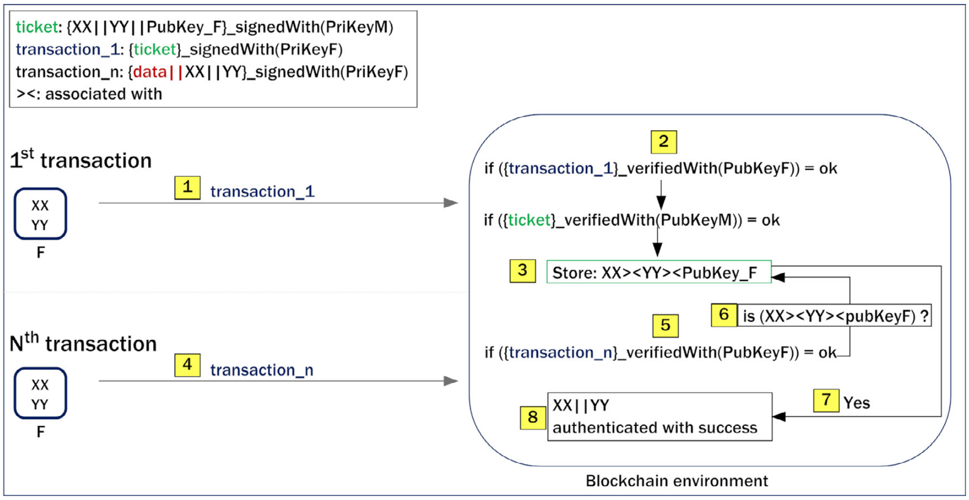

ticket: {XX||YY||PubKey_F}_signedWith(PriKeyM)
transaction_1: {ticket}_signedWith(PriKeyF)
transaction_n: {data||XX||YY}_signedWith(PriKeyF)
><: associated with
1st transaction
XX
YY
F
1 transaction_1
Nth transaction
4 transaction_n
2 if ({transaction_1}_verifiedWith(PubKeyF)) = ok
if ({ticket}_verifiedWith(PubKeyM)) = ok
3 Store: XX><YY><PubKey_F
6 is (XX><YY><pubKeyF) ?
5 if ({transaction_n}_verifiedWith(PubKeyF)) = ok
7 Yes
8 XX||YY authenticated with success
Blockchain environment

Figure 2.3: Transaction procedures between devices [9].

The notable characteristics of the Bubble of Trust system are the decentralised nature, scalable with regards to creating tickets, the virtual zones between devices, generically applicable, and MD only needs to be present for the addition of new Followers. Additionally, the system is robust against DDoS attacks due to the decentralised architecture of Ethereum Blockchain. The system is resilient against Sybil attacks because the public key of MD signs the tickets. Therefore, valid tickets are secure under the security of the signature scheme. The system also protects against Message replay attacks due to timestamps and signing of a transaction.

On the other hand, the system's shortcomings are the inability to remove identities and communicate with other Bubbles. Additionally, the lifespan of identities is endless.

If an adversary gains access to a device secret, identity will remain valid. Even if the communication via that ticket is disregarded, the public nature of the Ethereum blockchain means an adversary could read previous communication.

The use of transaction to send messages makes the system speed dependant on the transaction speed of the Ethereum Blockchain. Additionally, the number of IoT devices expected to grow begs the question of the scalability claims regarding communication. Also, to execute a transaction, devices need to have an Ethereum wallet with Ether. Therefore, devices can become susceptible to attempts to steal their Ether. Also, there is no explanation on how communication is instantiated if all communications go via the Blockchain.

In this chapter we have discussed the state-of-the-art with regards to IoT authentication. The information gathered from this chapter is used for our system. In the next chapter, we present the preliminairy information for our system.

3
Preliminaries

In this chapter, we discuss the preliminary knowledge and techniques used in this research. First, we discuss the cryptographic primitives in Section 3.1, then we introduce blockchain technology in Section 3.2.

3.1. Cryptographic Primitives

In this section, we describe the cryptographic primitives and their specification for this thesis. We elaborate on the cryptographic encryption schemes, cryptographic hash functions, digital signatures, and Merkle Trees.

3.1.1. Cryptographic Schemes

For this thesis, we briefly elaborate on symmetric and asymmetric encryption schemes. Followed by discussing a key agreement protocol that incorporates both types of encryption scheme.

In symmetric encryption, two parties agree on a shared key K. Key K is the secret key for the chosen scheme. The corresponding key K can take part in two functions: the encryption function E_K and the decryption function D_K. The encryption function encrypts plaintext messages into a ciphertext. The decryption function deciphers ciphertext back into plaintext.

The NIST standard for symmetric encryption is AES [34]. However, due to the limitations of IoT devices, alternative schemes can be used. Examples of such schemes are KLEIN [35] and Tiny Encryption Algorithm (TEA) [36].

Asymmetric cryptographic schemes are alternatively named public-key encryption schemes. In these schemes, each party has a key pair. A key pair consists of two connected keys, a private and a public key. A key pair is denoted as (pk, sk), with pk representing a public and sk as private key. As the name already indicates, the private key should be kept private, and the public key can be shared. A pk is used to encrypt a message m. Afterwards, the corresponding sk can decrypt to retrieve message m.

For our research, we utilises a key agreement protocol. The chosen protocol is Elliptic Cuve Diffie Hellman (ECDH), which combines a symmetric and asymmetric scheme. The asymmetric side is based on Elliptic Curve Cryptography (ECC), whilst the choice of symmetric scheme is arbitrary.

Elliptic Curve Cryptography (ECC)
In 1985, ECC was introduced as an alternative to other public-key cryptosystems [37]. ECC is based on an algebraic structure on elliptic curves over finite fields. The system's security is based on a hard problem of solving the discrete logarithm problem [38]. An elliptic curve can be defined as follow.

Definition 3.1.1 (Elliptic Curve)**.** An Elliptic Curve is the set of point describe in the following equation :

$$\{(x, y) \in Z_p \mid y^2 = x^3 + ax + b,\ 4a^3 + 27b^2 \neq 0\} \cup \{0\}. \tag{3.1}$$

The requirement of $4a^3 + 27b^2 \neq 0$ is so curves do not have cusps, self-intersections, or isolated points. A cusp is point where the curve moves in the opposite direction. A set of values compromise the domain parameter for ECC. These parameters are denoted as $D = (p, a, b, G, n, h)$. The domain parameters consist of p is an integer that specifies the field of the curve F_p. $a, b \in F_p$ are constants to define the equation for elliptic curves. G is a base point (x, y) on a corresponding curve. n is a prime number; that is, the order of G. h is an integer that defines the cofactor on the curve. The domain parameters are publicly available to all participants. Now that we define Elliptic Curve, we continue to discuss point operations. We then briefly discuss the generation of key-pairs, followed by the Elliptic Curve Diffie Hellman (ECDH).

Point operations
The standard operations for elliptic curve are point addition and point doubling. Point addition is when two points, P and Q are added to form a new point. $R = P + Q = (x_p, y_p)$ + $(x_q, y_q) = (x_r, y_r)$. There is one requirement for point addition, that is $P \neq Q$. The addition is shown in the following equation :

$$\begin{aligned} \lambda &= \frac{y_q - y_p}{x_q - x_p}, \\ x_r &= \lambda^2 - x_p - x_q, \\ y_r &= \lambda(x_p - x_r) - y_p. \end{aligned} \tag{3.2}$$

For point doubling, the calculation of x_r and y_r stay equal, except for the first calculation, which is :

$$\lambda = \frac{3x_p^2 + a}{2y_p}. \tag{3.3}$$

Key-pair generation
A party can create a key-pair by choosing a random value $d \in_R [1, n-1]$. The chosen d

is the private key. From d, the public key is determined by computing dG. Q symbolises the result of dG. Therefore the key-pair is (Q, d).

Elliptic Curve Diffie Hellman (ECDH)
ECDH is a key agreement protocol that uses asymmetric cryptography to determine a session for a symmetric encryption. It should be noted that both parties operate with the same domain parameters. Algorithm 1, consisting of two function; **Generate Public Key** and **Create Session Key**. **Generate Public Key** generates a Public from the domain parameter G multiplied with the random value s between 1 and $n-1$ to produce public key p.

Create Session Key takes a public key as a parameter and multiplies it with its secret value. The result of the multiplication is a symmetric key denoted as sk. Figure 3.1 depicts an ECDH handshake between two parties. sk can get inserted in a key derivation function (KFD), which produces a session key. A KDF is a function that derives one or more secret key from a secret value. The chosen KDF should meet the needs of chosen symmetric encryption.

Algorithm 1 ECDH

```
1: procedure GENERATE PUBLIC KEY
2:     $s \in [1, n-1], p = sG$
3:     return $(p)$
4: end procedure
5: procedure CREATE SESSION KEY($q$)
6:     $r = q \times s$
7:     $sk = KFF(r)$
8:     return $(sk)$
9: end procedure
```

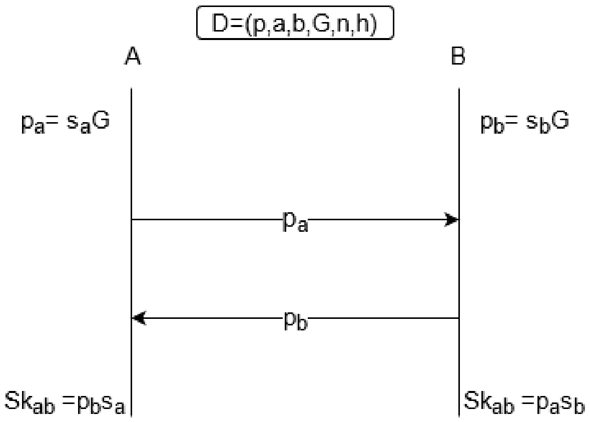


Figure 3.1: ECDH interaction graph between two parties.

3.1.2. CRYPTOGRAPHIC HASH FUNCTIONS

A cryptographic hash function maps an input of arbitrary length to output a short fixed-length string. A cryptographic hash function possesses three properties: pre-image resistance (1), second pre-image resistance (2) and collision resistance [39]. Pre-image resistance means that it should be computationally difficult to find the inputted message from a hashed value. Second pre-image resistance means that given a message and resulting hash value, it should be computationally infeasible to find a different message that would result in the same hash. Collision resistance means it is computationally infeasible to find two different messages that produce the same hash value by a hash function.

In this research, we use SHA-3 as the hash function. The standard hash function by NIST is SHA-3 [40]. Cryptographic hash functions are useful for cryptographic primitives such as Merkle Trees and Digital Signatures.

