FlyTrap: Decentralised Blockchain Security & Auditing Architecture for IoT and MQTT Brokers

Konrad M. Dryja

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master in Science
of the
University of Aberdeen.

Department of Computing Science

2020

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: April 14, 2020

Abstract

An expansion of the title and contraction of the thesis.

Acknowledgements

Much stuff borrowed from elsewhere

Contents

Abbreviations

ABI Application Binary Interface.

ACL Access Control List.

API Application Programming Interface.

BTC Bitcoin.

CCPA California Consumer Privacy Act.

CLI Command Line Interface.

DPA Data Protection Act.

ETH Ethereum.

EVM Ethereum Virtual Machine.

GDPR General Data Protection Regulation.

IoT Internet of Things.

JSON Javascript Object Notation.

MQTT Message Queuing Telemetry Transport.

PII Personal Identifiable Information.

PoA Proof-of-Authority.

PoS Proof-of-Stake.

PoW Proof-of-Work.

QoS Quality of Service.

RFID Radio-Frequency Identification.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

Chapter 1

Introduction

1.1 Overview

1.1.1 Internet of Things

Internet of Things, also known as IoT, is a growing field within technical industries and computer science. It is a notion first coined in Ashton [1] where the main focus was around RFID (radio-frequency identification) tags - which was a simple electromagnetic field usually created by small-factor devices in the form of a sticker capable of transferring static information, such as a bus timetable or URL of a website (e.g. attached to a poster promoting a company or an event). Ashton argued the concern of data consumption and collection being tied to human presence at all times. In order to mine information, human first was required to find relevant data source, which then could be appropriately evaluated. However, as it was accurately pointed out, people have limited resources & time, and their attention could not be continuously focused on data capture. Technologist suggested delegating the task to the machines themselves; altogether remove the people from the supply chain. A question was asked, whether "things" could collect data from start to finish. That paper is known to be the first mention of IoT and building stone, de facto defining it as an interconnected system of devices communicating with each other without the need for manual intervention.

With time and ever-expanding presence of smartphones, personal computers and intelligent devices, the capabilities of those simple RFID tags were also growing beyond just a simple static data transmission functionalities. Following the observation by Moore et al. [23], the size of integrated circuits was halving from year to year, allowing us to put more computational power on devices decreasing in size. They were now not only capable of acting as a beacon, but actively process the collected information (for example, temperature) and then pass it along to a more powerful computer which then could make decisions on whether to increase or decrease the strength of radiators at home - all without any input from the occupants. Eventually, IoT found their way to fields and areas such as households (smart thermostats or even smart kettles), physical security (smart motion sensors and cameras) or medicine (smart pacemakers). The scope is expected only to grow in the future. Data from Juniper Research [28] and CISCO [11] suggests that total number of IoT devices might reach 50 billion by 2023.

1.1.2 Security of data

The growing presence of smart-devices significantly increased the convenience and capabilities of "smart-homes" - at the same time, IoT also started handling more and more sensitive data - especially considering the last example from the previous paragraph. Scientists from the University of Massachusetts successfully performed an attack on a pacemaker [15], reconfiguring the functionality, which - if performed with malicious intents - could have tragic consequences. Nevertheless, even less extreme situations, such as temperature readings at home, are nowadays heavily regulated by data protection laws. Examples being the General Data Protection Regulation (GDPR) introduced by European Commission [10] or California Consumer Privacy Act by California State Legislature [7]. Collection of data is required to be strictly monitored and frequently audited in case of a breach - which also includes restrictions on the collection of Personal Identifiable Information (PII, as per GDPR). Those and more put an obligation on every company willing to exchange user data to govern the data appropriately and ensure its security - which includes data collected by Internet of Things devices.

1.1.3 MQTT

IoTs are usually low-power with limited computational power - mostly to decrease the required maintenance and ensure long-lasting life, without the need of replacing the power source (which is often a fixed battery) - meaning that only minimum amount of work should be performed on the "thing" itself, instead of sending it off to a centralised structure (e.g., a server hosted on the cloud) for further processing. One of the popular choices includes an intermediary, a broker, relaying communication between clients connected to it. That way, Peer-to-Peer connection is not required and can be wholly delegated to separate backend server. A popular choice for the broker is MQTT (Message Queuing Telemetry Transport)[1] standard defining the exact shape and form of TCP packets, handling unexpected timeouts & reconnects along with distributing channels of communication onto different topics containing separated information. From there, clients can either subscribe (i.e. consume) or publish (which can also be used for issuing commands) the data. Unfortunately, the OASIS standard introduces limited security capabilities (offering only username/password authentication) and no auditing or logging.

1.1.4 FlyTrap

This project will be aiming to develop a novel approach - further referred to as **FlyTrap** - for handling security in systems utilising MQTT brokers and their implementations, focusing on platform-agnostic solution hosted within a containerised environment. It will not depend on the specific software implementing the broker but instead will aim to work with any broker that fully implements MQTT v5.0 standard. Furthermore, to ensure decentralised operation resistant to data breaches, downtime and full transparency, Ethereum[2] platform would be used as a data layer: capturing relevant interaction as publicly available transactions. In order to limit the quantity of data put on the blockchain (as computational and storage power there is limited), I will also introduce several rules dictating logging of only specific events. The system's purpose is to incorporate **Authentication, Authorisation** and **Accountability** (AAA) framework to IoT devices communicating through MQTT.

[1] https://mqtt.org/
[2] https://ethereum.org/

1.2 Motivation

1.2.1 MQTT

MQTT v5.0 (as per the specification[3]) does not dictate nor specify any requirements regarding the security. It does offer an option of restricting some topics only to specific users, defined in access control lists (ACLs). The users then are required to provide a password when initiating a connection with the broker. Although, the basic username/password authentication is known to be cumbersome, only offering limited security. This also puts a burden on system administrators to maintain those ACLs in some centralised system, which then again is at risk of breaches or leakage. Moreover, placing the burden on a singular MQTT broker creates a single point of failure, where system downtime could halt the entire architecture.

1.2.2 Blockchain

By decentralising the data layer of the AAA framework and in the process, placing it on a distributed ledger, I can ensure maximised uptime and complete transparency of performed transactions. Events such as permission changes, failed authentication attempts will be recorded as a separate transaction which then could be audited by anyone knowing the public address of the system. This then could be handed over to authorities or auditing corporations to ensure that data is passed lawfully. Utilising Blockchain technologies also opens an opportunity to require payment (in the form of cryptocurrency) from potential consumers of data effectively expanding the business model.

1.2.3 Legislature

The rise of awareness of the necessity of data protection also encouraged governments to introduce legal requirements (such as GDPR or CCPA) of data governance and face heave fines in case of non-compliance. MQTT standard and their implementation at the moment would be considered non-compliant, due to effectively no way to trace past operations. General Data Protection Regulation requires entities handling user data to maintain proper retention of data and purge if requested by the data owner. MQTT at its current state is not capable of either, as messages are removed from the broker as soon as they are consumed (with small exceptions), leaving no trace of "who" accessed "what" (not to mention questions such as "why" they accessed it).

1.3 Goals

The project can be divided onto four main goals and two extras, leaving some field for manoeuvring in case of roadblocks or difficulties resulting from the challenges faced in the dissertation. By having soft targets, I will be able to stop sooner in case of overestimating the schedule, or carrying on with extra work, should I find myself meeting the targets quicker than expected.

Main Goals:

1. Design structure of blockchain network, relevant data models that would be placed on the blockchain and deploy on the Ethereum platform, capable of recording transactions and allowing for modification of ACLs, i.e. which wallets are permitted to access specific resources on the MQTT brokers.

[3] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

2. Design rules that would be used for capturing the transactions. For example, a rule stating that if the client makes more than five consecutive, failed authentication attempts would be placed on a blacklist and that action would be added onto the blockchain as a transaction.

3. Design containerised software acting as a secure proxy between brokers and connecting clients. This will handle both authentication and log performed action as an immutable transaction on a blockchain network. Logging will only be performed if the requested operation triggers some pre-defined rules.

4. Perform evaluation of the designed solution using an off the shelf MQTT broker and a range of experimental scenarios with a simulated network of MQTT clients.

Extra Goals:

1. Create public API for the auditors to freely access the contents of blockchain and thus transactions containing information about suspicious operations.

2. Generalise the implementation of the framework so it can be deployed with any broker following the MQTT standard.

What project is NOT trying to be:

- Design a new blockchain platform from scratch. Rather existing solution - Ethereum - is going to be used.
- Write / modify operating system of IoT devices.
- Design a new MQTT Broker. The system is going to be built on top of MQTT layer.

1.4 Report Structure

The dissertation is going to be divided onto seven chapters, each describing the following aspects of the project:

Chapter 1 **Introduction** chapter will outline the main motivation behind the project and introduce the notions used a building block in the design. It will also list goals and no-goals defining success.

Chapter 2 In **Background & Related Work**, similar research and state of the art will be described along with outlining the differences between them and this project. Furthermore, a thorough explanation of used software will also be attached, such as what is blockchain, Ethereum, MQTT.

Chapter 3 **Requirements** will include analysis of both functional and non-functional requirements, main use-cases that are driving the project and provide stories inspired by real-life scenarios on the kind of problems that this project is trying to address.

Chapter 4 **Design** will provide a thorough overview on the design of the project, explaining what components is the software composed of and detailing their operation & interconnectivity.

Chapter 5 **Implementation** will talk about the process of implementation of the design into software. It will include notions such as followed processes, used frameworks and sample code snippets.

Chapter 6 Inside **Testing & Evaluation** a comparison between state-of-the-art software, vanilla and FlyTrap will be performed. Tests checking for performance impact, cost of operation on public blockchain, and whether common attacks can be detected/stopped will also be run.

Chapter 7 **Discussion & Future Work** will include conclusions of the project, elements that were left-over, but beneficial for future iteration and all blockages encountered throughout.

Chapter 2

Background & Related Work

In this section, I will list all technologies that are used in this project along with discussing other papers which were trying to address security with IoT devices by also trying to include blockchain technology.

2.1 Legal Background

One of the biggest motivators for this project are the recent changes in laws concerning how user data should be stored and treated. Prior to that, there was no exact limitations or regulations and companies often suffered no consequences (other than loss of trust by the customers) in case of leaking their information. This has changed severely in 2018, when General Data Protection Regulation (also known as GDPR) was introduced. Ever since lot of companies, such as Google rushed to ensure their compliance - but in the end to no avail, as both Facebook and Google suffered fines as high as $8.8 billion on the very first day when the bill became enforceable [4].

Truth is, the regulation changed the perspective on how digital industrial should be managing the information of their customers in a substantial way. First and foremost, Article 17 of GDPR, titled Right to erasure (or, 'right to be forgotten'). This particular point forces all industries to allow their customers to request data erasure at any point, with no questions asked (with small exceptions, such as banking, where the data is retained because of money laundering or terrorism laws). Then, the company is forced to guarantee erasure of all logs - and if it's determined that it was not the case - face heavy fines.

The notion of Personal Identifiable Information was also put into law with special regulations on how such data should be handled. Article 4 of GDPR (titled 'Definitions') describes it as:

> 'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;

Such data should either be anonymized before storing or access should be heavily audited and monitored, to minimise chance of leakage - and again, in case of non-compliance - face large fines.

Though, GDPR was only the beginning and other jurisdictions followed suit, aiming to introduce similar protections. The UK Government decided to ratified slightly modified version of

GDPR into their own law and called it Data Protection Act 2018. Most recent example being state of California, introducing California Consumer Privacy Act (or CCPA for short) put into law on January 1st, 2020 [7]. It shared lot of similarities with GDPR and DPA, such as right to erasure, defining personal information, setting penalties and aiming to protect consumer's and their data.

2.2 MQTT

When designing architecture with the main target being the communication of many (even couple of thousands a second) clients continually exchanging data, scalability and availability needs to be kept in mind. The first and obvious solution would be to directly connect data consumers and data produces, by making them communicate in Peer-to-Peer fashion, removing the need for any extra infrastructure. This might work perfectly fine with small systems (disregarding issues such as dynamic DNS or static IP). However, as the number of clients requesting access to data increases, the total capacity of the sensor would eventually be capped - since IoT usually are of limited power and computation capacity. Imagine a scenario where a single temperature sensor continually getting bombarded with requests for current readings. It might be able to cope up to 5 incoming requests every second, everything else would cause malfunction or significantly slower response times.

Then there is also an issue of security. By allowing clients to connect to our IoT devices, we are opening a new attack vector. What if the client does not want only to access the temperature readings, but perhaps inject a worm which would intercept other sensors (such as cameras). Recently "smart nannies", responsible for alerting the parents when the child is crying and also relieving the adults from having to be always nearby, gained popularity. A direct camera feed could be accessed via a smartphone, no matter where. This eventually led to exploitation, as it was found that many of those devices were vulnerable to remote access by third parties[26].

MQTT aims to address those issues (and not only), by moving the communication to a separate entity, which operates in a publish-subscribe fashion. This would mean that IoT devices only have to publish information that is available to them (e.g. temperature readings), allowing to altogether remove remote access, effectively mitigating this particular attack vector. Furthermore, the MQTT brokers can be further placed behind load balancers and such to enhance their availability further.

In short, MQTT, fully expanded to Message Queuing Telemetry Transport is an open protocol, certified by OASIS and ISO[2], responsible for the publisher-subscriber architecture. It is important to point out that MQTT is not a piece of software or a server, but rather a set of standards defining what potential clients can expect (what kind of responses and data) while connection to brokers following the standard. Figure 2.1 briefly shows how MQTT-compatible broker can relay information between clients. Smart Phone and Smart Kettle do not have to be online at the same time in order to receive information, nor Smart Phone is even permitted to initiate a direct connection to Smart Kettle. The broker's responsibility is to track connected subscribers (which must specify the topic of their interest) and maintain the connection until subscribe advertises session termination or abruptly disconnects (e.g. loss of power or unreliable connection).

In MQTT architecture, Client ID identifies each of the connecting entities (publisher/subscriber) and topic identifies a bridge between publishers and subscribers connected to the same topic.

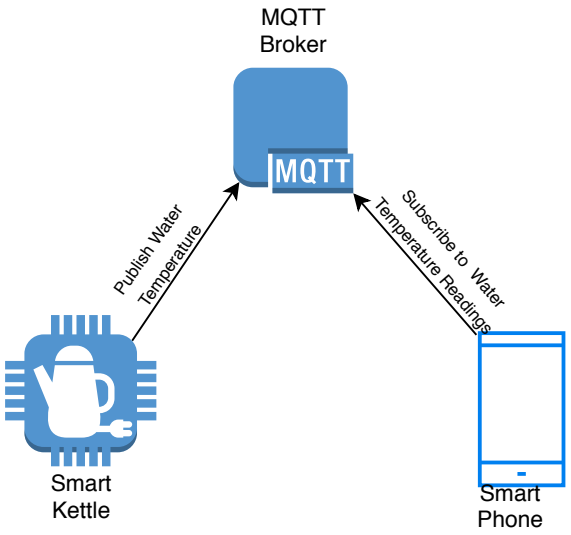

Figure 2.1: MQTT Broker Architecture

For example, a Smart Kettle could be publishing temperature readings under a topic called "UK/Aberdeen/Kettle" - then, a smartphone would need to request the same topic to receive those readings.

2.2.1 Message persistence

This will be discussed in depth when I will describe the process of publishing and subscribing, but it is worth pointing out that by default, the messages are not saved nor cached on the broker. That is if Kettle publishes the temperature reading, but no subscribers are listening to this information, the message will perish. This is not ideal, for a situation where a smart device could wake up only every couple of minutes and then go to low-power mode again. To address this, MQTT Messages can be enriched by "Retain" flag. If such a flag is present, the broker will keep the message and send it straight away to any new subscribers requesting given topic. This is also useful for issuing commands to IoT devices. For example, a phone could send a command to turn off the lights with "Retain" flag set. Then, the smart light switch could check for retained messages every couple of minutes, removing the need for constant connection.

2.2.2 Implementation

MQTT by itself is only a collection of standards instructing implementors on what patterns should be followed and the structure of particular messages. Thus it is not shipped with any piece of software. It assumes operation on TCP layer of the network (although newer versions also allow for WebSocket support [22]), thus also allowing for encrypted connection via Transport Layer Security. Every exchanged message is a TCP packet, following a strict convention- which in case of deviation is discarded as corrupted.

Two of the implementations that I have considered during this project are Mosquitto[1] by Eclipse and Moquette[2]. The former written in C and the former in Java, although there is many, many more. In a paper by de Oliveira et al. [9], scientists compare Moquitto and RabbitMQ, arguing their choice by the offered cloud infrastructure with more significant scalability opportunities.

[1] https://mosquitto.org/
[2] https://github.com/moquette-io/moquette

Moreover, some solutions are paid, whereas the considered approaches are free and open-source, allowing for a better understanding of operations. The paper is concluded with the finding that hardware and network latency has a far more significant impact on the performance, rather than the choice of the individual broker, which leaves the decision mostly down to offered extra features.

Mosquitto also offers a Docker container [21] in which the broker can be run, allowing for further isolation and removal of extra dependencies.

2.2.3 Publishing

The most popular method of passing MQTT messages is still under the Transport layer, as TCP packets. This allows for slightly higher freedom (compared to stricter protocols, such as HTTP), at the cost of more sophisticated parsing. MQTT standard is composed of several message types with the most important being:

- CONNECT - used to initiate the connection
- PUBLISH - used by the client to publish messages and by the broker to publish messages to subscribers
- SUBSCRIBE - used by the client to request a subscription to a given topic
- UNSUBSCRIBE - used by the client to request removal of subscription to given topics
- Along with relevant *ACK counterparts (e.g. CONNACK) used to indicate the successful transmission of the message

As shown in figure 2.2, the publishing flow starts with the CONNECT messages. Inside, there are several flags included, such as Quality of Service requested (MQTT can periodically send heartbeat ping to clients to check if they are still alive), requested version of MQTT protocol (at the moment, v5.0 and v3.1). This part is also referred to as “Variable header”. The second part, known as “Payload” consists of the client ID.

Then, once the client has established its identity to the broker, the broker responds with CONNACK message, which contains bit informing whether a further connection is allowed or not. From this point, the client is cleared to start publishing session.

Usually, for every message to be published, there is one PUBLISH packet. A newer version of MQTT allows for spreading larger messages across multiple packets, although this will not be covered in this paper. The PUBLISH packet contains mostly two properties - topic to be published on and the actual payload. Each of the properties is prepended with 8 bytes indicating the length. From this fact, we can derive the maximum possible size of individual payload - 65535 characters (pure ASCII, no Unicode, which may take more than 1 bytes per character). Same as with CONNECT, each message is responded to with PUBACK, acting as a receipt for receiving the payload.

The client can continue to publish new messages without having to connect again, as long as the TCP session has not been terminated. Should the client want to disconnect, it should follow standard TCP flow, i.e. issue FIN/ACK packet to the broker. For situation, where the connection has been terminated abruptly, there are options such as Will flag (message to pass in case of sudden

disconnection) or Keep Alive (to indicate how long should the connection be kept alive for before assuming the client has lost connection).

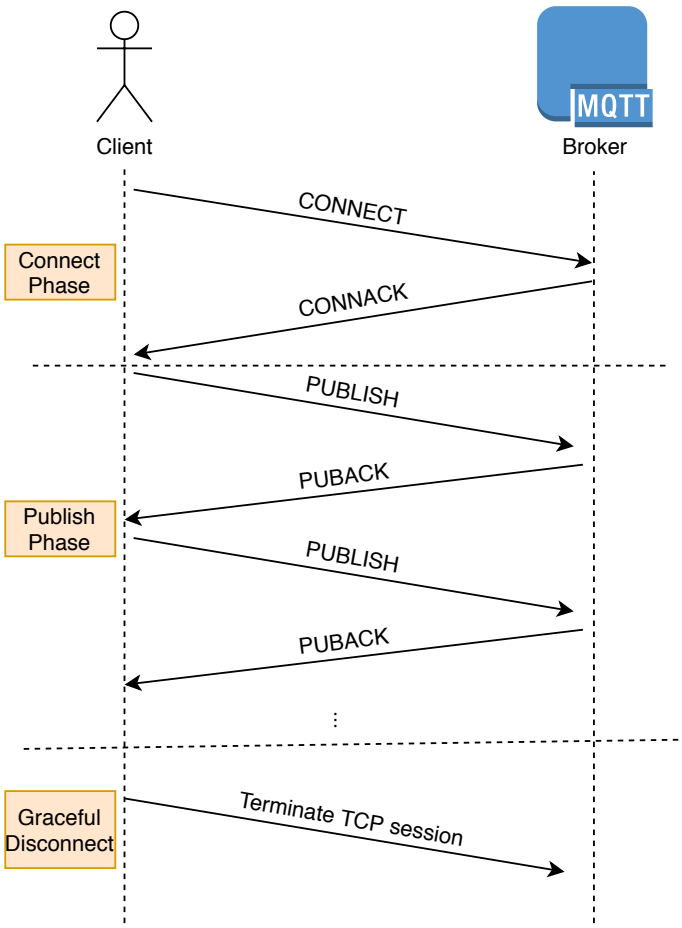

Figure 2.2: Publishing flow with MQTT

2.2.4 Subscribing

Subscribing flow is quite similar to Publishing, with some minor differences. Following figure 2.3, first and foremost, a connection needs to be established by instating standard TCP/TLS session and then sending CONNECT packet. The contents follow the same standard, i.e. containing information such as Client ID or even optional parameters in the form of "key: value" (particularly useful for this project).

After a successful connection, the client can proceed to send a request for subscription. Similar with PUBLISH packet, the client specifies the type of the packet in the variable header and then requested topic for subscription in the payload. The extra element is the QoS flag - Quality of Service. MQTT has three levels of QoS:

1. 0 - No response to PUBLISH messages
2. 1 - PUBLISH messages will be followed by PUBACK
3. 2 - More granular control over PUBLISH, with extra packets such as PUBREC (Publish Received), PUBREL (Publish Release) and PUBCOMP (Publish Complete).