3.1.3. DIGITAL SIGNATURES

Digital signatures are schemes that prove the authenticity of data. Digital signatures provide three properties: authentication, non-repudiation and integrity. Authentication is to identify an entity correctly. Integrity refers to the ability to protect data from alterations. Non-repudiation refers to the incapability to deny an action.

Asymmetric cryptographic schemes and cryptographic hash functions are used to achieve these properties. A message m is hashed to produce hash h. The signer then signs h with its private key to create a digital signature s. The verification process takes the same message m, hashes it and then signs it using the party public key, and the result should be equal to s.

We choose the Elliptic Curve Digital Signing Algorithm (ECDSA) because of the smaller signature sizes. The pseudocode for ECDSA is display at Algorithm 2 consisting of two functions **Generation** and **Verification**.

The **Generation** function generates the signature by taking the parameters D, m, sk_a. D is the domain parameters of the curve, m is the message, and sk_a is the private key of Party a. The message is hashed, by hash function H that produces the digest z. k is then generated from random value in Z_p space. (x_1, y_1) is the coordinates resulting from the multiplication of k and the base point G. The signature is the variables r and s. r is the x_1 coordinate in modular n. While, s is the result of inverse of k multiplies with the addion of z and $r \times sk_a$.

Verification is the function that verifies if a specific key has signed a signature. The parameters for the function is domain parameters D, the message m, the public key pk_a and signature r, s. The message is hash with the same hash function in **Generation**. The inverse of s is computed into variable w. w is multiplied with z and r, resulting in u_1 and u_2 respectively. u_1 is then multiplied with G and then added with the result of $U_2 pk_a$. The resulting point should be equal to the point generated by kG. The x coordinates compared to r.

ECDSA also has the capabilities to be unforgeable under the right set of conditions [41]. These conditions are :

- Collision-resistance for the chosen Hash function, as well a uniformity property.

- Pseudorandomness in the key space for the private key generator.
- Ephemeral mapping from the public key into the private key space.
- Generic treatment of the underlying group.

Algorithm 2 ECDSA

```
1: procedure GENERATION($D, m, sk_a$)
2:     $z \in H(m)$
3:     $k \in Z_p$
4:     $(x_1, y_1) = kG$
5:     $r = x_1 (mod\ n)$
6:     $s = k^{-1}(z + r \times sk_a)(mod\ n)$
7:     return $r, s$
8: end procedure
9: procedure VERIFICATON($D, m, pk_a, (r, s)$)
10:    $z \in H(m)$
11:    $w = s^{-1}(mod\ n)$
12:    $u_1 = zw(mod\ n)$
13:    $u_2 = rw(mod\ n)$
14:    $(x_1, y_1) = u_1 G + U_2 pk_a$
15:    return $r == x_1(mod\ n)$
16: end procedure
```

3.1.4. MERKLE TREE

Merkle tree is a data structure in the form of a binary tree, where the leaf nodes are associated via a cryptographic hash [42]. The data structure is bandwidth-efficient and provides secure verification of elements within the tree. Figure 3.2 depict an example of a Merkle tree. The original elements are referred to as leaves. These leaves are hashed, which are combined by hashing till one value. This hash value is named the root. Given the properties of the cryptographic hash, the inclusion of an element can be verified. In this thesis, we refer to this proof as proof of inclusion. Figure 3.2 denotes the proof of inclusion for the value T_3 in yellow. Hashing T_3 and combining with the hashes within the proof of inclusion would result in root value.

3.2. BLOCKCHAIN

There are numerous definitions for a blockchain technology [43]. For this thesis, we have chosen the definition of a system that performs accurate and irreversible data transfer in a decentralised Peer-to-Peer (P2P) network [44]. As the name suggests, blockchain is a chain of blocks. We first elaborate on the blockchain basics, continued with the classification of blockchain technology. We finalised this section by discussing the Ethereum blockchain.

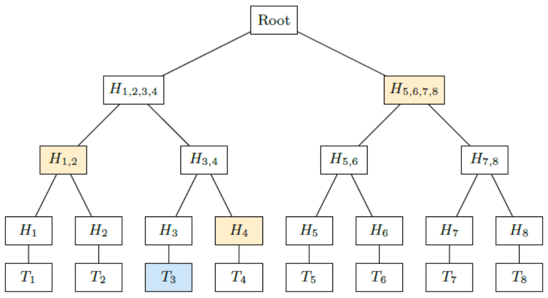


Figure 3.2: Merkle Tree representation with Merkle Proof coloured in yellow for data T_3 [42].

3.2.1. BLOKCHAIN BASICS

We discuss the essential basic of blockchain technology based on the paper by Nakamoto [45].

Transactions and addresses

Transactions are the transfer of value between parties recorded on the blockchain. A transaction consists of five properties: a sender, a receiver, a transaction message, a recipient, and an exchange of ownership. Digital signatures sign transactions via the sender's private key. Blockchain technology stores these signed transactions in blocks.

Blocks

Blocks are container data structures where transactions are aggregated and timestamped. Merkle trees reduce the needed space while maintaining membership of transactions. In blockchain technology, blocks can be connected to the previous one. The connection of blocks can be made by hashing a block and using it as a base for the next block. Due to the linkage, small changes in one block propagate to every succeeding blocks. The order of blocks is determined by a consensus mechanism.

Consensus

Each blockchain technology utilises a consensus algorithm to achieve agreement between the parties involved. There exist various consensus algorithms such as, but not limited to, Proof-of-Work (PoW), Proof-of-Stake (PoS), and Proof of Elapsed Time. For this thesis, we discuss the consensus mechanism relevant to Ethereum, which are PoW and PoS.

Proof-Of-Work

PoW reaches consensus by posing task which parties must try to solve. The parties that try to solve this task are called miners. In Ethereum, creating a new block is to find a value that, together with new transactions and previous blocks, results in a value posed by the consensus algorithm [46]. The difficulty is adjusting accordingly to the number of miners. A benefit for PoW is that rewriting a single transaction is difficult because a small change will propagate to every block that follows. Ethereum has the maximum capabilities to handle twenty-seven transaction per second [47].

Proof Of Stake
Ethereum is moving from PoW to PoS [48]. For Ethereum PoS, parties need to stake an amount of ether as a collateral for creating blocks. The amount for Ethereum is 32 ether [48]. The party that verifies the new block will be chosen at random from the stake owner to execute the task. Compared to PoW, PoS is more energy-efficient, lower hardware barrier to join, and more time-efficient. However, PoS has not been rigorously tested in practice. Validators with a large amount of ether can have an outsized influence on the network.

Ethereum 2.0 is the named used for when Ethereum swithces to PoS [49]. However, Ethereum 2.0 is not only limited switching consensus mechanism, it potentionally could also include sharding [49]. Sharding, with regards to Ethereum is to split the current chains of block into multiple chains [50]. The switch toEthereum 2.0 with sharding According to Vitalik Buterin, the founder of Ethereum, Ethereum could scale to hundred thousand transactions per second [49].

3.2.2. TWO-DIMENSIONAL CLASSIFICATION

For this thesis, we chose to follow the categorisation proposed by Peters and Panayi [51], which differentiates blockchain according to two dimensions:

- **Public or Private**: In a public blockchain, anyone can read the state and submit transactions. In contract, in a private blockchain, these right are restricted to a set of nodes.
- **Permissionless or Permissioned**: In a permissioned blockchain, a central authority or a set of central authorities can verify transactions. While in a permissionless blockchain, any node can take part in the verification process.

From this categorisation, there are four possibilities visualised in Table 3.1.

Access to transactions	Ablility to validate transactions	
	Permissionless	Permissioned
Public	All nodes can read, validate current transactions, as well submit new transactions	The validation of transaction is done by selected group of nodes, however the addition of new transation and reading of transaction can be done by all nodes.
Private	The validation can be done by all parties, but only a restricted set of nodes can submit or read transactions	The submit of new transaction, read and validation of current of transactions can only be done by certain set of nodes.

Table 3.1: Categorisation of blockchain technology.

3.2.3. ETHEREUM

Ethereum is a protocol for executing a virtual computer in an open and distributed manner by reaching consensus among nodes. The virtual computer's name is called Ethereum Virtual Machine (EVM), and its programs are called smart contracts. According to Vitalik Buterin, smart contracts act as autonomous agents that reside on the blockchain and execute a specific piece of code triggered by a message or transaction [52]. Smart contracts have direct control over their ether balance and persistent key/value store. We discuss the essential concepts of Ethereum, taken from the white paper [52].

Ethereum Accounts

Ethereum consists of two types of accounts. All Ethereum accounts have thee attributes: a nonce, a balance in ether, and internal storage.

- **Wallets or externally owned account**: Wallets have three state attributes and act as bank accounts. The owner of the account has control of the private key linked to the account. Owners can use their wallets to execute transactions.
- **Smart contracts or contract accounts**: Smart contracts have an additional attribute called the contract code. This code controls these accounts. If a contract account receiver a message, trigger the code. The contract code can perform operations such as interacting with internal storage, performing certain computations, or sending messages to other contracts.

Ether and Gas

Ethereum also has two types of tokens; these are ether and gas. Ether is the currency within Ethereum. Like Bitcoin, ether is minted by miners and traded. At the moment of writing, the miners earn ether as a result of PoW. Gas act more as a commodity, like gasoline. Gas is a unit measuring the computational work of running transactions or smart contracts in the Ethereum network. Examples of computational work are hashing, addition, and multiplication. Gas is the fuel for transactions in Ethereum.

Transactions and Messages

Just as gasoline, gas price fluctuate based on supply and demand, upon execution of a transaction, the needed gas is calculated and converted on the ether and deducted from the account. The execute of transactions is dependant on the type of account executing transactions. Wallets can initiate transactions to other wallets and smart contracts. Smart contracts are activated by a transaction. However, contract accounts only communicate in response to received transactions.

Events

Smart Contracts in Ethereum can emit events. These events are stored in the blockchain with the corresponding transaction. Events behave as a log for the transaction. An event can be named, and the logged data can be structured. The data emitted from an event is not stored in the internal storage of the smart contract. Therefore, a smart contract cannot retrieve the previously emitted data.

Nodes

Nodes is the name for the computers that verify blocks and transactions. Ethereum has three types of nodes, these are : light full, and archive node [53]. Light nodes store the small subset of the recent blocks of the blockchain. Full nodes store the entire blockchain. Archive nodes, store the entire blockchain as well as the intermediary state of every account and contract from the beginning of the blockchain [54]. The preliminary information discussed in this chapter, is used for our system in the succeeding chapter.