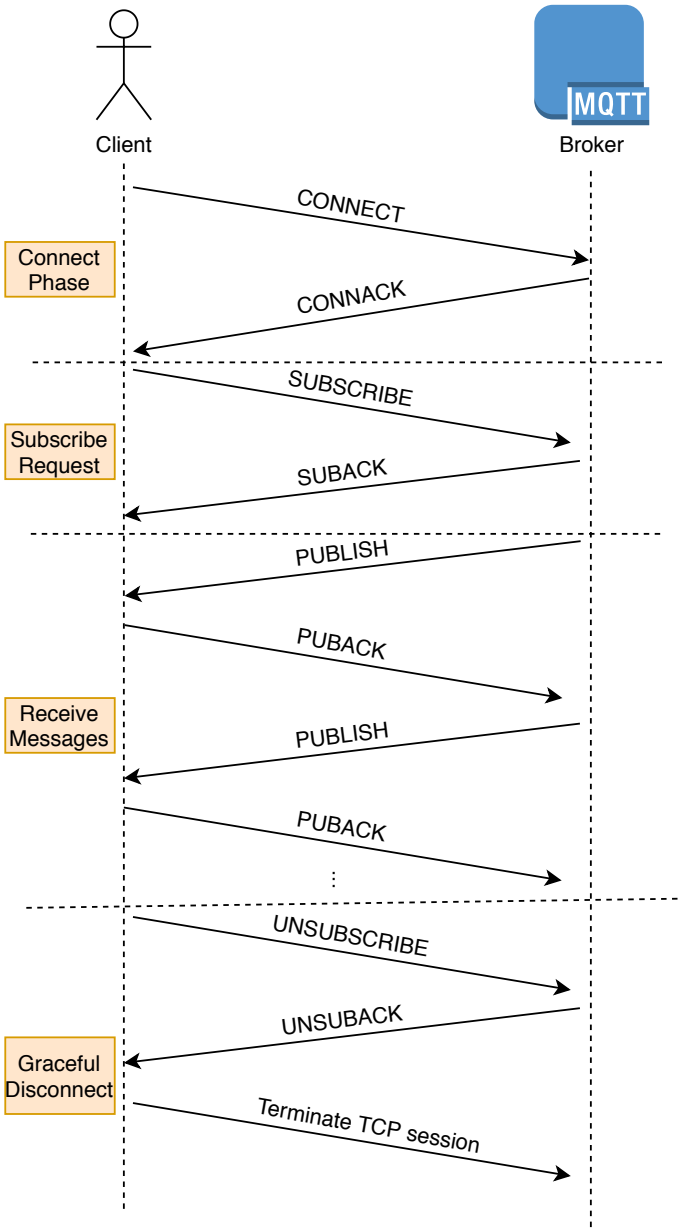

Figure 2.3: Subscribing flow with MQTT

Once the SUBSCRIBE message has been processed and approved by the broker, it will issue SUBACK message and remain connected to the client. From this point, any message that is published on the topic specified in SUBSCRIBE packet will be published (as PUBLISH packet) to every client currently subscribed to it. Of course, depending on requested QoS, the broker might then await for PUBACK message (or even issue other messages such as PUBREC, PUBREL, PUBCOM). The diagram demonstrates a simple exchange with QoS set to 1.

To close off MQTT, I also wanted to overview an example packet and dissect it byte by byte to demonstrate exactly what kind of information is included - this can be seen in table 2.1

1 Control field, specifies the type of the message (CONNECT, SUBSCRIBE etc.)

2 Remaining length of the message. Can be expanded to 2 bytes.

82	04	00	01	00	07	F	L	Y	T	R	A	P	00
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Table 2.1: Example SUBSCRIBE to topic FlyTrap packet

3-4 Packet ID

5-6 Payload length

7-13 Payload. Corresponding hex encoding of characters, replaced with actual characters for clarity

14 Requested QoS

2.3 Blockchain

Lots of concepts in this paper involve blockchain methodologies, which, by itself, is an expansive area. As part of this section, I will be only covering the most relevant topics necessary to understand the design choices taken within my project, but further reading is strongly encouraged.

2.3.1 Architecture

Blockchain often goes by its infamous name of simply overly complicated linked-list, and in fact, it is not very far away from being true. The concept was first introduced and popularised by Nakamoto et al. [24] in a paper introducing a highly controversial notion of digitalising and decentralising currency, by moving it into a structure called a blockchain. Blockchain network was meant to operate on a peer-to-peer basis, with different peers validating each other's transactions and holding a copy of the entire block. This removes the need for a central authority governing the currency (for example, central banks), by placing a copy of all records on every participant's computer - one problem remained, and that was trust. How do we trust other peers that they do not inject fraudulent transactions? However, before I answer this question, let us focus on figure 2.4, which outlines the difference between distributed and centralised ledgers. With the current economic model, usually there exist some central authority (in this example, a central bank) which is responsible for tracking, verifying and authorising all transactions between participants. Compare it with a decentralised ledger, where there is no such central entity. Instead, each participant verifying all transactions that happen between nodes. They no longer have to trust Central Bank to do their job currently, as they are free to confirm the authenticity of all transactions themselves. Nevertheless, again, we get back to the same question - how does the authentication happen?

2.3.2 Consensus Algorithms & Proof-of-Work

Before a participant can add their transaction ("block" from blockchain) to the public records ("chain" from blockchain), we need a cryptographically secure mean to verify whether this particular participant can add this block. Establishing trust between participants on a blockchain is often referred to as "consensus". Thus, several consensus algorithms exist. Currently, perhaps the most popular one, it is proof-of-work. In fact, PoW dates even before the paper by Satoshi Nakamoto, all the way back to Jakobsson and Juels [17]. It utilises one of the most critical properties of hash functions, that is pre-image resistance. The blockchain will offer a cryptographic puzzle to the participant willing to add a new block. This puzzle would be based on reversing a hash, i.e. a hash

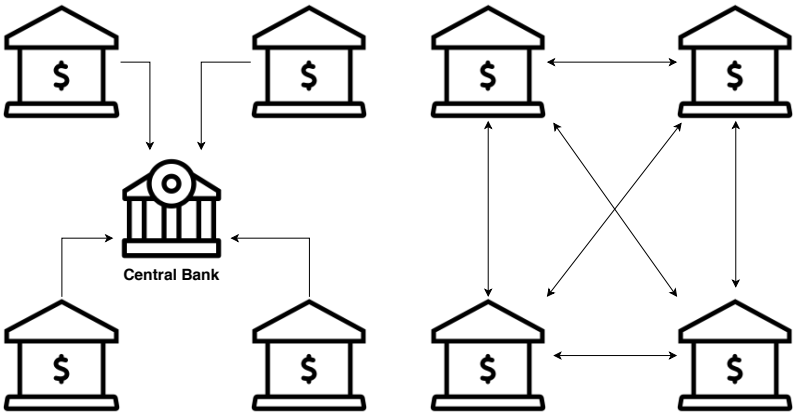

Figure 2.4: Centralised Ledger (on the left) vs Decentralised Ledger (on the right)

would be generated, and the participant would be tasked with reversing it, thus getting the original value. This "puzzle" is also often referred as mining a new block, that is, finding a value that after passing through specified hash function would produce expected output (also known as cracking hashes) - that is also part of the reason why modern mining requires much computational power.

Then, everyone starts a race towards reversing this hash. The first person to achieve the target is rewarded with a possibility to add new a block to the network (along with the found value). In the future, any peer can verify the authenticity by feeding the attached value through the hash function and verifying whether the obtained value matches the expected hash. All of it is possible since computing hashes is relatively fast and not a very computationally expensive operation. At the very end, when attaching the new block to the chain, the miner is usually rewarded with cryptocurrency, which can then later be exchanged with other participants.

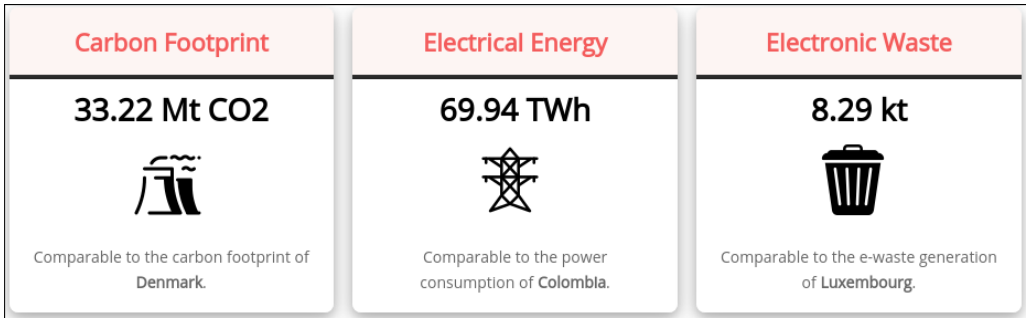

Figure 2.5: Annualized Total Footprint of Bitcoin network [16]

Of course, this approach has several downsides. First of all, all the computational power is effectively wasted to this cryptographic puzzle, with no real end-use - especially if you take part in the race to crack the next hash and someone ends up being faster than you - all your effort went for nothing. This was widely discussed by scientists [13], who currently point out a negative impact on the environment. As reported by portal Digiconomist [16], as of 2020, the annualised carbon footprint of Bitcoin network can be compared with the carbon footprint of the entire country of Denmark, with extra samples such as electrical consumption or electronic waste in figure 2.5

Another problem that Proof-of-Work algorithms create is a 51% attack. Nowadays, setting up your Bitcoin node and starting to mine is not very feasible since people with higher hash rates

(the speed at which a person can crack hashes) usually form organisations, that share this power amongst each other and then once they can crack the individual hash, reward each of the members with only a small portion. This might sound good for individuals since now they are guaranteed a payout (rather than risking taking part in the race and losing, winning nothing), but it effectively defeats the decentralised concept of blockchain. If one organisation holds more than 51% of the hash rate of the entire blockchain, it can start authorising fraudulent blocks and adding them to the chain. Since they hold the majority of the network's hash rate, nobody can defy them.

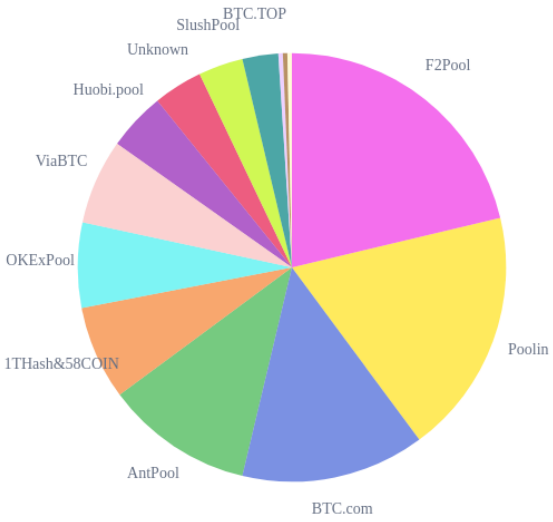

Figure 2.6: Summary of Mined Blocks as of 2020-03-30, per blockchain.com

Figure 2.6 shows the approximate split of total mined blocks in the past 48hrs since March 30th. Now, imagine a situation where organisations BTC.com, Proolin and F2Pool started collaborating, taking over 51% of the market. They would be able to add arbitrary blocks, self-verifying them - and since they hold the majority, nobody could oppose it.

2.3.3 Proof-of-Stake

A slightly different approach to verifying the transaction is called proof-of-stake, first introduced by [19] - which does not involve cryptographic puzzles nor requires high computational power and is all based on some pre-defined amount of cryptocurrency that is being put on hold, while delegates are selected. Every participant can bet any amount of crypto - which is returned to them after the validator is selected. The process can be outlined in the following steps:

1. Participants define their stake.

2. Network selects one participant that is going to be acting as a validator for the next block. The higher the stake, the higher the chance of getting selected. Losers get their stake back.

3. validator goes ahead and verifies the next block gets added to the chain, there is no block reward, although they receive network fees for the transaction.

4. After a couple of days (once other participants verify the transaction was not fraudulent), the stake is released and goes back to the validator.

This partially eliminates the issue of 51% attack mentioned above. First, because hash rate nor computational power no longer matters (and thus forming organisations loses the point) and secondly, even if the fraudster held more than half of the entire markets crypto, they would still be at risk of losing the stake, as their chance of getting selected as a validator is not 100%.

Sadly, this approach also is not free of any issues. Contrary to what I mentioned above, it creates a bias towards participant with more significant wealth, which can put more value on stake. This might create situations where rich get richer - though there is always a non-zero chance of getting selected. Furthermore, while they might not have malicious intents, they would be at a higher chance of getting selected as validator and thus collecting more network fees.

As this field is still expanding, more work is published, suggesting refined approaches. At the current day, both Bitcoin and Ethereum use Proof-of-Work, though the latter aims to move towards Proof-of-Stake in the future iterations [27].

2.3.4 Proof-of-Authority

However, what if we do not care about full decentralisation and want to avoid extra operational costs through proof-of-work or proof-of-stake algorithms? A simpler solution, called proof-of-authority [25] can also be used. This approach offers no rewards for adding new blocks to the chain, so it is not used in public blockchains. The validators are pre-selected and are responsible for vetting new blocks. This has more uses in situations where data does not have to remain secret, and we do not mind the lack of decentralisation. In fact, if the validators become compromised, they would be able to start allowing malicious blocks.