4

Bubblechain

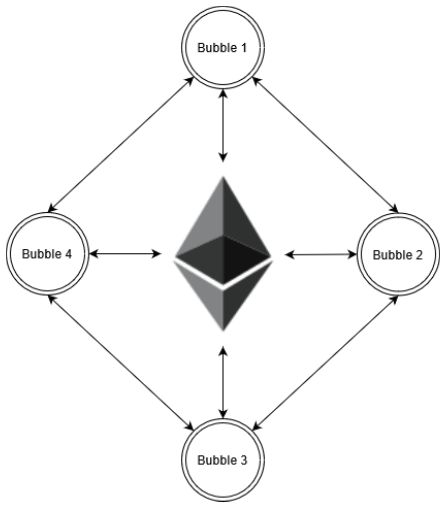
In this chapter, we propose Bubblechain, an IoT authentication system that meets the previously mentioned requirements of scalable, decentralised, incorporates the lifecycle of devices, generic, capable of M2M communication, and has a trust barrier. First, we present the model and assumptions for our system, followed by the workings of the Bubblechain system.

4.1. Our Model

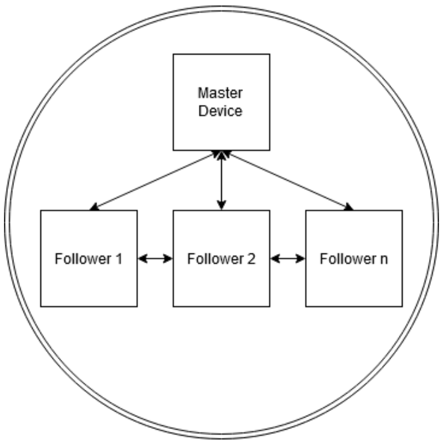
In Chapter 2, we discussed a taxonomy for IoT authentication mechanisms. In this subsection, we first discuss how our system fits according to the taxonomy. We follow by discussing the two types of devices: Master Device (MD) and Follower. In Figure 4.1a, we depict a high-level overview of the Bubblechain system. The figure shows various Bubbles that interact with each other and the Ethereum blockchain, depicted in the middle by its logo. The interaction with the Ethereum blockchain occurs via a smart contract. A Bubble is a group of devices that belong to the same owner. Figure 4.1b zooms into one of the Bubbles from Figure 4.1a. Within a Bubble, we can find the two types of devices. After covering the taxonomy of Bubblechain, we continue by elaborate on the assumptions for these devices.

4.1.1. Taxonomy of Bubblechain

The taxonomy discussed in Section 2.2 consists of six categories: IoT layers, authentication factor, procedure, token-based, hardware-based, and architecture [17]. As for IoT layers, our system uses all three layers. For the authentication factor, we choose an identity-based approach. In Subsection 4.2.3, we elaborate on the structure of the identities within our system. For the authentication procedure, Bubblechain uses a two-way and three-way procedure. For devices that belong to the same Bubble, our system uses a two-way procedure. For devices from different Bubbles, our system uses a three-way procedure. Bubblechain uses a token-based approach by having one entity create a piece of data to authenticate. How the token is used is discussed in Subsection 4.2.4.



(a) High-Level overview of the Bubblechain system.



(b) Overview of Bubble.

Figure 4.1: High level overview of Bubblechain system and zoomed in view on a Bubble.

Our solution explicitly uses hardware in the form of storage and the creation of keys and identities. For the architecture, Bubblechain uses a centralised and hierarchical architecture. A centralised architecture means that one entity manages the identities. In our system, the MDs manage the identities.

4.1.2. Master Device (MD)

Figure 4.1b depicts a Bubble with one MD. The reason is that each Bubble is limited to only one MD. Our system assumes that the MDs follow an Honest-But-Curious (HBC) adversarial model. HBC means that the party follows the protocol but is interested in breaking the privacy of the following participants [55]. If the MD does not follow this model, then identities cannot be managed. Subsection 4.2.4 elaborates on managing identities and the role of an MD.

For an MD to be able to carry out the task of managing identities, there are requirements. These requirements are :

- Possession of an Ethereum wallet containing Ether and storing it securely. The MD would have to be able to execute transactions on the Ethereum blockchain.
- Storage capabilities for the created devices. The MD has to be able to store all valid identities.
- Capable of creating and maintain a Merkle Tree. Updating the Merkle Tree from time to time and create proof of inclusions for the values.
- Connection to all devices within the same while managing identities.

Naturally, with the various types of IoT devices, some devices are better suited for the task of an MD. The devices should be able to maintain an Ethereum wallet securely.

One option would be a desktop or a laptop. Alternatively, specialised equipment can be created for the task with options such as a Raspberry Pi.

4.1.3. Follower

The remaining devices within a Bubble take the role of Followers. Each Bubble can contain multiple Followers. Each Follower must have an ECDH key pair, either generated by itself or preloaded. A Follower needs a limited amount of storage to store its identity and proof of inclusion. A Follower should be capable of verifying a signature, validate a proof of inclusion and establish a session key via ECDH. Followers also have to be online when MDs are managing identities. The reasoning for the requirements is elaborated in the following section. Examples of devices that could be a Follower are smart cars, smart speakers, and smart lighting systems.

4.2. Our System

In this section, we present the working of our system. First, we cover the initialisation steps for our system. Second, we cover the smart contract that we created. Third, we elaborate on the structure and creation of identities within our system. Fourth, we cover the management of these identities. Fifth, we cover the authentication procedures. Last, we briefly discuss how to establish communication between devices.

4.2.1. Initialisation

As previously stated, our system interacts with the Ethereum blockchain via a smart contract. For initialisation, the smart contract needs to be deployed on the blockchain. The deployment of the contract address makes the contract address and application binary interface (ABI) accessible. An ABI is a lower-level implementation of the smart contract. The MDs and Followers need to have the contract address and ABI to interact with the smart contract.

Additionally, a Follower would need to have a connection to a full or archived Ethereum node. Followers also need to have their ECDH key-pair generated or preloaded. MDs need to have an Ethereum wallet with Ether and the capability to execute transactions.

4.2.2. Smart Contract

In this subsection, we present the functionalities of our smart contract. In Algorithm 3, we depict the functions of the smart contract. The smart contract consist of three functions: **emitRootValue**, **storeRootLocation**, and **getBlockNumber**. In the succeeding subsection, we elaborate on how these functions fit into the Bubblechain system.

The **emitRootValue** function takes three parameters. An Ethereum address noted as $address$, a root value from Merkle Tree as $root$, and an expiration date as exp_date. The function has the requirement that $address$ is equivalent to the address of the entity executing the transaction. We denote the address of the entity executing the transaction as $sender$. The function then emits an Ethereum event name RootEvent, an event containing the parameters and $address$. The **emitRootValue** function, alters the state of the smart contract.

The **storeRootLocation** function takes an Ethereum address as $address$ and block number as *block* as parameters. The **storeRootLocation** function has the requirement that the address should be the same as the address of the entity executing the transaction. The *sender* variable is the Ethereum address of the entity executing the transaction. The function then maps *address* to *block* within a mapping named *root_locations*. Executing of the **storeRootLocation** also alters the state of the smart contract.

The last function within the smart contract is the **getBlockNumber** function. The function takes an Ethereum address, noted as *address* as a parameter. For this function, the *address* can be any Ethereum address. The functions returns value stored at location *address* within the mapping *root_locations*. If there is no value, then zero is returned. This function does not alter the state on the blockchain but only views it.

Algorithm 3 Bubblechain Smart Contract

```
1: procedure EMITROOTVALUE($address, root\_value, exp\_date$)
2:     require( $address = sender$)
3:     $emit\ RootEvent(address, root\_value, exp\_date)$
4: end procedure
5: procedure STOREROOTLOCATION($address, block$)
6:     require( $address = sender$)
7:     $root\_locations[address] = block$
8: end procedure
9: procedure GETBLOCKNUMBER($address$)
10:    return $root\_locations[address]$
11: end procedure
```

4.2.3. IDENTITY OF THING (IDOT)

In this subsection, we present the structure of these identities Within our system. We call the identities of Followers IDoT, which stand for IDentity of Thing.

Figure 4.2 depicts a template for an IDoT. The figure shows the IDoT consisting of five attributes: Bubble-ID, Device-ID, Device-Key, Expiration-Date and Signature. Bubble-ID is an Ethereum address. Device-ID is a unique identifier within a Bubble. Device-Key is the Follower's public key from its ECDH key pair. Expiration-Date is the previously determined date that an IDoT will expire. Last is Signature, a signature created from the hash of the four previous attributes by the Ethereum wallet with the corresponding address set as the Bubble-ID.

Algorithm 4 shows the procedure to create an IDoT. The function **createIDoT** takes a public key, noted as *pub* and expiration date as *exp_date* as parameters. The function first takes the address from its Ethereum wallet and assigns it to the variable *addr*. Then a unique random identifier for the Bubble is created and stored in the variable *device_ID*. **createIDoT** proceeds to takes the parameters, *device_ID*, and *addr* and hashes with SHA-3 hash function. The result is signed by the Ethereum wallet connected to *addr*, and the signature is stored in the variable *sig*. Therefore, only MDs can execute **createIDoT** because it requires the capability to sign a message with the Ethereum wallet. The combination of *addr*, *device_ID*, *pub*, *exp_date*, and *sig* forms an IDoT,

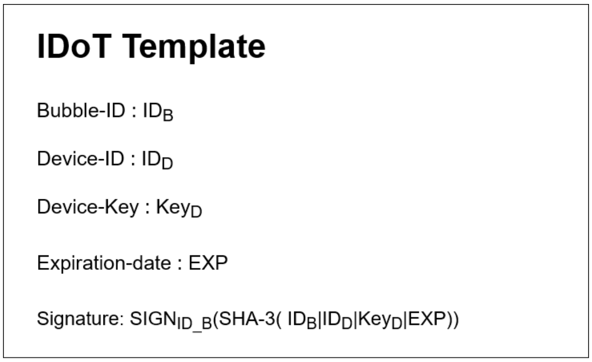


Figure 4.2: Template of an IDoT.

which is shown by the variable $IDoT$. $IDoT$ is then added to mapping based on their Device-ID named $IDoTs$. $IDoTs$ is stored locally by the MDs and is a global variable used for managing identities.

Algorithm 4 Procedure to create IDoT

```
1: procedure CREATEIDOT($pub$, $exp\_date$)
2:     $addr$ = Wallet.address
3:     $device\_ID$ = random(0,N)                    ▷ value is unique in the Bubble
4:     $sig = SIGN_{addr}$(SHA-3( $addr$, $device\_ID$, $pub$, $exp\_date$) )
5:     $IDoT$ = [$addr$, $device\_ID$, $pub$, $exp\_date$, $sig$ ]
6:     $IDoTs[device\_ID] = IDoT$
7:     return $IDoT$
8: end procedure
```

4.2.4. IDENTITY MANAGEMENT

In this subsection, we discuss the management of IDoTs. We refer to the addition, removal and updating of IDoTs as the management of identities. The task of managing identities is done by MDs and needs the **createIDoT** function, as well as the functions in Algorithm 5.