2.3.5 Ethereum

Proof-of-Work proved itself to be a tremendous waste of energy and resources, with Bitcoin using it solely for authorising the transaction and nothing beyond it. That particular period was also a time when a lot of different currencies started showing up, as Bitcoin's source code was open, everybody was allowed to host their network. Ethereum was one of them, but it was also the first to introduce a concept known as smart-contracts - a way to put the proof-of-work energy to some use (though still a lot of was wasted), introduced in 2015 by Buterin et al. [5]. The network also gave birth to so-called decentralised applications (or Dapps for short), through smart-contracts. Smart-contract can be understood as pieces of code which can get executed on the blockchain, written in a specialised language called Solidity inside EVM (Ethereum Virtual Machine). The transactions were no longer limited to the information about transferring currency between accounts, but also could execute code and act as a persistent database, which could not be altered by anyone and change history was publicly available.

2.4 Related Work

In this section, I would like to point towards recent research that also had in mind imporving the security of IoT by looking into combining it with Blockchain platforms. Below, I describe two papers - both concluding with positive aspect of such approaches.

2.4.1 IoT, Hyperledger and GA

Attempts at combining IoT authorisation with blockchain has been made in the past. One of the examples is a recent work by scientists from Khon Kaen University in Thailand. In that paper [20],

researchers look into Authorization Architecture for IoT (using MQTT broker as an intermediary entity). They are arguing about the benefits of combining any solutions for low-power devices and distributed architecture, which ultimately enables much better scalability and removes the single point of failure.

They are also utilising Hyperledger Fabric - another blockchain-based ledger. Compared to Ethereum, Hyperledger [6] is used mostly for Business-to-Business scenarios, as it does not feature any reward for mining, i.e. adding extra blocks to the chain. Transparency is also limited, as the information is no longer placed on a publicly available platform but rather depends on trusting the nodes connect to the network although I will spend some more time discussing differences in implementation and discussion chapters of this paper.

Moreover, the focus of that paper is at finding optimised consensus algorithm, such that any latency caused by permission lookup is minimised. Scientists suggest using Genetic Algorithms to compose Optimal Consensus. Their experiments were executed on Kafta MQTT[29]. Thai researchers were able to achieve the performance of their solution called GA Kafka improved by 69.43% compared to standard Kafka.

Although this paper does not take into consideration other parts of the AAA framework, focusing solely on Authorisation. It has no mention of authenticating connecting clients or making them accountable by keeping audit crumbs of the most sensitive operations conducted on the chain. In my work, I will also be less focus on the performance of the blockchain network itself, leaving this down to the blockchain itself. As mentioned in the previous section, Ethereum has had some rapid movements in terms of improving their consensus algorithms and moving away from Proof-of-Work instead aiming to implement Proof-of-Stake.

2.4.2 IoT Communication using Blockchain

In a related paper, scientists from Insitut Teknologi Bandung [12] offer a solution of replacing MQTT protocol with smart contracts, arguing increased level of security and resilience to the outside threats. The approach is somewhat different, as they are comparing the MQTT protocol with the option of using Ethereum for communication. This differs from the architecture proposed in this paper, as FlyTrap still operates through MQTT and requires a broker to function, i.e. it's only a middleware with regards to the vanilla implementation - whereas the Fakhri et al. removes MQTT and instead moves the communication to Ethereum.

The evaluation consisted of performing attacks on the networks. They also experimented with the SHA-256 [14] algorithm commonly used for producing message hashes and compared its avalananche effect with Keccak-256 used by Ethereum. Research is concluded with findings that Ethereum's algorithms are more resilient than direct hashing + encryption approach and suggests further study in combining blockchain with IoT technologies, suggesting great security gains.

Chapter 3

Requirements

In this chapter, I will outline base requirements for the project along with sample stories that would later dictate the workflow. In the second part, I will also include an overview of the architecture proposed for the system, correlating each element with relevant requirement and explaining how they would address the use-cases.

3.1 Requirements

3.1.1 User stories

1. As a government regulator, I would like to overview access history to specific MQTT topics, to make sure the data is handled in GDPR-compliant manner.

2. As a government regulator, I would like to verify why / when / who accessed given resource at a specific time, such that I can issue fines for potential non-compliance and inspect data breaches.

3. As a topic owner, I would like to restrict people that can publish/subscribe to them, to maintain their confidentiality.

4. As a topic owner, I would like to collect payments from people willing to access my data.

5. As a topic owner, I would like to block access to my information from requests coming outside the requested country, to comply with GDPR requirements.

6. As a broker owner, I would like to collect payments from people willing to publish their data on my system, to keep the system profitable.

7. As a broker owner, I would like to secure a distributed network of brokers (with varied implementations), to increase the system's availability.

8. As a broker owner, I would like to block access to the system to malicious clients performing denial-of-service attacks, to avoid system downtime.

9. As a data consumer, I would like to publish/subscribe my messages on low-power devices, such that I can utilise my IoT sensors.

10. As a data consumer, I would like to access the broker from over a hundred parallel sensors, each publishing data independently.

3.1.2 Use-case Scenarios

Below I have listed a collection of stories which relate to real-life scenarios (though they are not real, any similarities are coincidental) and present various problems faced in different areas - both in industrial and commercial environments. FlyTrap is looking to address the issues surfaced in those short stories. In the Evaluation chapter of this paper, each of them will be referred to and tested on whether it is indeed helpful and how the discussed solution solves the problems.

3.1.2.1 Scenario #1: Air Quality study in the UK

Scenario: Robert is working as a Research Fellow at a University located in Manchester. The research aims at issuing air quality IoT sensors to staff across University, intending to capture information such as pollution or carbon dioxide level to analyse contents of air in the state. Each sensor is issued to an individual taking part in an experiment (e.g. member of staff, lecturer, PhD student), which is based in a specific room on campus. Robert needs to be able to track the inventory, and thus every sensor must be traceable down to a person.

The budget allows to issue up to 1000 sensors, and Robert would like to use MQTT broker to receive the data from the IoT devices. Additionally, he would like to share the dataset with researches across the country; thus, he makes the MQTT broker public. As per GDPR, such data, containing full name, office location and detailed temperature readings, is fully protected and needs to adhere to various governance requirements within the European Union. This information should also be stored only within the jurisdiction, that is, only within EU, to ensure enforceability of GDPR. Robert needs to make sure that only researchers that are located in Europe can access the date and everyone else (for example, people from the USA or Russia) will be denied access.

What problem is addressed here: Data containing personal information of European citizens needs to be handled in GDPR compliant way. This cannot be ensured if the information is accessible outside EU, as there would be no jurisdiction or enforceability once the data leaves the region. The person responsible for data storage needs to ensure that no people outside EU can access it.

3.1.2.2 Scenario #2: Data breach in an oil drilling facility

Scenario: Bob is a Chief Information Technology in a company Chell handling processing of oil and gas in Scotland. Bob's company also contracts many smaller companies which provide staffing and direct drilling services. Many sensors are used in the company, which are responsible for collecting data such as air pressure, humidity, occupancy on drilling platform or temperature. Those IoT sensors are utilising MQTT Broker, which is restricted only to authorised Chell employees.

Unfortunately, due to unrelated reasons, access to the broker has been compromised and thus allowing third parties to peek into the data flowing through potentially. Bob is approached by Judy, who works with the team responsible to trace the severity of leakage and determine the potential scope & damages.. Judy asks Bob to outline who might have had access to the leaked information and what the leaked information contained. Judy also instructs Bob to inform all people and contractors that might have been affected by the breach.

What problem is addressed here: Companies that use MQTT brokers to store information which are confidential and contain trade secrets of high commercial value want to determine the range

of leak that happened. This is part of disaster recovery procedure and is important for business continuity; i.e. determine how much of data has leaked and what kind of information was exposed to malicious actors.

3.1.2.3 Scenario #3: Unsatisfied Customer

Scenario: Mary recently purchased a smart assistant, which comes with several smart sensors to be placed around the house. Unfortunately, Mary decided to return all the sensors and cease further usage, she reaches out to Moogle's representative - Matt.

Matt knows that Moogle is using MQTT brokers to connect their smart sensors and then use the phone app to issue commands back to them through the broker. Although the phone app is not the only piece of software that has access to the data from the smart sensors. Analytics teams also consume those in order to help Moogle create better products. Those servers have their own data storage capabilities and replicate the data that they consume. Matt is now tasked with identifying which internal analysis services might have accessed Mary's sensors in order to erase this information since it is on of GDPR requirements, also called "right to be forgotten".

What problem is addressed here: Again, GDPR comes into action here, in particular Article 17 - Right to erasure. Moogle needs to permanently erase all trail coming from Mary's sensors, that includes any analytics datasets. Since those services are using MQTT brokers, there is no access trail and without proper infrastructure, impossible to go in the past and track which services were accessing the data.

3.1.2.4 Scenario #4: Monetization of data

Scenario: Frank owns several farm fields in Scottish Highlands. As part of the ongoing digitalisation, he decided to purchase some IoT sensors to learn more about the environment his farms are located in. Some of the sensors include specialised soil acidity meters capable of measuring pH or determine how moist it is.

He decided to connect all sensors to one MQTT broker for ease of access and started to consume the information through provided mobile application. As the equipment was quite expensive, Frank would like to also sell access to the sensors to the interested parties, such as other land owners or University researchers looking to learn more about the soil quality in the area. No personal information is included, though the data is highly dependant on the location (which currently only Frank can access to) and thus might be also useful for other people.

What problem is addressed here: Selling data is a very lucrative business, especially if the data is valuable and has potential audience. MQTT does not have provisions to accommodate such requirements - anything like that would need to either be stored in some persistent location (and then processed for commercial purposes) or routed through extra layer of verification.

3.1.2.5 Scenario #5: Securing access to the broker

Bob works for a start-up company contracted by the local council to deploy smart sensors across the city to measure various statistics, such as air quality, traffic intensity, parking spot occupancy. All of those sensors are connected to the MQTT broker, which later can also be accessed by third

parties - providing they have a legitimate and approved reason. The MQTT broker is accessible by people that might not have overlapping permissions, for example company A can access only data set Y and company B can access only data set X - both X & Y are published on the same MQTT broker.

Bob is aware of the vanilla implementation of username/password authentication for MQTT brokers, but is concerned about the insufficient security in case of leakage. He also already is using Ethereum within the company to hold internal data, so can issue more wallets - if necessary.

What problem is addressed here: Security capabilities of vanilla MQTT Brokers can be found lackluster and not sufficient for many needs. Using symmetric password authentication can be very damaging if the said password leaks to third parties. For people that are already using related blockchain technologies, the transition to FlyTrap would be even simpler, as it can make use of any and every wallet compatible with ETH.

3.1.3 Functional Requirements

The user stories can then be further formulated into the following functional requirements:

(FR1) The system will provide an interface to manage access to the topics along with inspecting the audit trails.

(FR2) The system can connect to any Ethereum node, be it a public endpoint or a locally running, closed network. This will provide the flexibility of either using transparent and with 100% uptime resource or a closed node with reduced costs.

(FR3) The system should provide a way to collect payments in ETH from clients attempting to gain access to relevant resources. This payment would then in the process, be transferred to the resource owner's Ethereum wallet.

(FR4) The system should offer an option to specify an exact amount of ETH required to publish or subscribe - with the possibility of separating the costs and also setting the cost to 0 (=free).

(FR5) The system should be capable of fending of primitive denial-of-service attacks by blocking continuous, failed attempts to connect.

(FR6) The operations performed by clients will be of limited complexity, such that they can be executed on devices with limited computational power.

(FR7) The system can answer crucial GDPR questions, such as who accessed given resource, why did they have access, when they accessed it and what exactly was accessed.

(FR8) The system should offer an option to restrict the client's country that can access the resource, which will be verified using GeoIP lookup, as various countries have various data protection laws.

3.1.4 Non-functional Requirements

In addition to the functional, it is also vital to mention the following non-functional requirements, as the system is intended for end-users (potentially non-technical) and due to incorporation with blockchain can introduce performance overhead.

(NFR1) The system should provide **an overhead of no more than 2 seconds cumulative** per MQTT session. This is important, as the intention is to provide an add-on on top of the existing MQTT brokers. This might further compromise the current efficiency, so the system should aim to minimise the added latency

(NFR2) The system should be agnostic of the used MQTT broker, as long as the broker **fully implements MQTT v5.0 standard**. As pointed out earlier, there is a variety of brokers available to use, such as Mosquitto or Moquette. FlyTrap should not rely on the implementation of a broker, but rather only on the standard utilised.

(NFR3) The system should be capable of extending any MQTT broker with **Authentication, Authorisation, Accountability** framework. This is to ensure that data can only be accessed by authenticated entities, which are authorised to access requested resources - and in case of a breach or other disaster, keep them accountable to their actions.

(NFR4) The system should only be based on **Free and Open-Source Software**. Since the ultimate aim is to provide increase security, keeping the source open would allow any potential users to inspect its operation. Furthermore, third party security audits can happen without the system owner's intervention.

(NFR5) The system should be capable to run inside **virtualised container**, to ensure that it's platform agnostic.

Chapter 4

Design

In this chapter, I will talk in-depth the elements that I created during this project and how they interconnect. You can expect a design charts showcasing high-level overview, alongside dividing it onto smaller subsections - each independent and capable of running on its own. I will also explain novel approaches such as verifying the authenticity of the connecting clients, data model stored on a blockchain or how the connection is secured from man-in-the-middle attacks through TLS encryption.

4.1 Architecture

This section includes the diagram of them system and two sample workflows demonstrating how packets are flowing through the framework.

4.1.1 Overview

Figure 4.1 presents an overview of the system, decomposing it onto four layers, each responsible for a different part of the framework. It also demonstrates how those layers are coupled and the direction of data flowing between them.

To overview, we can distinguish four layers:

Consumer Layer - layer responsible for interacting with end-devices. To them, FlyTrap should be indistinguishable from normal MQTT broker and thus accepts/responds with MQTT v5.0 compliant TCP/TLS packets. In order to compute the signature and attach it to

Consumer Layer - layer where FlyTrap acts like a client for MQTT Broker. Similar to Consumer Layer, all packets sent by FlyTrap need to be compliant with MQTT standard in order to receive valid responses from the broker. In this situation, used MQTT broker is not relevant - as long as it implements the standard.

Blockchain Layer - layer in which FlyTrap performs communication with the Ethereum node through a separate CLI. FlyTrap is capable of either reading the past contracts/transactions or submitting new ones. That is also the only way to amend the state of the blockchain - given the private master key has remained secret.

Presentation Layer - layer used by FlyTrap's end-users which allows to overview the state of the blockchain in a user-friendly way, easily extracting most relevant information such as recent access changes or audit trail for major operations on the chain.

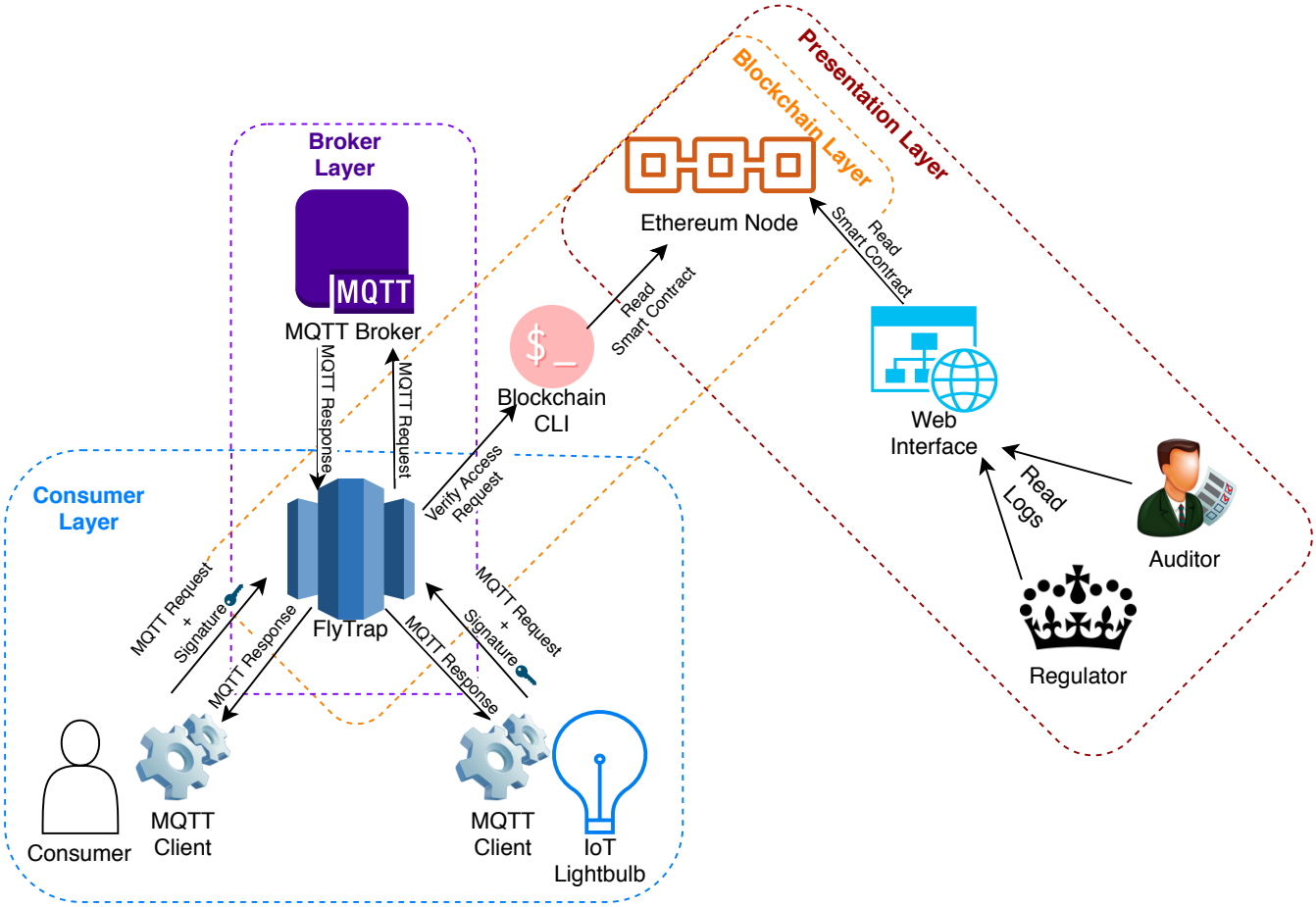

Figure 4.1: FlyTrap high-level architecture overview

4.1.2 Sample successful PUBLISH workflow

To provide further context, figure 4.2 provides an example process in which a client wants to publish a message to the broker and is successful in doing so. The diagram shows step-by-step logic performed at each stage of the connection, until client receives the response. I'll annotate each step with either **BL** - Blockchain Layer, **CL** - Consumer Layer or **ML** - MQTT Broker Layer to signify on which layer this step is taking place on.

Explanation of each step:

0. Marked as optional since each client can accept a pre-computed signature, which can be loaded onto the device; this can be helpful for situations where there is not enough computational power for calculations. (**CL**)

1. client sends CONNECT packet, including signature + public key in the optional fields of MQTT message. (**CL**)

2. FlyTrap will extract the signature from the optional field and then verify its integrity. It will also check if the client has not been attempting many unsuccessful connections. Finally, FlyTrap will respond with CONNACK, signalling to the client that it may now submit relevant payload packets. If the integrity check has failed, CONNACK would also have a flipped flag indicating rejected connection and cease further communication. (**CL**)

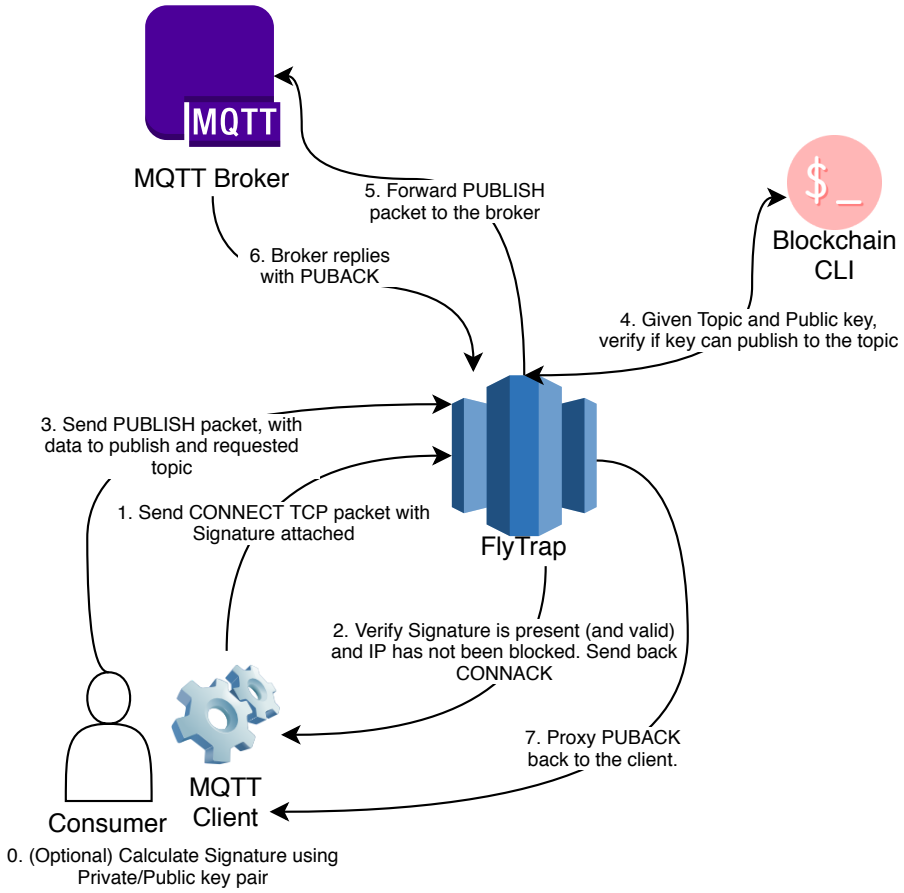

Figure 4.2: Example successful workflow to publish a message on the broker through FlyTrap

3. client will now forward the relevant PUBLISH/SUBSCRIBE packet to FlyTrap (as it still believes that it is a regular MQTT broker). (**CL**)

4. FlyTrap will now extract requested topic from the MQTT packet and communicate with the blockchain, presenting Public Key and requested topic to verify whether data can be accessed. For this example, the access check was successful. (**BL**)

5. FlyTrap proxies (unchanged) PUBLISH packet to the actual MQTT broker. (**ML**)

6. MQTT broker will now respond with PUBACK, indicating successful PUBLISH. (**ML**)

7. Finally, FlyTrap will proxy the same PUBACK packet back to the initial client to let them know that the operation was successful - and at the end, gracefully terminate the connection. (**CL**)

4.1.3 Sample failed PUBLISH workflow

Operation similar as in the previous section with the difference being that this time connection is not allowed, as the presented Public Key is not allowed to publish information on the given topic. For the brevity sake, I will skip explaining steps 0-3 - as they are identical as with the successful scenario. I will also use the same notation to signify which layer is responsible for this operation.

5. Having verified authenticity of Public Key, FlyTrap again tries to verify with the Smart

Contract whether the client can access the topic. This time though, the response is negative, and the client is not allowed to publish on the requested topic.

6. FlyTrap will send PUBACK back to the client, setting reason code[1] to "Not authorised", at the same time terminating the connection with the client. To client, the situation is identical as with providing invalid username/password for a vanilla MQTT broker.