Algorithm 5 consists of one functions named **UpdateRootValue**. The **UpdateRootValue** function is used for addition, removal and updating of IDoTs. We first cover the workings of function, then we elaborate on how it is applied to manage identities.

The function **UpdateRootValue** takes an expiration date as parameter, denoted as exp_date. The function starts by using the function makeTree, which creates a Merkle Tree from its input. In **UpdateRootValue** the input is the global variable $IDoTs$, the result is stored in MT, which is also a global variable. **UpdateRootValue** proceeds to retrieve the root value from MT and store it in $root$. The MD get its address from its wallet and assigns it to $addr$. Naturally, the wallet is the same wallet that signed for identities in $IDoTs$. Then the function executes the **emitRootValue** function for our smart contract with $addr$, $root$, and exp_date as parameters. The result of the transaction is a receipt, which is stored in variable of the same name. The block number of the receipt is set in $block$. The **UpdateRootValue** function then triggers an **storeRootLocation** with $addr$ and $block$ as parameters.

Algorithm 5 IDoT Management

```
1: procedure UPDATEROOTVALUE($exp\_date$)
2:     $MT$ = makeTree($IDoTs$)
3:     $root = MT$.root
4:     $addr$ = Wallet.address
5:     $receipt$ = emitRootValue($addr$,$root$, $exp\_date$)        ▷ Ethereum transaction
6:     $block = receipt$.blockNumber
7:     storeRootLocation($addr$, $block$)
8: end procedure
```

Adding a Follower

By utilising the functions in Algorithm 5 and **createIDoT** MDs can add devices as Followers to their Bubble. Figure 4.3 depicts the interaction graph to add a Follower. The process starts with a soon to be Follower sending its public key to the MD. The MD then creates a IDoT using the **createIDoT** function. The MD then proceeds to use the **UpdateRootValue** function, which interacts twice with the Ethereum blockchain. The dotted line in the figure shows till which point the function is in progress. The Follower then receives the newly created IDoT from the MD. Afterwards, every Follower within the Bubble receives their new proof of inclusion. Each Follower authenticates itself with the Ethereum blockchain and stores the root value and expiration date. Followers can also be added in bulk by creating them before triggering the **UpdateRootValue** function.

Removing a Follower

Within our system, MDs also have the power to remove a Follower from their Bubble. MD's remove Followers by removing the Follower's IDoT from $IDoTs$. Afterwards, the MD executes **UpdateRootValue**, which emits an Ethereum event containing the latest root value and update the location of the root value on the mapping. The MD must also update the proof of inclusions of every Follower with a valid IDoT within $IDoTs$. Every Follower then proceeds to authenticate themselves by connecting to an Ethereum Node and storing the root value and expiration date. Figure 4.4 depicts the interaction graph on how to remove Followers. The removal of devices can also be done in bulk.

Update a Follower

Updating Followers is a combination of adding and removing. The newly created IDoT takes the place of an old. The MD then executes the **UpdateRootValue** function to update the root value, expiration and their location in mapping on the contract. For updating Followers, every valid Follower needs to have their proof of inclusion updated and authenticate themselves. The Follower then stores the root value and expiration date. Each of the operations to manage the number of Followers finalises with them au-

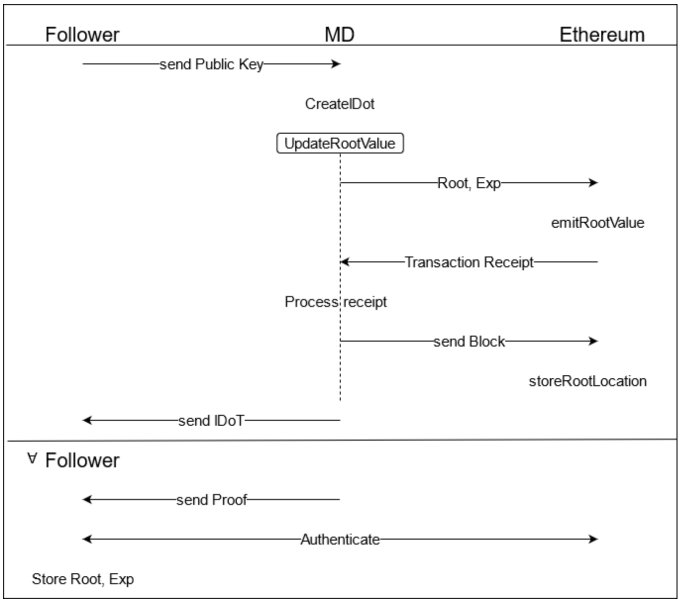


Figure 4.3: Interaction graph on how to add a Follower to a Bubble.

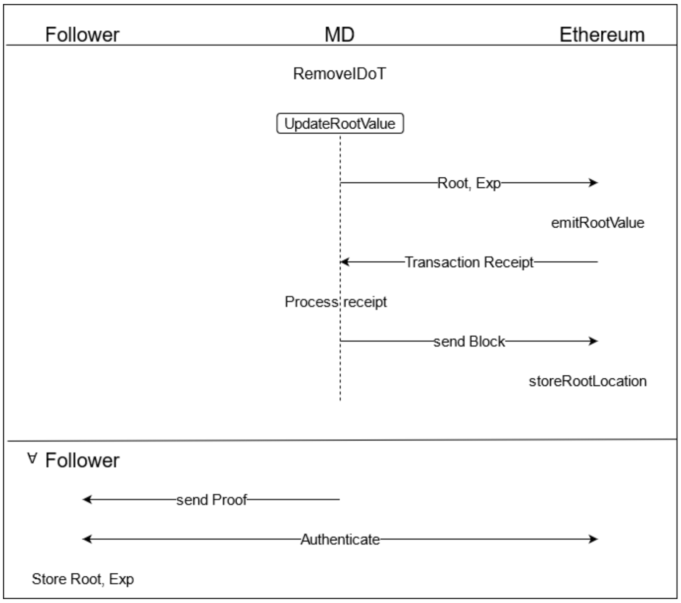


Figure 4.4: Interaction graph on how to remove a Follower from a Bubble.

thentication themselves and storing the root value and expiration date. In the following subsection, we elaborate on how the authentication occurs.

4.2.5. AUTHENTICATION

In this subsection, we present the authentication procedures for our system. We categorise authentication into two types: Bubble-Level and Global-Level authentication. For both types of authentication, the process starts by sending their proof of inclusion and IDoT to the other party. Then, a Follower determines which type of authentication is need by checking the opposing parties Bubble-ID. If the Followers have the same Bubble-ID, then they do Bubble-Level authentication. Otherwise, the Followers execute a Global-Level authentication.

Before discussing each type of authentication, we discuss the core of the authentication procedure. These are the steps used on a Bubble-Level and Global-Level authentication. Algorithm 6 shows the pseudocode. Algorithm 6 contains one function named **Authentication** and takes four parameters. The first parameter is $root$, which is a root value from a Merkle Tree. The second parameter is exp_root, which is the expiration date for $root$. $IDoT$ is the IDoT that is going to authenticated. Last, $proof$ is a proof of inclusion. The function starts by retrieving the current time with the now function to compare with exp_root. If valid the function continues by inserting $IDoT$ in the checkIDoT function. The checkIDoT function checks the integrity of the given IDoT, as well as the time stamp. checkIDoT return True if the timestamp of the IDoT is valid, and the signature is correct, otherwise False is returned. The function then follows by inserting $root$, $IDoT$, and $proof$ into the validateProof method. validateProof check if the proof of inclusion given combined with IDoT result in the root. ValidateProof returns true if that is the case; otherwise, False. If the conditions above are not met, the function returns False.

In the previous subsection, we mentioned that Followers authenticate themselves. The authentication process is similar to a Global-Level authentication, even though it is within the same Bubble. However, this is how Followers get their Bubble's root value and expiration date to execute a Bubble-Level authentication.

Algorithm 6 Core Authentication Procedure

```
1: procedure AUTHENTICATION($root$, $exp\_root$, $IDoT$, $proof$)
2:     if $now() <= exp\_root$ then
3:         if checkIDoT($IDoT$) then
4:             return validateProof($root$, $IDoT$, $proof$)
5:         end if
6:     end if
7:     return False
8: end procedure
```

Bubble-Level Authentication

Bubble-Level authentication is the authentication procedure for devices that are in the same Bubble. Followers within the same Bubble have their identities signed by the same Ethereum wallet. Because of the signature scheme, Follower IDoT has the properties of non-repudiation and unforgeability.

Additionally, Followers within the same Bubble belong to the same Merkle Tree. Intrinsically connecting these identities. Therefore, Followers within the same Bubble can have an elevated trust level in each other because of the combination of being signed by the same Ethereum wallet and being connected via the root value.

Figure 4.5 depicts an overview of the process of authentication on a Bubble-Level. The figure shows two steps. The first step is the authentication process, while the second is establishing the communication between the Followers.

The authentication procedure starts with Followers exchanging their IDoTs and proof of inclusions with each other. The Followers notice that they have the same Bubble-ID. They proceed to use the **Authentication** function with their stored root value and expiration date and the opposing party's IDoT and proof of inclusion as input. The result of the function determines if the Follower establish a session key. Followers within the same Bubble can authenticate each other without external help, making a Bubble-Level authentication a two-way procedure.

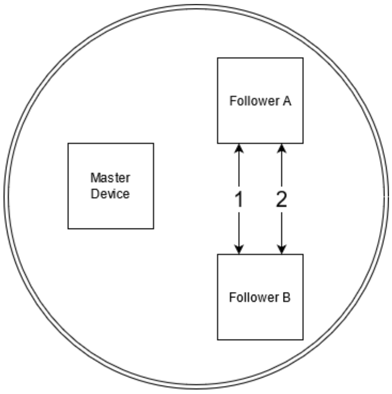


Figure 4.5: Overview Bubble-Level authentication and establishing a connection.

Global-Level Authentication

Global-Level authentication is the authentication procedure for Follower from distinct Bubbles. For Follower to execute a Global-Level authentication they need assistance from the Ethereum blockchain, making it a three-way procedure.

Figure 4.6 depicts the process for Global-Level authentication and communication. Overall the process consist of three steps to authenticate and establish communication. These steps are :

- Exchanging IDoTs and proof of inclusions with the opposing party and noticing a different Bubble-ID.
- The second triggering the **getBlockNumber** with the opposing party's Bubble-ID as input. The result is the block number containing the latest root value for that Bubble-ID. The Follower retrieves the root value and expiration date of the opposing party. These values are inserted in the **Authentication** function to authenticate.

- The result of the **Authentication** function determines if the parties should establish communication between each other.