7. framework will verify with the cached values to check if the originating IP has not exceeded the maximum number of allowed tries. If it did, it could place it on a blacklist, and every subsequent connection will be denied for the specified time. In this example, that is 5 minutes. This step is optional

8. If the ban happens, FlyTrap will also register a new transaction on the blockchain to persistently log this event to check potential attack spikes or generate reports w.r.t. captured data. This step is also optional, as it depends on the previous one.

Note: as you can see, the MQTT Broker is never connected to nor contacted with the potentially malicious message. FlyTrap rejects the message and forbids the connection.

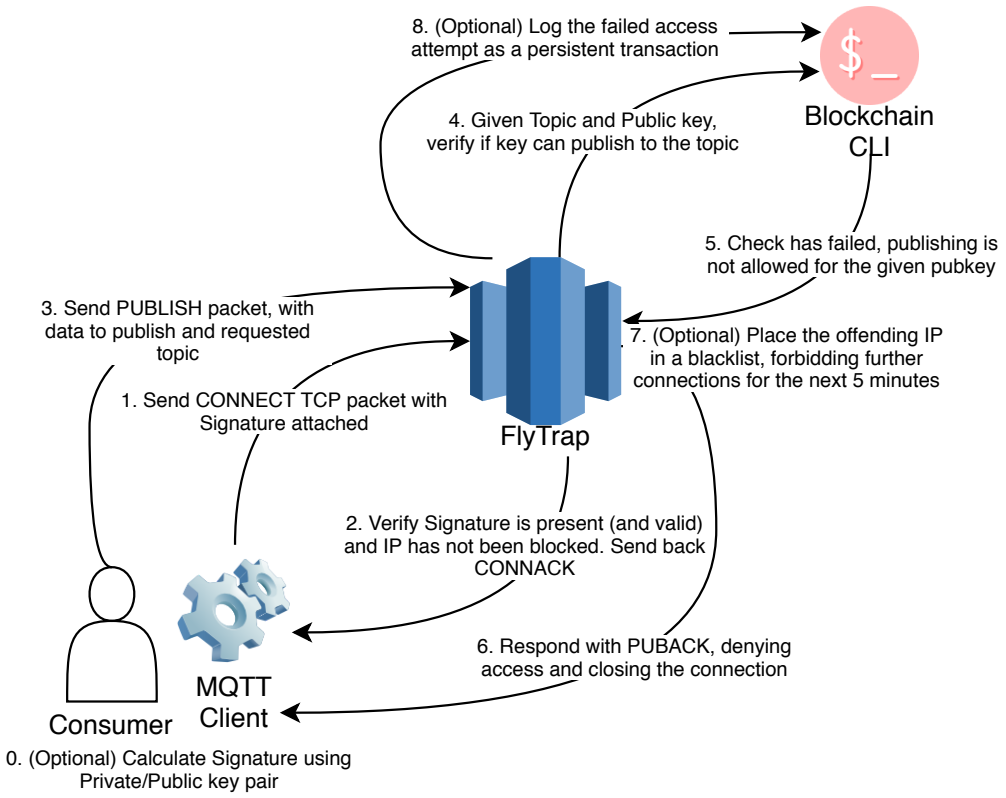

Figure 4.3: Example failed workflow to publish a message on the broker through FlyTrap

4.2 Consumer Layer

First and foremost, the consumer layer. This part of the framework handles all connections between clients attempting to PUBLISH or SUBSCRIBE to data. To them, FlyTrap should be indistinguishable from a vanilla MQTT broker, which should be accepting all regular MQTT payloads.

[1] https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901124

Though since every MQTT client produced in this project can connect with every MQTT Broker, a specific client is required for connection with FlyTrap - which is further explained in the section below. Moreover, an approach of identifying whether the client holds a Private Key for the presented Public Key is also needed and explained.

4.2.1 MQTT Client

As mentioned above, FlyTrap makes use of the introduced in MQTT v5.0 User Properties[2], allowing clients to include key-value pairs in the MQTT packets, which then can be utilised by the brokers (or other middleware). That is also where the signature and public key is being placed by the client when attempting connection with FlyTrap - and that is also the need for a custom client since regular clients are not capable of producing highly specialised signatures to connect with Ethereum blockchain.

It is important to point out, that the client is only slightly altered to provide (and compute if needed) the signature from public/private key pair. It is not a central system of the framework, as any client capable of setting User Properties for MQTT message would be sufficient, though, for the purposes of this dissertation, custom implementation has also been designed and included.

4.2.2 Secure Proxy

In order to enable FlyTrap to make decisions on whether the requests for publishing or subscribing should be accepted or denied, a secure proxy needs to be established between the clients and the MQTT Broker. As the communication between the broker and the consumers happens on Transport Layer, it is possible to insert a middleman who would be capable of inspecting the packets flowing through, dissecting it for relevant information and finally make a decision about their future journey - all without the client ever knowing that someone has intercepted the connection. Figure 4.4 demonstrates all 3 possibilities when client attempts connection to a broker. In the

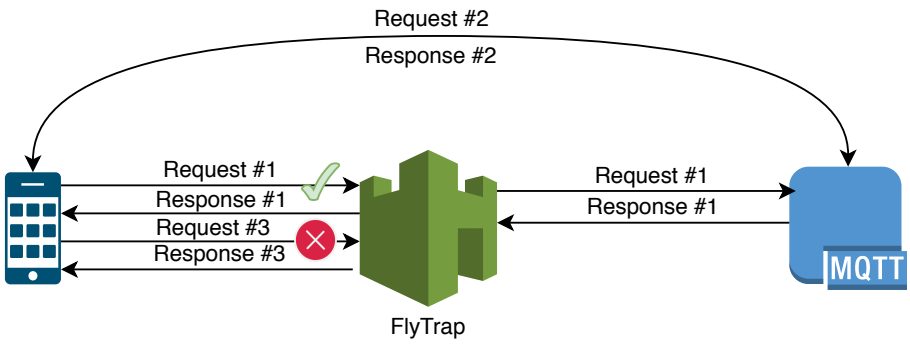

Figure 4.4: FlyTrap acting as a proxy

Request #1, FlyTrap will dissect the packet and confirm that the phone indeed can be allowed to access specific topic and then start bidirectional proxy with the broker, passing the TCP packets between two. Request #2 shows that the same packet can be used for vanilla MQTT Broker without FlyTrap, thus decoupling the client and secure proxy, as the former can be used without the need to change the latter. Finally, for the third request, it is found that the client cannot access the requested resource and will be presented with CONACK response, with access denied flag set,

[2]https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901068

terminating the connection. Though, in order to make such decisions, the proxy needs to inspect the contents of the packets.

Although this solution enough will not be sufficient, as quickly as FlyTrap can tap into the connection, the same can be assumed for potential malicious actors, which could be listening on the flowing through packets. The solution will support an extension to standard TCP - Transport Layer Security, or TLS for short, responsible for encrypting the TCP packets, significantly reducing the threat of man-in-the-middle attacks.

TLS sessions can be summarised in the following steps:

1. Initiate standard TCP session
2. ClientHello with client's cypher capabilities
3. ServerHello and exchange of the cypher suite, along with server's certificate
4. Key exchange and change of cypher spec
5. Encrypted session starts

It is vital to point out, that due to step 3 requiring server's certificate, FlyTrap will need to either obtain a copy of broker's certificates or generate a new pair, ensuring that the connecting clients will trust it. Though secure TLS connection remains optional, as it is understandable that sometimes enhanced security might cause undesirable performance losses or the MQTT broker simply does not support TLS connections - TLS will be configurable via command-line arguments.

For every new connection, a new thread (or, goroutine) is spawned which has its own context and is separated from others.

4.2.3 Authenticity of public keys

Public Keys from the Ethereum wallet are used as an identifier when determining whether a client can access a given resource or not. Though it only handles a part of the problem. Public Keys, by definition, are public, meaning that anyone could impersonate legitimate holders of the public/private key. This calls for an approach similar to the Certificate Authority problem when attempting encrypted HTTPS connections. Unfortunately, FlyTrap cannot expect every IoT device to have its own set of certificates, which would then need to be trusted by the framework in order to become recognisable - as those devices are often of limited storage and power.

In order to solve the problem of establishing whether the person presenting a public key also holds a corresponding key, signatures are used. Below you can see two figures, each outlining client-side and FlyTrap side of the signing/verification process.

Figure 4.5 shows how client - in possession of public/private key pair produces a signature which then is attached to the final MQTT CONNECT packet. First, it hashes the public key using Keccak-256 algorithm [3] (commonly used in Ethereum, e.g. for block hashes), then it uses the private key to sign this hash and attaches obtained signature to the user properties part of the MQTT CONNECT message. Plain-text version of the public key is also attached in another field. Ethereum signatures are created by signing arbitrary bytes through a generated private key - which then can only be verified using the corresponding public key. It is also possible to extract signed bytes from the same signature in the process.

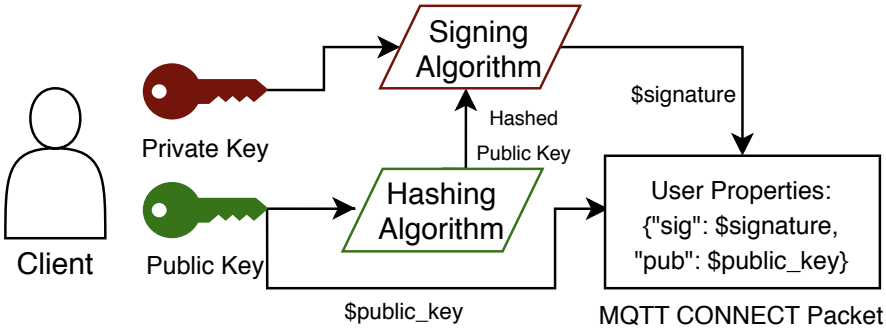

Figure 4.5: Client signing Public Key and attaching it to the message

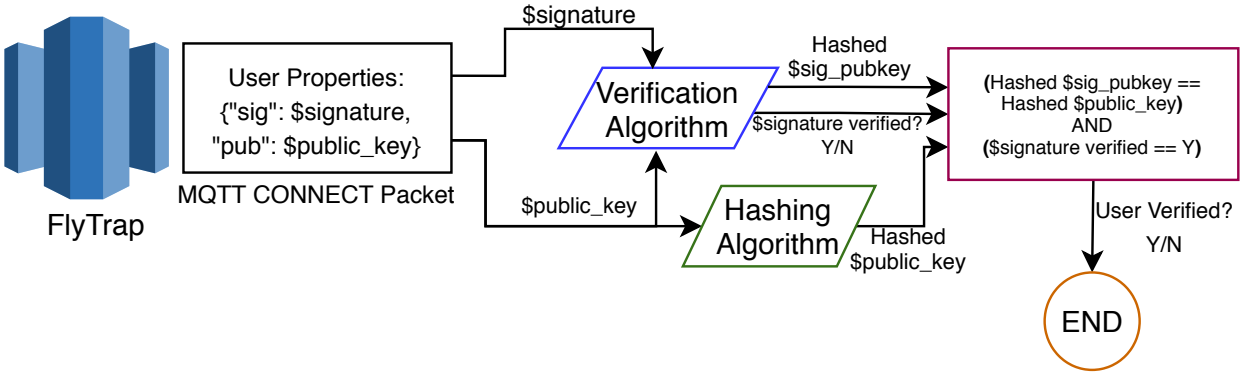

Figure 4.6: FlyTrap verifying the signature

Then, as per figure 4.6, FlyTrap receives the CONNECT packet and extracts both the signature & public key from the message. As the corresponding private key for the public key was used to produce the signature, framework verifies this through Verification Algorithm. The output is a binary yes/no value - determining whether the public key was used to create the signature - along with the signed value (in this case, hashed public key) by the client. Then, framework can verify whether both signature was indeed created with the private key corresponding to the attached public key and (by hashing it again) compare the extracted value with the attached public key. This gives a definite answer that the person presenting the public key also holds the private key and thus is successfully authenticated.