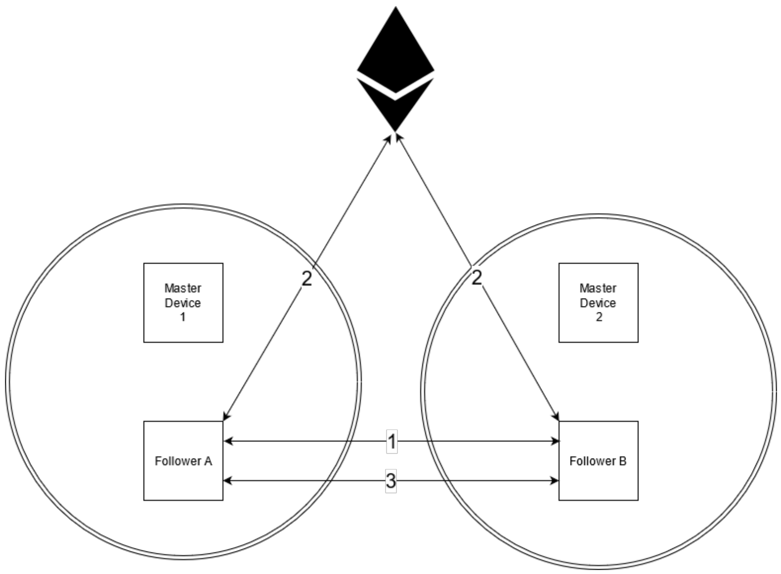


Figure 4.6: Overview Global-Level authentication and establishing a connection.

4.2.6. ESTABLISHING COMMUNICATION

After successful authentication, Followers can establish communication with each other. In this subsection, we discuss how Followers establish communication with each other. Take the example of two Follower: Follower A and Follower B. For the authentication process, each Follower sends their proof of inclusion and IDoT to the other party.

If successful, Follower A takes the Device-Key from Follower B IDoT to execute an ECDH to determine a secret s. Follower B does the same but with its secret and Follower A Device-Key. It results in both Followers having the same secret s. s can be inserted in KDF to utilise a symmetric encryption scheme. Naturally, both Followers need to use the same KDF to achieve the same result. It should be noted that the secret s is static for every ECDH with the same values. Additionally, devices can add randomness to the value before inserting it into the KDF; this can be the time, weather, or two randomly exchanged variables.

4.3. CONCLUSION

In this chapter, we introduced Bubblechain, a decentralised IoT authentication system. Our system refers to a Bubble as a group of devices that belong together to the same owner. A Bubble acts as a trust barrier for devices, which is achieved with the help of Digital Signatures and Merkle Trees. Digital Signatures help determine whether a signature is originated from the same secret key. The Merkle tree helps by connecting these identities. The Ethereum blockchain helps to maintain the state of the latest root of the Merkle Tree. The Ethereum blockchain also makes it possible to retrieve the state of others, making it also possible to authenticate them. With the Ethereum blockchain maintaining the current state of the Merkle Tree, identities can be added, removed, and update by updating the state of the Merkle Tree. At the same time, the only instance that all devices need to be online is for the management process. Also, because our system groups devices that belong together to the same owner together, our system scales to the number of owners instead of the number of devices. In the next chapter, we analyse the Bubblechain system with regards to security and performance. As well as comparing it with the Bubbles of Trust solution, discussed in Chapter 2

5 Analyses

In this chapter, we analyse our system Bubblechain. First, we discuss the security requirements of the Bubblechain system. Second, we present the security model for our system. Thirds, we list the assumptions for our system. Fourth, we discuss the security of the Bubblechain system. Fifth, we cover the theoretical performance of our system. Sixth, we discuss a proof-of-concept with the result. Lastly, we compare our system with the Bubbles of Trust [9].

5.1. Security Requirements

In the previous chapter, we present the IoT authentication system name Bubblechain. In this subsection, we present the security requirements for our presented system. The list of security requirements include:

- **Scalability**: While the expected number of IoT devices is expected to continue growing in the future, this should not lead to malfunctions.
- **Non-Repudiation**: For our system, it refers to the ability to refute a created identity.
- **Availability**: The requirement of availability means that the resources must be accessible to legitimate users.
- **Mutual Authentication**: Mutual authentication is the procedure where both communicating parties authenticate each other. This requirement is necessary to immune the system against spoofing identities.
- **Identification**: Identification refers to the capability of identifying an identity of a device.

5.2. Security Model

In this section, we elaborate on the security model for Bubblechain. We first discuss the network model for Bubblechain, followed by the attacker model.

5.2.1. Network Model

The messages within the system pass through a communication network, which can be an unreliable and potentially lossy network. At the same time, the participants within the network cannot be trusted. The network is tasked solely with forwarding packets and does not provide any security guarantees. For this thesis a malicious user can read or inject network messages.

5.2.2. Attacker Model

For this thesis, we assume that malicious users have some control over the network. A malicious user can sniff, replay, inject, and delay messages arbitrarily with negligible delay. However, within our system, devices can receive unaltered messages. Malicious users potentially can have more computation and storage power than the benign devices within our system.

Within our system, we disregard the possibility of physical attacks on devices to obtain their secret keys. We followed the assumption that these devices can securely store their secret's. There are many methods to protect devices from these types of attacks by making this information readable only by the device itself [56].

5.3. Security Assumptions

In this section, we group the needed assumptions regarding security to make the Bubblechain system functional. Regarding the cryptographic primitives, we assume that our system's chosen hash function, SHA-3, is collision, pre-image, and second pre-image resistant [40]. Additionally, we assume that the elliptic curve discrete logarithm problem (ECDLP) is computationally hard.

Regarding the devices, we assume that storage capabilities to store and maintain their private keys. Devices also can store their identities and execute operations such as SHA-3 operation and ECDH.

For the Blockchain, we have the following assumptions. The first assumption is liveness is guaranteed. The nodes maintaining the Ethereum blockchain can reach a consensus. The second assumption is the safety property. The choice between honest nodes is the same, therefore making it computationally infeasible for an adversary to make honest nodes alter their choice.

5.4. Security Analysis

In this section, we present the security analysis for the Bubblechain system. We divide the security analysis into the structure of the components, the authentication process, and the resiliency against certain attacks.

From the preceding chapters, we note the use of ECDSA, ECDH, and SHA-3. Before discussing the system's security, we discuss the security of these building blocks of the system. The security of ECDH is based on the ECDLP [57]. The security of ECDSA is also based on ECDLP [58]. Additionally, ECDSA can be existentially unforgeable given the right conditions. The conditions to achieve this is stated in Chapter 3.

5.4.1. STRUCTURE

In this subsection, we cover the security of the components for identification. The components are IDoT and the proof of inclusion.

Lemma 5.4.1. (IDoT) An adversary is not able to create a valid IDoT for any Bubble without possessing the secret key.

Proof: In Section 4.2, we presented the structure for the Identities for our system. An IDoT consists of Bubble-ID, Device-ID, Device-Key, Expiration-Date and Signature. Signature is a signature of the hash of the other attributes. The signature is created by an Ethereum Wallet, which is based on ECDSA. Under the assumption that ECDSA is unforgeable, an adversary cannot create a valid signature without the secret key. As a hash function, we used SHA-3, making the digest of the hash second pre-image resistance. □

Lemma 5.4.2. (Proof of inclusion) An adversary is not able to create a proof of inclusion for a IDoT that does not belong to a Bubble, without the secret key.

Proof: An adversary would need to try one of two options: creating a proof of inclusion to match the root value or altering to root value to match a created proof of inclusion. For an adversary to create a proof of inclusion value for a given root value would mean breaking the second pre-image resistance of used hash function. As stated before, SHA-3 is resistant against second pre-image attacks. The root value is signed by an Ethereum wallet and posted on the Ethereum blockchain, making altering the root value without the secret key as difficult as breaking ECDSA. Also, an adversary would need to be able to create an IDoT, which we have proven to be secure against in Lemma 5.4.1.
□

5.4.2. AUTHENTICATION

In this subsection ,we present the security of trust barrier, and the security of Bubble-Level and Global-Level authentication.

Lemma 5.4.3. (Trust Barrier) An adversary cannot create a valid Follower for a specific Bubble without possessing the secret key.

Proof: Based on the assumption of unforgeability of ECDSA, an adversary cannot forge an IDoT for a specific Bubble without the secret key. □

Lemma 5.4.4. (Bubble-Level Authentication) An adversary cannot impersonate a Follower within a specific Bubble without the secret key.

Proof: For Bubble-Level authentication, an adversary would need to falsify an IDoT for a specific Bubble and create a proof inclusion for that Bubble. Lemma 5.4.3 proves that an adversary cannot create an IDoT for a specific Bubble. Lemma 5.4.2 proves that a proof of inclusion for a Bubble is also secure. Therefore proving the security of Bubble-Level authentication. □

Lemma 5.4.5. (Global-Level Authentication) An adversary cannot impersonate a Follower within a different Bubble without the secret key.

Proof: For an adversary to be able to impersonate a Follower it first needs falsify and IDoT. In Lemma 5.4.1 we have proven that IDoT are unforgeable. An adversary would also need to provide a valid proof of inclusion, which we have proven to be secure in Lemma 5.4.2. Therefore, proving the Global-Level authentication is secure. □

Lemma 5.4.6. (Revoked Identity) An adversary is not able to establish a session key with a revoked identity.

Proof: MDs can remove Followers from a Bubble. The removal process entails that MD executes **UpdateRootValue** function. The function makes and emits a new root value on the Ethereum blockchain, and the mapping is updated. Under those circumstances, an adversary would need to be able to forge an IDoT, which we have proven to be secure against in Lemma 5.4.1. Alternatively, an adversay could falsify a proof of inclusion, which we have proven to be secure against in Lemma 5.4.2. Therefore, making our system secure against revoked identities. □

5.4.3. ATTACKS

In this subsection, we discuss our system's resistance against five types of attacks: DDoS, MITM, Sybil, whitewashing and message replay attacks. We choose to discuss MITM, Sybil, whitewashing, and message replay attacks are commonly discussed attacks under IoT systems [17]. Additionally, we discuss DDoS attacks because it is covered by the Bubbles of Trust paper and is a shortcoming of our system.

One of the shortcomings of our system is that it is not DDoS resistant. As a result, adversaries can block the communication between two Followers, making it impossible for them to authenticate each other to establish communication between them.

Lemma 5.4.7. (Sybil Attack) An adversary cannot create identities to undermine the reputation of devices without the secret key.

Proof: Adversaries can create as many Bubbles as they desire. However, those identities are outside of the trusted barrier. The security of creating an IDoT without the secret key is proven in Lemma 5.4.1. Therefore, an adversary cannot create identities within a Bubble to gain an advantage. □

Lemma 5.4.8. (MITM attack) An adversary cannot impersonate an IDoT without their secret key.

Proof: After authentication devices establish communication with each other. The Follower take the Device-Key from the other party IDoT to execute and ECDH. The result of the ECDH key agreement protocol results in a secret value. Therefore, for an adversary to be able to retrieve the secret value it must either break the ECDLP or it must forge an IDoT, which we have proven to be secure in Lemma 5.4.1. Proving that our system is secure against $MITM$ attacks. □

Lemma 5.4.9. (Whitewashing Attack) An adversary cannot create a new identity to replace a compromised identity without the secret key.

Proof: For and adversary to replace its IDoT, it would need the create an IDoT. In Lemma 5.4.1, we have proven that IDoTs are unforgeable. Making our system secure against whitewashing attacks. □

Lemma 5.4.10. (Message Replay Attack) An adversary cannot replay a message to trigger an action.

Proof: Using a salted KDF makes our system secure against message replay attacks. After authentication the Follower determines a shared secret via ECDH. For an adversary to obtain this secret, the adversary would need to be able to break the ECDLP. Alternatively, an adversary can try to forge IDoT, eventhough Lemma 5.4.1 have proven that it not possible. The shared secret is inserted in to a salted KDF. □

5.5. THEORETICAL PERFORMANCE ANALYSIS

In this section, we present the theoretical performance of the Bubblechain system. We start by covering the theoretical cost of managing identities. Then, we follow with the cost of authentication. Finally, we discuss the cost of storage.

5.5.1. COST OF MANAGING IDENTITIES

In this subsection, we cover the theoretical performance for managing the identities of Followers. With managing identities, we refer to the addition, removal and updating of Followers. In Subsection 4.2.4, we presented the steps need to execute each of these three operations.

The number of operations needed for identity management functions is based on the number of Followers within a Bubbles. In Table 5.1, we express the cost for addition, removal, and updating for a Bubble containing N Followers. The first column denoted the number of proof of inclusions that needs to create. The second column shows the number of Ethereum transactions for each operation. Finally, the third column shows the number of Lookup on an Ethereum node to complete the operation.

Table 5.1: Cost of communication of Adding, Updating, and Removing Followers.

	Proof of Inclusions	Ethereum Transactions	Look-Ups
Addition	N+1	2	2 (N+1)
Removal	N-1	2	2 (N-1)
Updating	N	2	2(N)

5.5.2. AUTHENTICATION COST

In this subsection, we cover the communication cost of authentication. The communication cost is dependant on the type of authentication.

For Bubble-Level authentication, a Follower would have its root value and the root value expiration date stored. The Follower can execute an authentication right after receiving the IDoT and proof of inclusion.

For Global-Level authentication, Follower would need to retrieve the value from the Ethereum blockchain. Therefore, each Follower first executes the **getBlockNumber** function to retrieve the block containing the latest root value. The Follower then proceeds to retrieve the event that contains the latest root value. Therefore, making two lookup operation by each Follower. Because each Follower does these operations, it makes the total number of Lookup four for every Global-Level authentication.

5.5.3. STORAGE COST FOR FOLLOWERS

In this subsection, we discuss the number of bits needed to store a device's identity. We start by covering the storage cost of IDoT and follow with the cost of proof of inclusion.

An IDoT is a fixed-sized component, with the cost of storage being the sum of its parts. An IDoT consists out of five parts: Bubble-ID, Device-ID, Device-Key, Expiration-Date, and Signature. Bubble-ID is an Ethereum address, which is forty characters or 160 bits. Device-ID is a value in a 16-bit Integer. Device-Key is a public key of the Elliptic Curve key pair, and we store a compressed version around 33 bytes or 264 bits. The expiration date is a UNIX timestamp stored in 32 bits. Last, the Signature is signature from the Ethereum wallet, which is 65 bytes or 520 bits, making a total of 992 bits or 124 bytes to store an IDoT.

The size of the proof of inclusion is dynamic and based on the number of Followers within a Bubble. The size of proof of inclusion for a Merkle Tree grows according to the binary logarithm [59]. Therefore, the size of the proof of inclusion for a Follower is equal to $\lceil log_2(N) \rceil$, with N being the number of Followers within a Bubble. The size of proof of inclusion is the number of hashes needed, with each hash value being 256 bits.

Additionally, Followers also store the root value of their Bubble and the root's expiration date. The root value is an additional 256 bits, and the expiration date is 32 bits.

5.6. IMPLEMENTATION

To validate our claims, we create a proof-of-concept. We implemented the Bubblechain using the following tools: Truffle, a NodeJs developing environment and testing framework for Ethereum [60]. The smart contract is written in Solidity and is deployed on our Truffle instance. The code for Followers and MDs are written in Python version 3.9.1. The code can be found on our GitHub. As for the elliptic curve, we used the secp256k1 curve for our entire system.

5.6.1. RESULTS

In this subsection, we present the results of experiments we run on our proof of concept. We start by covering the cost of interacting with Bubblechain's smart contract. Followed by presenting the results of timed experiments we ran for Bubble-Level authentication, Global-Level authentication, and executing ECDH key agreement protocol.

Gas Cost

In Table 5.2, we present the gas usage for the three functions within the Bubblechain smart contract. First, we take the gas used because the gas is the unit that measures the computation cost. Then, the gas price gets converted to Ether based on demand. Within the first column, we denote the gas used for each operation. The second column depicts the gas convert into Gwei. Gwei is one-billionth of an Ether. Note, we depicted a range which the market performs. We took the low of 53 GWei per Gas and high of 63 Gwei per Gas from Etherscan on 21th of June 2021 [61]. The last column depicts the conversion rate of Gwei to USD based on the Ethereum evaluation at the moment on coinmarketcap [62].

We notice a few things from our table. The first time executing the **storeRootLocation** uses more gas than the following times. The reasoning would be that the first time the smart contract must allocate space, but the time after that space just get rewritten. The second noteworthy thing is that **getBlockNumber** has no cost. The reasoning is that because the function does not alter the state of blockchain, one can trigger the function with a call instead of a transaction. Calls are a read-only operation done locally by the Node and return the value [63]. The tables also shows of that the price of creating your own Bubble is between $7.69 and $8.56. With the cost of each identity management operation being around $5.97 and $6.66. Based on the conversion rate at that moment.

Table 5.2: Gas cost of Bubblechain smart contract functions.

	Gas Used	Gwei	USD
emitRootValue	25981	1454936 - 1636803	$2.92 - $3.29
new storeRootLocation	41546	2368122 - 2617398	$4.77 - $5.27
update storeRootLocation	26546	1513122 - 1672398	$3.05 - $3.37
getBlockNumber	0	0	$0.0

Bubble-Level Authentication

We measure the duration to authenticate a Follower from the same Bubble. Because certain IoT devices are resource constraint, we simulate the authentication procedure on different memory sizes. We run our test simulating 20, 30, and 60 MB of memory. These values are possible memory capabilities for IoT devices, with devices such as Arduino Yún LininoOS containing 64 MB of memory [64].

By running the authentication process within a docker container, we simulated the different sizes of memory. A docker container is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [65]. The x-axis shows the size of the proof of inclusion. We ran the authentication a hundred times for each size of the proof of inclusion and took the median time. Figure 5.1 depicts our result for Bubble-Level authentication for different proof of inclusions.

Global-Level Authentication

In Figure 5.2, we depict our timed experiments for Global-Level authentication. Similar to our Bubble-Level experiments, we ran our test a hundred times and took the mean. We achieve the different sizes of memory by using docker. In practice, this time may vary due to the time needed for the Node to respond due to influences from the network. Depending on the use case, the time of around two hundred milliseconds might be acceptable. Whereas for other cases, it might be too slow.

Key Agreement Protocol

After successful authentication, Follower used the other Follower's Device-Key to establish a key. The key is used further for symmetric key encryption between the devices. In Figure 5.3, we show the duration for Followers to create a key. With the help of Docker,

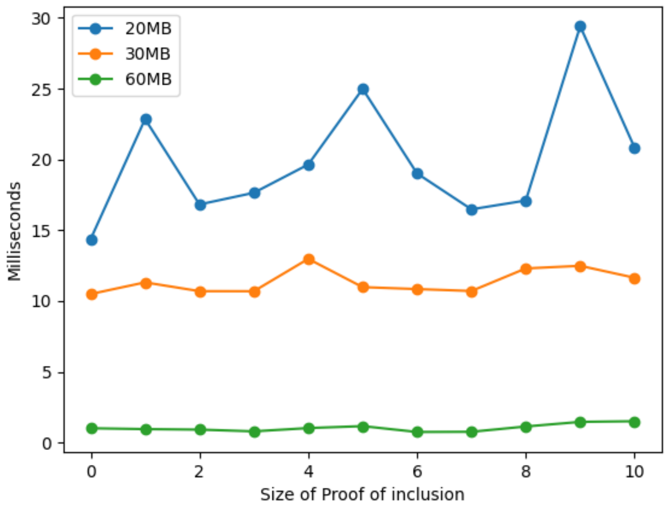


Figure 5.1: Time measurements for a Bubble-Level authentication with different sizes of proof of inclusions.

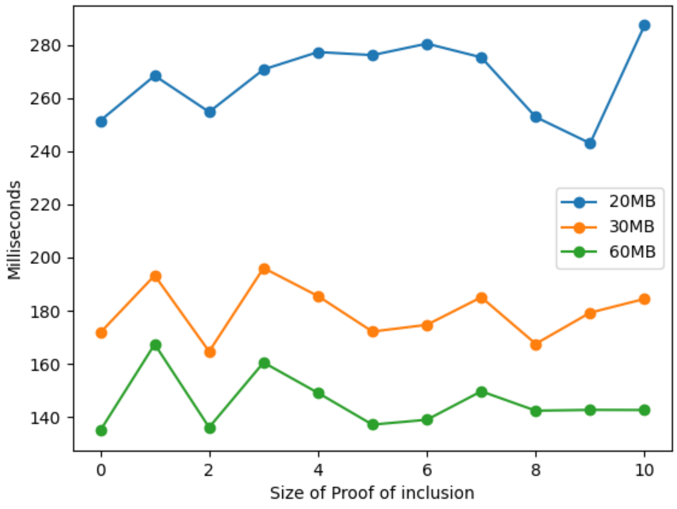


Figure 5.2: Time measurements for a Global-Level authentication with different sizes of proof of inclusion s.

we simulated the results for different sizes of memory. In the figure, we show a box plot of a thousand runs for each memory size. The result shows that in most cases, it takes less than one millisecond to establish a key.