For Ethereum, both Verification Algorithm and Signing Algorithm is part of Elliptic Curve Digital Signature Algorithm [18], where a public key has exactly 160 bits (which, coincidentally, is also used for ETH addresses). Hashing Algorithm - as mentioned above - is Keccak-256.

4.2.4 Protecting from brute-force attacks

Some of the failed attempts can result in a persistent log to the blockchain, which often implicates costs - primarily if FlyTrap is operating on a public blockchain. This can open a door for malicious actors attempting to drain the framework from available funds by logging many operations in a short period. Distributed Denial of Service (DDoS) attacks can also occur and overload the server, which - depending on hardware - could only handle a limited number of simultaneous connections.

To combat those problems, framework will track any failed authentication attempts in an internal dictionary, mapping IP address to the number of failed attempts. This dictionary then will be consulted whenever a new connection is initiated. If it is, then the TCP link will be shut

down, and the client informed that it is currently blacklisted. Though to give the benefit of a doubt, there is a grace period of 2 prior failed attempts before a timed ban is applied. Whenever failed authentication occurs, the counter is increased by one - and if that count increases 3, the connection is terminated and dictionary updated with time 5 minutes in the future - that is the earliest time given IP can attempt the connection again.

4.3 Broker Layer

This layer is where the communication with the actual MQTT Broker occurs. Typically, it would be desirable to place both FlyTrap and the broker (e.g. Mosquitto) on the same machine (or at least the same network) to minimise the latency - though it is not necessary. Depending on the outcome of the authorisation from the Client Layer, every packet from the client is forwarded to the broker, and every packet from the broker sent back to the client. Since each connection is an individual thread, it also maintains information such as originating (& destination) address and port - and this information will persist as long as the original connection Client <=> FlyTrap remains open - which will only be terminated if client times out, requests disconnection or for any reason authentication to FlyTrap fails.

Similar to the section discussing encrypted connection, FlyTrap is capable of either connecting with TLS-capable brokers or via plain TCP if desired (though keeping the security implications in mind, as everyone intercepting the connection would be able to read the payload).

4.4 Blockchain Layer

In this layer, communication with the Ethereum node occurs. As described in the architecture section, FlyTrap can either read or write new data onto the chain. FlyTrap should be allocated its own smart contract containing chain code capable of verifying connecting clients and relevant data structures.

4.4.1 Data model

The root of all communication with blockchain is a smart contract that contains all chain code responsible for retrieving and storing information required by FlyTrap. Each organisation or entity willing to use FlyTrap should configure and deploy a new contract, which would be tied to a singular owner (i.e., a private key). Ethereum's chain code execution can be limited to only particular set of addresses, here most of the sensitive operations (such as adding new publishers or subscribers) are restricted to either an owner or payable ETH (set by the owner) - if the requestor is not an owner.

Then, each contract contains a single variable called "topics" which is a mapping (dictionary structure in Solidity, a programming language for ETH) from a string (name of the topic) to structure "Topic" where the metadata can be located, such as control lists. Every person can create their own topic on the contract, of which they would become the owner - this operation can be payable, if set so by the contract's owner, meaning topic creators would need to pay a fee. Figure 4.7 shows all fields used in FlyTrap's chain code placed on the blockchain, to explain further the usage and purpose of each of the fields:

isValue - in Solidity, it is not possible to determine whether the given key exists in the mapping, as every value points to some arbitrary address space. An extra field helps to mitigate this

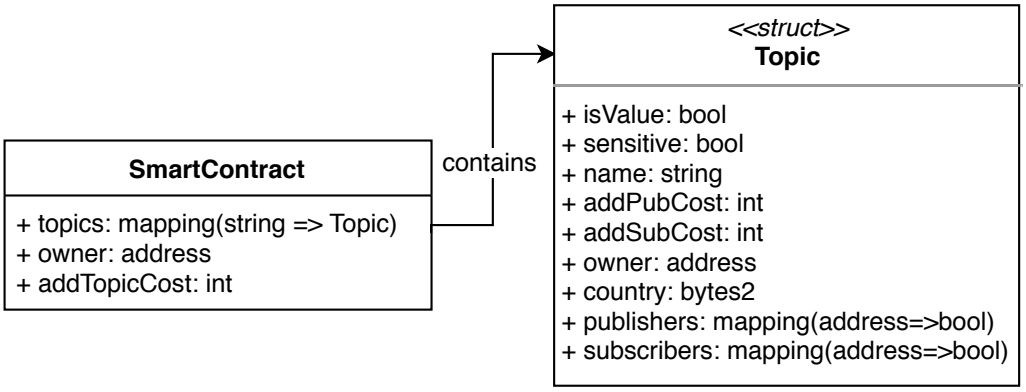

Figure 4.7: Topic structure stored on blockchain

issue, e.g. to avoid overwriting existing topics with new data.

sensitive - a bool flag marking given topic as sensitive which enhances it with extra data reporting functionality outlined in section 4.4.2.

name - a string storing topic's name, as used by the clients on the broker. It is also used as the key in the topics mapping.

addPubCost - integer, allowing to set a price on adding new publishers to the given topic. Then, the payment will be transferred to the topic's owner.

addSubCost - integer, similar as above, but for publishing.

owner - ETH public address of the person that created this topic.

country - 2-letter encoding of country to which access should be limited to for all subscribers/publishers.

publishers - mapping from the address of a person to a true/false bool value to determine whether the given public key can publish to this topic. It is a way of creating dynamic lists in Solidity, while maintaining $O(1)$ lookup time.

subscribers - same as above, but for subscribers.

Apart from base structure holding information inside the contract, events are also utilised. Event is a special structure used in Solidity, which attaches itself to the transaction log. Meaning that it is possible to append information such as user-specified reason or timestamps to all operations. This is then encoded in Application Binary Interface (ABI)[3] JSON and sent along with the transaction. Though, it is also possible to emit empty events solely for the logging purposes [8]. Figure 4.8 outlines what each event consists of:

source - ETH address of the entity that caused the log to emit. E.g. for adding new topics or modifying topic's properties (such as subscribers) it is the initiators request. For system-caused logs (e.g. report summaries), it would be the contract's address.

[3] Encoding type used in Solidity: https://solidity.readthedocs.io/en/latest/abi-spec.html

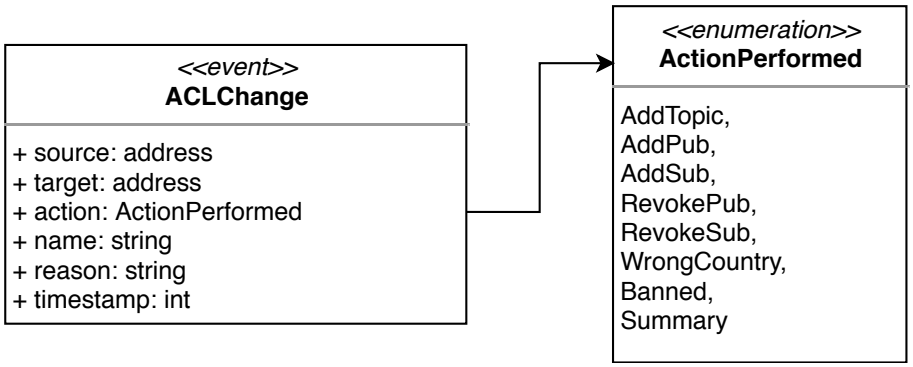

Figure 4.8: Event structure as stored in transaction log

target - ETH address of the entity affected by the logged action. E.g. for revoking publishers/subscribers, it would be the person that's getting removed.

action - enumeration, which defines the exact reason for the emitting the log, this can be one of the following values:

AddTopic - when a new topic is created on the contract

AddPub - when a new person is added as a publisher to the topic

AddSub - when a new person is added as a subscriber to the topic

RevokePub - when a person is removed from authorised publishers in the topic

RevokeSub - when a person is removed from authorised subscribers in the topic

WrongCountry - when a connection was attempted from an IP located in country that's not permitted

Banned - when a person failed to authorise when connecting for three times in a row and was placed on a blacklist

Summary - system-generated log for sensitive topics (see 4.4.2

name - string used to signify which topic was affected. If an action does not involve a topic, it will contain the offending IP address used to make the request.

reason - string provided either by the user or by the system to provide further context on the action (in particular, to answer questions given by GDPR, "why was access provisioned?")

timestamp - UNIX timestamp (expressed as an integer) to determine when the transaction has happened

4.4.2 Report Generation

For the most sensitive topics, it is possible to enhance the security with extra measures and reporting. For every topic marked as sensitive, FlyTrap would maintain an in-memory list of all publishers or subscribers that have recently accessed the topic. Then, every specified period of time, a log will be generated and placed on blockchain that would outline all Ethereum public keys that have either published or subscribed to the given topic.

To put it in an example: if the reporting frequency is set at 30minutes and the system is started at 12:00, then client A publishes to topic X at 12:05, and client B publishes to topic X at 12:15, finally, at 12:30 a report would be generated and placed on the blockchain which would signify that both client A & client B published to topic X between 12:00 and 12:30. The reports should be read as follows: In the past X minutes, following public keys accessed following sensitive topics.

It is important to consider the implications of more and less frequent reporting - since every transaction placed on a blockchain with PoW consensus algorithm has embedded gas price[4], so the owner of the system would need to either accept the increased costs or decrease the reporting frequency. Of course, for PoA algorithms, there is no gas cost, but still, the chain code's size would grow faster in case of more frequent reporting. However, with decreased frequency, the precision also falls, so if our reporting is set to 24hrs, now we can only determine the access history in daily windows (rather than 30min ones, as shown in the example above).

4.4.3 Caching operations

Communicating with blockchain is a computationally expensive operation, and it is vital to ensure that it happens as rarely as possible in order to limit the latency caused by the security checks. Following the brute force approach, framework would issue a new request every time a connection starts - regardless whether it is part of a series of requests arriving in bulk. FlyTrap is using Ethereum as a de facto database layer, but at the same time aiming to cache the operations in-memory, which can be then quickly accessed if repeated requests occur. For checking the permissions, whenever a new request is issued, the result is stored in a map with mutex lock (to avoid race conditions).

To avoid memory filling too quickly (and eventually reaching the capabilities of the server that is running the software), the mapping used for caching will be erased every 24hrs or when the process is terminated/restarted - whichever happens first.

4.4.4 Interacting with blockchain

As FlyTrap aims to be deployed on a publicly available blockchain, anyone can interact with the data (as long as they pay the requested fee or identify with the relevant owner's private key). For administrative operations, a simple CLI is also included, which can be called directly (not necessarily through FlyTrap) to perform administrative tasks, such as adding new topics or modifying topics restrictions.

4.5 Presentation Layer

This layer combines everything above in a front-end allowing users to inspect and interact with stored information. Similarly to section 4.4.4, this could also be read through any of the publicly available front-ends used to interact with Ethereum transactions, but to provide a complete package, the project will also ship with a simple website aiming to satisfy requirements stated in chapter 3.

4.5.1 Website

The website will not allow for writing data to blockchain (thus, does not require Ethereum wallet) which implies that all reading operations are free. Instead of using the Blockchain CLI from

[4]Gas is a unit of work performed on Ethereum. The more complex the operation, the more gas is required, and thus more ETH currency is needed

Blockchain Layer, the website will ship with API of its own, which will perform calls against the specific Ethereum node directly.

Due to how blockchain is designed, it is not possible to lookup a transaction from specific time. Since it follows design of a linked-list, lookup of all elements is required to find requested date, thus resulting in the worst case complexity of $O(n)$, where n is the amount of transactions in the smart contract. This might severely impact performance, especially as the chain grows, thus the web app will keep track of transaction hashes in 5-day intervals in-memory, updating as further requests are performed.