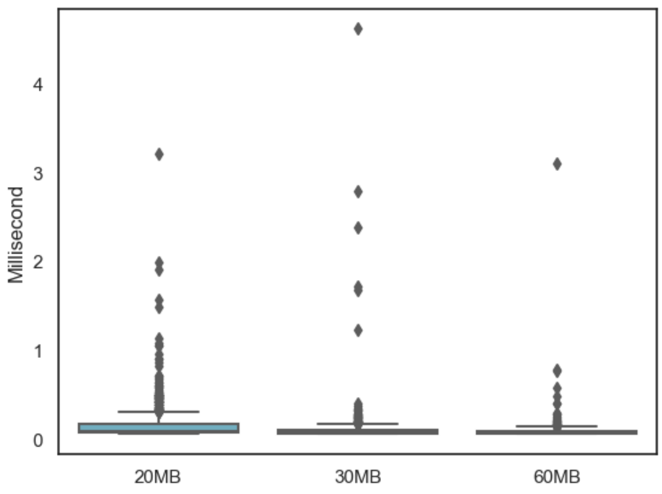


Figure 5.3: Results from timed experiments to execute a ECDH key agreement protocol.

5.7. Bubblechain and Bubbles of Trust Comparison

In Section 2.4, we discussed the closely related work Bubbles of Trust. In this subsection, we compare our system to the Bubbles of Trust. We start by covering the similarities and differences between the two systems. We follow with arguments for each system against the other.

Both systems are smart contract-based IoT authentication on the Ethereum blockchain. Both systems use a centralised and hierarchical architecture for managing identities. Both systems provide a trusted barrier for devices within the same Bubble. Bubblechain and Bubbles of Trust make use of ECC to reduce space. Both systems are resilient against Sybil and MITM attacks for devices within the same Bubble.

The systems differ in the way they authenticate. The Bubbles of Trust systems does the authentication on the smart contract, whereas Bubblechain does it locally on each device. The communication for Bubbles of trust is done via transactions, while Followers in Bubblechain communicate directly with each other. For that reason, each Follower in the Bubbles of trust system would need an Ethereum wallet with ether, which is not the case for Bubblechain. Thus, the trust barrier within our solution is permeable, while the one in Bubbles of Trust is not. Also, Bubblechain can remove identities within a Bubble, while Bubbles of trust cannot do this.

Suppose we would have to argue for using the Bubble of Trust system against our system. We would make the case that adding devices within Bubbles of Trust is more efficient. Adding in Bubblechain requires updating all of the Followers within a Bubble. Their identities are smaller than our system, and Bubbles of Trust is resilient against DDoS attacks.

If we would have to argue the use of Bubblechain ahead of Bubbles of Trust, we would argue the following points :

1. The ability to remove and update identities. Compromised identities can be revoked within our system.

2. Communication within Bubblechain is faster than Bubbles of Trust. Bubbles of Trust communicate via transactions, meaning the speed depends on the Ethereum blockchain pace. Bubblechain does the communication and authentication locally and queries previous values from the Ethereum node if necessary.

3. Bubblechain stores a value to validate identities, whereas all the communication in the Bubbles of Trust system is on the blockchain.

We proceed with the discussion and future work for our system in the following chapter.

6
Discussion & Future Work

In this chapter, we discuss our system, and we cover the future work for Bubblechain. We finalise with some concluding remarks.

6.1. Discussion

For the discussion, we split our system into two parts. First, we cover the research question and the corresponding sub-questions. Then we cover the design choices for our system.

6.1.1. Research Question

In Chapter 1, we proposed the following research question:

> **"How can we design a decentralised generic authentication mechanism for IoT devices that incorporates different trust levels for these devices and also incorporate the life-cycle of devices?"**.

We believe that our propose solution is the answer to this research question. To elaborate on why we believe this. We reiterate and answer each of our previously state subquestions. The subquestions from our research question are:

1. *How can we apply an authentication technique that would be suited for M2M communication?*
   Our proposed solution provides an IoT authentication that autonomously can authenticate other devices. Devices can authenticate any device with the help of the Ethereum blockchain.

2. *How can the a IoT authentication mechanism in question incorporate different trust levels for these devices?*
   Within Bubblechain, we chose for an centralised and hierarchical architecture to manage the identities. The centralised entity, in our system the MD, signs off on

identities with its Ethereum wallet. The used signature schemes provides unforgeability of the signed messages. Therefore, creating an assurance that devices were signed by the same entity. Our simple put creating more trust for these identities.

3. *How to make this IoT authentication mechanism decentralised?*
   To achieve decentralisation we utilise a smart contract on the Ethereum blockchain. The smart contract code maintains the current state of each Bubble, while also being accesible by every entity.

4. *How can individual devices authenticate identities?*
   Within Bubblechain there are two form of authentication Bubble-Level and Global-Level authentication. On the Bubble-Level, devices rely on security of ECDSA to prove that identities are signed by the same key. On the Global-Level, the Ethereum blockchain makes sure that root value of a Bubble is up-to-date.

5. *How can the IoT authentication mechanism be scalable, yet still be able to incorporate the life-cycle of devices?*
   Our solution merges the valid identities of a Bubble into a root value of a Merkle Tree. The Merkle Tree is updated based on the removal, addition and updating of identities. The latest root of value is emitted via an Ethereum event. Which the public nature of blockchain anyone can read it. We utilise a mapping within our smart contract to contain the location of the latest root value.

6.1.2. DESIGN CHOICES

In this subsection, we argue the design choices of our system.

- Ethereum -We wanted a decentralised storage that was already operational and has established security. Additionally, we did not want a device to be online always to maintain the system. In our solution this is done by the Ethereum blockchain. Also full and archived nodes within the Ethereum system must store all the previous state. Making it so that any party can access the root of a Bubble via any full or archived node. However the operation of the Ethereum blockchain is based on the worth of the Ether. In the event that Ethereum blockchain crashes, our system would also follow.

- Identity-based Authentication - Bubblechain relies on authentication based authentication in the form of public keys. The use of public keys has the benefits to be able to rotate or update key values if compromised. Physical-based charactericss authentication such as PUF, are faster and more compact their identity-based counterparts. However, PUF are based on physical characteristics. Altering the characteristics if the circuits is could result in faulty output.

- ECDH and ECDSA - For our identity based solution for IoT devices, the choice of ECC meant smaller key part then their counterparts. Also the use of ECDH made our system more modular. Devices can switch symmetric encryption schemes to cope as much device as possible. Making this choice does bring the problem that if the choice of symmetric encryption scheme is weak, then the whole authentication process was for nothing.

- Centralised and Hierchical Architecture - With our choice of using the Ethereum blockchain, we wanted to have a single entity which has the Ether. In the Bubbles of Trust solution every device would need an Ethereum wallet with ether to execute transactions. Also with the centralised and hierachical approach, we had one device which was capable of signing identities. Upon verification of these identities, can verify that they have been signed by the same key.

- Expiration date - For Both the IDoT and root value we have tied to an expiration date. The reason behind it that even if owner cannot access an MD, the identities do not remain valid forever. However, this introduces more work for the MD, which has to update identities if the expiration date is surpassed

- Storing block number on Smart Contract - Within our smart contract, we emit the Root value and expiration date and store the block number containing the event.The cost of events and storage on the smart contract are based on the size of the data. However events are cheaper than interal storage on the smart contract. The reasoning for this is because events are not accessible later by the contract itself, but stored as a log on Ethereum transaction. Alternatively, we could have stored the root value within the mapping. This approach would have been faster, however is system would not be able to grow. In the future work, we briefly discuss on extending the system to include more then one root value.

6.2. FUTURE WORK

In this section, we cover the future works for our system. These are the areas in which we see room for improvement. There are :

- **Ethereum V2.0** - As previously mentioned, the Ethereum blockchain has plans to switch from PoW to PoS with sharding. Sharding could increase the throughput out the system. However, the data of smart contracts could face the problem of not being accessible by all shards. For our system, this could become a problem or limit our system. Potentiol solutions, could be either to deploy the smart contract on mulitiple shards, or alter the smart contract for multiple shards.

- **White-List Bubbles** - Our solution provides a trusted barrier for device that belong to the same owner. However, communication with devices outside is untrusted. Bubbles that an owner or user know is maintain, can whitelist them. This can avoid quering the blockchain, to avoid cost. This option can avoid cost, but at the cost of not being able to guarantee the latest root version.

- **Split into Multiple Tree** - Instead of an MD having one Merkle Tree, an MD can have multiple trees. Devices within a Bubble can be subdivided into multiple trees. Therefore the MD ensure smaller proof of inclusions, and less expensive updates. However this comes at the cost of more gas being to store the root value and devices need to store more values.

- **Lower Performance alternative**- Alternatively devices that are more resource constraint could get aid from their MD to authenticate Followers on their behalf. The result of the authentication could be relayed to Follower to determine whether it should establish a connection. This approach, without implementing secure channel between the MD and Follower, can lead to malicious entities impersonating an MD.

6.3. CONCLUDING REMARKS

In this thesis, we presented Bubblechain. Bubblechain is a decentralised IoT authentication system, with two trust levels and capable of adding, removing and update identities of devices. To achieve these capabilities, we made decisions that come at a cost. We now discuss the practicality of our system.

In Chapter 2, we categorised various authentication schemes according to their setting. Their setting influences the requirements for each system. From our analyses, we determine that Bubblechain is not suited for all settings. For settings that require a low latency for devices within different owners our system would not be ideal. For example, in a transportation setting where latency could result in loss of life.

In settings with a large number of devices belong to the same owner our system as it stands is not ideal. A large number of devices, would result in a large Bubble. A large Bubble would not have a significant impact on the authentication time, however processes to manage the identities would be extremely expensive, requiring every device's proof of inclusion to be updated. However, by extending our work to incorporate white listed Bubbles and split the Bubbles into various group. Our solution could be better equipped for these scenarios.

Our system is more suitable for the smart home setting. Because smart homes can have a relatively small number of devices, and most of communication is within the same Bubble. Also the appliances are not frequently replaced. Therefore, the management of identities would be feasible.

REFERENCES

[1] *Google nest: Why google finally embraced nest as its smart home brand - the verge,* https://www.theverge.com/2019/5/7/18530609/google-nest-smart-home-brand-merging-hub-max-rebrand-io-2019, (Accessed on 01/06/2021).

[2] *Why samsung's smartthings is pivoting from hardware to software,* https://www.businessinsider.com/exclusive-why-samsungs-smartthings-is-going-all-in-on-software-2020-6, (Accessed on 05/21/2021).

[3] R. Kadam, P. Mahamuni, and Y. Parikh, *Smart home system,* International Journal of Innovative research in Advanced Engineering (IJIRAE) **2** (2015).

[4] J. Rivera and L. Goasduff, *Gartner says a thirty-fold increase in internet-connected physical devices by 2020 will significantly alter how the supply chain operates,* Gartner (2014).

[5] G. W. Hart, *Nonintrusive appliance load monitoring,* Proceedings of the IEEE **80**, 1870 (1992).

[6] *Hackers remotely kill jeep's engine on highway,* https://www.cnbc.com/2015/07/21/hackers-remotely-kill-jeep-engine-on-highway.html, (Accessed on 08/05/2020).

[7] *Keyless: danger car theft | adac,* https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/assistenzsysteme/keyless/, (Accessed on 08/05/2020).

[8] *Radio attack lets hackers steal 24 different car models | wired,* https://www.wired.com/2016/03/study-finds-24-car-models-open-unlocking-ignition-hack/, (Accessed on 08/05/2020).

[9] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, *Bubbles of trust: A decentralized blockchain-based authentication system for iot,* Computers & Security **78**, 126 (2018).

[10] M. Luecking, C. Fries, R. Lamberti, and W. Stork, *Decentralized identity and trust management framework for internet of things,* in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (IEEE, 2020) pp. 1–9.

[11] *What is the lifespan of a smartphone? - coolblue - before 23:59, delivered tomorrow,* https://www.coolblue.nl/en/advice/lifespan-smartphone.html (), (Accessed on 08/11/2020).

[12] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, *Future internet: the internet of things architecture, possible applications and key challenges,* in *2012 10th international conference on frontiers of information technology* (IEEE, 2012) pp. 257–260.

[13] I. Mashal, O. Alsaryrah, T.-Y. Chung, C.-Z. Yang, W.-H. Kuo, and D. P. Agrawal, *Choices for interaction with things on internet and underlying issues,* Ad Hoc Networks **28**, 68 (2015).

[14] O. Said and M. Masud, *Towards internet of things: Survey and future vision,* International Journal of Computer Networks **5**, 1 (2013).

[15] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, *Research on the architecture of internet of things,* in *2010 3rd international conference on advanced computer theory and engineering (ICACTE),* Vol. 5 (IEEE, 2010) pp. V5–484.

[16] Q. Wen, X. Dong, and R. Zhang, *Application of dynamic variable cipher security certificate in internet of things,* in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, Vol. 3 (IEEE, 2012) pp. 1062–1066.

[17] M. El-Hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, *A survey of internet of things (iot) authentication schemes,* Sensors **19**, 1141 (2019).

[18] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, *Internet of things (iot) security: Current status, challenges and prospective measures,* in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)* (IEEE, 2015) pp. 336–341.

[19] F. Chu, R. Zhang, R. Ni, and W. Dai, *An improved identity authentication scheme for internet of things in heterogeneous networking environments,* in *2013 16th International Conference on Network-Based Information Systems* (IEEE, 2013) pp. 589–593.

[20] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, and M. Ylianttila, *Two-phase authentication protocol for wireless sensor networks in distributed iot applications,* in *2014 IEEE Wireless Communications and Networking Conference (WCNC)* (Ieee, 2014) pp. 2728–2733.

[21] K. Mahmood, S. A. Chaudhry, H. Naqvi, T. Shon, and H. F. Ahmad, *A lightweight message authentication scheme for smart grid communications in power sector,* Computers & Electrical Engineering **52**, 114 (2016).

[22] D. Chen, N. Zhang, Z. Qin, X. Mao, Z. Qin, X. Shen, and X.-Y. Li, *S2m: A lightweight acoustic fingerprints-based wireless device authentication protocol,* IEEE Internet of Things Journal **4**, 88 (2016).

[23] S. Karthikeyan, R. Patan, and B. Balamurugan, *Enhancement of security in the internet of things (iot) by using x. 509 authentication mechanism,* in *Recent Trends in Communication, Computing, and Electronics* (Springer, 2019) pp. 217–225.

[24] A. Singh and K. Chatterjee, *A secure multi-tier authentication scheme in cloud computing environment,* in *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (IEEE, 2015) pp. 1–7.

[25] J. Shao, X. Lin, R. Lu, and C. Zuo, *A threshold anonymous authentication protocol for vanets,* IEEE Transactions on vehicular technology **65**, 1711 (2015).

[26] M. N. Aman, K. C. Chua, and B. Sikdar, *Mutual authentication in iot systems using physical unclonable functions,* IEEE Internet of Things Journal **4**, 1327 (2017).

[27] D. Mukhopadhyay, *Pufs as promising tools for security in internet of things,* IEEE Design & Test **33**, 103 (2016).

[28] A. C.-F. Chan and J. Zhou, *Cyber–physical device authentication for the smart grid electric vehicle ecosystem,* IEEE Journal on Selected Areas in Communications **32**, 1509 (2014).

[29] H. Li, R. Lu, L. Zhou, B. Yang, and X. Shen, *An efficient merkle-tree-based authentication scheme for smart grid,* IEEE Systems Journal **8**, 655 (2013).

[30] D. Li, Z. Aung, J. R. Williams, and A. Sanchez, *Efficient authentication scheme for data aggregation in smart grid with fault tolerance and fault diagnosis,* in *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)* (IEEE, 2012) pp. 1–8.

[31] M. A. Muhal, X. Luo, Z. Mahmood, and A. Ullah, *Physical unclonable function based authentication scheme for smart devices in internet of things,* in *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)* (IEEE, 2018) pp. 160–165.

[32] L. Zhang, C. Hu, Q. Wu, J. Domingo-Ferrer, and B. Qin, *Privacy-preserving vehicular communication authentication with hierarchical aggregation and fast response,* IEEE Transactions on Computers **65**, 2562 (2015).

[33] K. B. Frikken, M. Blanton, and M. J. Atallah, *Robust authentication using physically unclonable functions,* in *International Conference on Information Security* (Springer, 2009) pp. 262–277.

[34] N. F. Pub, *197: Advanced encryption standard (aes),* Federal information processing standards publication **197**, 0311 (2001).

[35] Z. Gong, S. Nikova, and Y. W. Law, *Klein: a new family of lightweight block ciphers,* in *International Workshop on Radio Frequency Identification: Security and Privacy Issues* (Springer, 2011) pp. 1–18.

[36] D. Williams, *The tiny encryption algorithm (tea),* Network Security , 1 (2008).

[37] V. S. Miller, *Use of elliptic curves in cryptography,* in *Conference on the theory and application of cryptographic techniques* (Springer, 1985) pp. 417–426.

[38] F. Vercauteren *et al.*, *Final report on main computational assumptions in cryptography,* European Network of Excellence in Cryptography II **11** (2013).

[39] H. Delfs, H. Knebl, and H. Knebl, *Introduction to cryptography,* Vol. 3 (Springer, 2015).

[40] M. J. Dworkin, *Sha-3 standard: Permutation-based hash and extendable-output functions,* (2015).

[41] D. R. Brown, *Generic groups, collision resistance, and ecdsa,* Designs, Codes and Cryptography **35**, 119 (2005).

[42] L. Ramabaja and A. Avdullahu, *Compact merkle multiproofs*, arXiv preprint arXiv:2002.07648 (2020).

[43] H. Stančić, A. Babić, N. Bonić, M. Kutleša, I. Volarević, V. Bralić, A. Lendić, and I. Slade-Šilović, *Blockchain technology for record keeping: Help or hype?* Blockchain technology for record keeping: Help or Hype? , 103 (2016).

[44] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction* (Princeton University Press, 2016).

[45] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, Tech. Rep. (Manubot, 2019).

[46] *Proof-of-work (pow) | ethereum.org*, https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/ (), (Accessed on 02/26/2021).

[47] P. Robinson, *The merits of using ethereum mainnet as a coordination blockchain for ethereum private sidechains*, The Knowledge Engineering Review **35** (2020).

[48] *Proof-of-stake (pos) | ethereum.org*, https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (), (Accessed on 02/26/2021).

[49] *Ethereum 2.0: What you need to know - the street crypto: Bitcoin and cryptocurrency news, advice, analysis and more*, https://www.thestreet.com/crypto/ethereum/ethereum-2-upgrade-what-you-need-to-know#:~:text=Right%20now%2C%20Ethereum%20can%20only,using%20sharding%20and%20other%20tactics., (Accessed on 06/02/2021).

[50] *Shard chains | ethereum.org*, https://ethereum.org/en/eth2/shard-chains/, (Accessed on 06/02/2021).

[51] G. W. Peters and E. Panayi, *Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money*, in *Banking beyond banks and money* (Springer, 2016) pp. 239–278.

[52] V. Buterin *et al.*, *Ethereum: A next-generation smart contract and decentralized application platform*, https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-White-Paper **7** (2014).

[53] *Nodes and clients | ethereum.org*, https://ethereum.org/en/developers/docs/nodes-and-clients/, (Accessed on 06/02/2021).

[54] *Are ethereum full nodes really full? an experiment. | by marc-andré dumas | medium*, https://medium.com/@marcandrdumas/are-ethereum-full-nodes-really-full-an-experiment-b77acd086ca7#:~:text=But%20in%20a%20nutshell%2C%20full,for%20every%20block%20since%20genesis., (Accessed on 06/02/2021).

[55] N. P. Smart *et al.*, *Cryptography: an introduction*, Vol. 3 (McGraw-Hill New York, 2003).

[56] D. Bong and A. Philipp, *Securing the smart grid with hardware security modules,* in *ISSE 2012 Securing Electronic Business Processes* (Springer, 2012) pp. 128–136.

[57] R. Haakegaard and J. Lang, *The elliptic curve diffie-hellman (ecdh),* Online at https://koclab. cs. ucsb. edu/teaching/ecc/project/2015Projects/Haakegaard+ Lang. pdf (2015).

[58] D. Johnson, A. Menezes, and S. Vanstone, *The elliptic curve digital signature algorithm (ecdsa),* International journal of information security **1**, 36 (2001).

[59] J. Kuszmaul, *Verkle trees,* Verkle Trees , 1 (2019).

[60] *Sweet tools for smart contracts | truffle suite,* https://www.trufflesuite.com/, (Accessed on 06/14/2021).

[61] *https://etherscan.io/gastracker,* https://etherscan.io/gastracker, (Accessed on 06/21/2021).

[62] *Cryptocurrency converter and calculator tool | coinmarketcap,* https://coinmarketcap.com/converter/eth/usd/, (Accessed on 06/21/2021).

[63] *Truffle | interacting with your contracts | documentation | truffle suite,* https://www.trufflesuite.com/docs/truffle/getting-started/interacting-with-your-contracts, (Accessed on 06/21/2021).

[64] *Arduino - arduinoboardyun,* https://www.arduino.cc/en/Main/ArduinoBoardYun/, (Accessed on 06/15/2021).

[65] *What is a container? | app containerization | docker,* https://www.docker.com/resources/what-container (), (Accessed on 06/15/2021).