Front-ends operations is also restricted by a time-constraint, i.e. it would be possible to request reports generated between 1970-01-01 and 1970-01-02, warning the user that less restrictive constraints might result in longer lookup times.

Chapter 5

Implementation

This chapter will discuss the technical specifications of the project and discuss the decisions that affected the process. I will also overview my workflow and iterative approach to the project. Last, but not least, I will include and credit a list of third party libraries that this framework makes use of.

5.1 Development process

In this section I will outline how the work on my dissertation was conducted along with pointing out the pitfalls and making sure that project remains fully functional.

5.1.1 Project plan

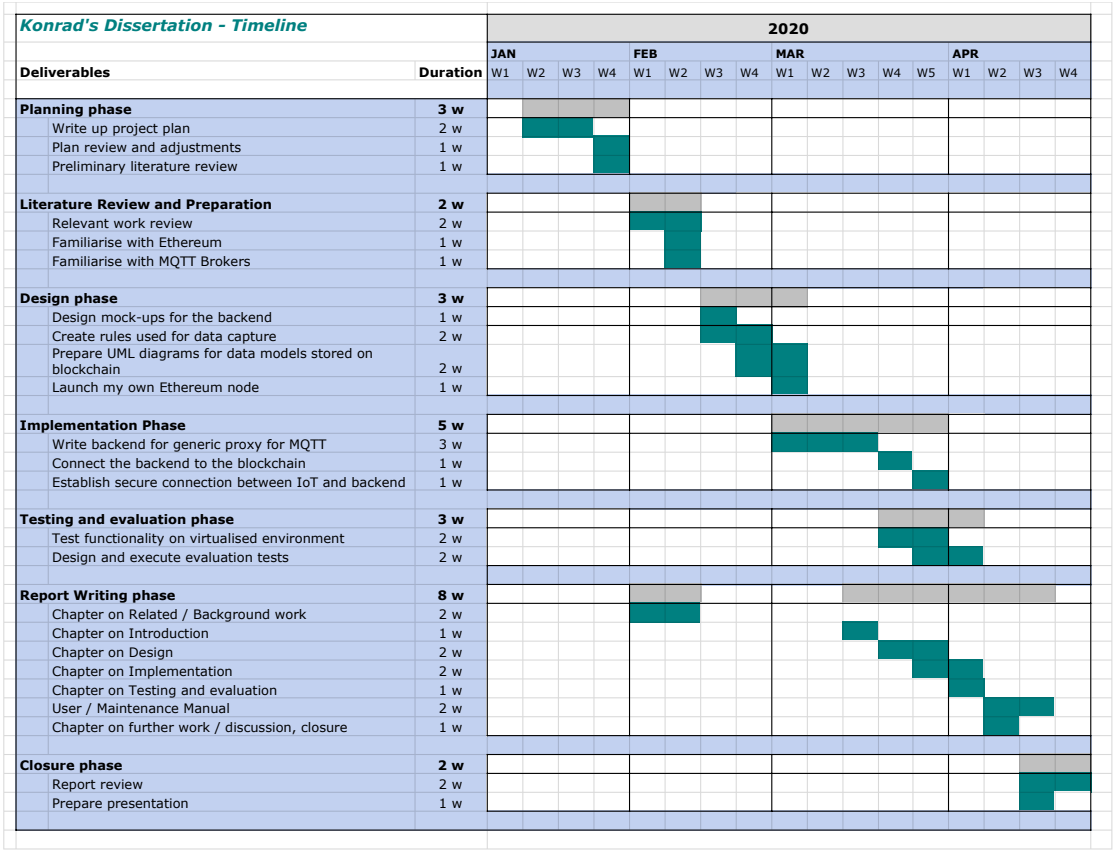

Figure 5.1: Project timeline

Figure 5.1 demonstrates the timeline of the project and all included phases. Each of those tasks was also divided onto smaller subsections (not shown on the diagram), which were worked on using agile approach, as described in the next section.

5.1.2 Iterative Approach

When designing my project plan, I distributed the workload onto small chunks and features which would have been implemented in an iterative way. I was following agile methodologies, dedicating each week on a different feature, where I would go through the entire development cycle for each unit. For example, generating the reports (section 4.4.2) was first introduced during a meeting with my supervisor, where I would establish the requirements and success criteria for this particular story. Then I would spend a day or two designing the flow and functionality, followed by an extra two days implementing designed features. At the very end, I would conduct testing and regression testing to make sure that the rest of the project still remains operational. This would have been concluded with a meeting with my supervisor to reflect on the sprint and determine whether success criteria were met.

5.1.3 Regression testing

Since I was following agile workflow when working on the project, it was important to ensure that none of the 'stories' (or, tasks) affected each other when completed. I could have found myself in a situation where working on the reporting of the events to the blockchain might have broken another, unrelated feature, e.g. verifying the authenticity of connecting clients. That is the reason behind keeping regression testing as a vital phase of development. I was able to achieve it through continuous, automated testing - which includes unit, integration, regression and manual testing.

5.2 Technologies

In this subsection I will describe the technologies used in this project, that is, languages, frameworks, third party libraries and different approaches when it comes to writing the code.

5.2.1 Languages used

Following programming languages have been used when working on this project:

- **Golang** (v1.14) - main driver behind the proxy, MQTT client and for communicating with the blockchain. It has also be used to write a simple backend for the web application.
- **Solidity** (v0.6.6) - language used to develop smart contracts on Ethereum.
- **HTML5 + CSS3 + JavaScript** - frontend stack used to create a simple website to display contents of blockchain.
- **Bash** (bash-5.0) - to design and execute tests capturing the latency of requests

5.2.1.1 Golang

Go (short for Golang) has been introduced for the first time in 2009 by Google [**?**]. It is a language which strongly follows parallel programming paradigms, allowing the developer to make use out of all available threads and cores to maximise the performance. Multi-thread performance was exceptioanlly sought for in this project, as FlyTrap is expected to handle many simultanous proxy connections. In Golang, programmer can make use of so-called goroutines, which the runtime can

dynamically either place on separate cores or run them concurrently on the same core - depending on the task.

All Ethereum frameworks (such as go-ethereum or geth, explained further in section **??** were also designed first in Golang. By making a decision to write my project in Go as well, I ensured maximum compatibility, as I was able to use official SDK's, rather than having to rely on unofficial solutions. I also was already experienced with Golang, as I spent year in industry at Google working on several projects in that language, which decreased the learning overhead for my dissertation.

Go in the project was used to write backend of the web server, CLI to communicate with the blockchain and the proxy itself which handles all incoming connections.

5.2.1.2 Solidity

Solidity [8] is the language used to write smart-contracts in Ethereum, so unfortunately it was a necessity. Its syntax is similar to Javascript, though it is statically, strongly typed. go-ethereum includes a utility which translates Solidity code into Golang methods, which then can be used to make calls against blockchain. There is no API which can be used to query / create transactions and rather developers need to use official SDKs published by Ethereum team.

5.2.1.3 HTLM5 + CSS3 + JavaScript

Since the web application serves only as a presentation and demonstration, I decided against any complex JavaScript or TypeScript frameworks and rather opted for a simple solution consisting of plain HTML + CSS and JavaScript. The web app also includes a small backend, thus asynchronous calls are performed from the client side towards the server to dynamically fill the content tables.

5.2.1.4 Bash

5.2.1.5 Considered alternatives

5.2.2 Third party libraries & resources

Following resources - which I was not the author of - were included in the project. All of them includ open-source license, allowing free use:

go-ethereum v1.9.10 [**?**] -

paho.golang v0.9.1 [**?**] -

tabulator v4.4.3 [**?**] -

5.2.3 Working with Blockchain

Ethereum was to be used as a data layer for the application. Of course, testing on the public chain was out of the question, due to the tremendous costs involved. Fortunately, Ethereum provides an easy way to start your own network, which would behave identically as the real one (including fake credits to use) - in the end, it would be indistinguishable from the real node for the applications attempting communication. When working on the project, I considered three ways of mocking the blockchain and in the end made sure to test FlyTrap in all three approaches:

5.2.3.1 Ganache

Ganache [**?**]

5.2.3.2 Geth

5.2.4 Development tools

5.2.4.1 Version Control

5.2.4.2 Text Editor

5.2.5 Configuration

Application involves a lot of configurables, to ensure that the end-user finds their preferred combination of settings. There are several ways

5.2.6 Logging

All major operations on FlyTrap, Blockchain CLI and Web Backend are logged to standard error through

Chapter 6

Evaluation & Testing

Chapter 7

Discussion

Appendix A

User Manual

Appendix B

Maintenance Manual

Bibliography

[1] Ashton, K. (1999). An introduction to the internet of things (iot). *RFID Journal*.

[2] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). Mqtt version 5.0. *OASIS Standard*.

[3] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2009). Keccak specifications. *Submission to nist (round 2)*, pages 320–337.

[4] Brandom, R. (2018). Facebook and google hit with $8.8 billion in lawsuits on day one of gdpr. The Verge, 25$.

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

[6] Cachin, C. et al. (2016). Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, page 4.

[7] California State Legislature (2018). Ab-375 privacy: personal information: businesses.

[8] Dannen, C. (2017). *Introducing Ethereum and Solidity*, volume 1. Springer.

[9] de Oliveira, D. L., Veloso, A. F. d. S., Sobral, J. V., Rabêlo, R. A., Rodrigues, J. J., and Solic, P. (2019). Performance evaluation of mqtt brokers in the internet of things for smart cities. In *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–6. IEEE.

[10] European Commission (2018). 2018 reform of eu data protection rules.

[11] Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11.

[12] Fakhri, D. and Mutijarsa, K. (2018). Secure iot communication using blockchain technology. In *2018 International Symposium on Electronics and Smart Devices (ISESD)*, pages 1–6. IEEE.

[13] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16.

[14] Gilbert, H. and Handschuh, H. (2003). Security analysis of sha-256 and sisters. In *International workshop on selected areas in cryptography*, pages 175–193. Springer.

[15] Halperin, D., Heydt-Benjamin, T. S., Ransford, B., Clark, S. S., Defend, B., Morgan, W., Fu, K., Kohno, T., and Maisel, W. H. (2008). Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 129–142.

[16] Index, B. E. C. (2020). Digiconomist.—[electronic resource]. *Mode of Access: https://digiconomist.net/bitcoin-energyconsumption*.

[17] Jakobsson, M. and Juels, A. (1999). Proofs of work and bread pudding protocols. In *Secure*

information networks, pages 258–272. Springer.

[18] Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63.

[19] King, S. and Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19.

[20] Klaokliang, N., Teawtim, P., Aimtongkham, P., So-In, C., and Niruntasukrat, A. (2018). A novel iot authorization architecture on hyperledger fabric with optimal consensus using genetic algorithm. In *2018 Seventh ICT International Student Project Conference (ICT-ISPC)*, pages 1–5. IEEE.

[21] Light, R. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265.

[22] Mijovic, S., Shehu, E., and Buratti, C. (2016). Comparing application layer protocols for the internet of things via experimentation. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–5. IEEE.

[23] Moore, G. E. et al. (1965). Cramming more components onto integrated circuits.

[24] Nakamoto, S. et al. (2008). A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*.

[25] Network, P. (2017). Proof of authority: consensus model with identity at stake.

[26] Pultarova, T. (2016). Webcam hack shows vulnerability of connected devices. *Engineering & Technology*, 11(11):10–10.

[27] Saleh, F. (2020). Blockchain without waste: Proof-of-stake. *Available at SSRN 3183935*.

[28] Sorrel, S. et al. (2018). The internet of things: Consumer industrial & public services 2018–2023. *Juniper, Sunnyvale, CA, USA*.

[29] Waehner, K. (2019). Iot and event streaming at scale with kafka & mqtt. https://www.confluent.io/blog/iot-with-kafka-connect-mqtt-and-rest-proxy/. Accessed: 2020-03-15.