FlyTrap: A Blockchain-based Proxy for Authorisation and Audit of MQTT Connections

1st Konrad M. Dryja
Computing Science
University of Aberdeen
Aberdeen, UK.
kdryja@gmail.com

2nd Milan Markovic
Computing Science
University of Aberdeen
Aberdeen, UK.
milan.markovic@abdn.ac.uk

3rd Peter Edwards
Computing Science
University of Aberdeen
Aberdeen, UK.
p.edwards@abdn.ac.uk

***Abstract*—MQTT brokers have become integral components of many IoT systems. However, the authorisation mechanisms defined by the MQTT protocol remain optional which may result in weak security configurations of MQTT brokers. In addition, with increasing prevalence of IoT solutions across many domains, MQTT brokers are used to forward various types of data including material that may be personal or commercially sensitive. Such data may be subject to a number of regulations (e.g. GDPR) which require trusted and auditable reporting that goes beyond local log files maintained by brokers. In this paper, we explore the potential of blockchains to facilitate a trusted and secure support infrastructure for authorisation and audit of MQTT connections.**

***Index Terms*—Internet of Things, MQTT, Blockchain, Transparency, Security**

I. Introduction

The term *Internet of Things* (IoT) refers to networks of devices with various sensing and actuating capabilities ubiquitously deployed in a physical environment and communicating using the Internet. IoT networks are typically comprised of a number of heterogeneous technologies, creating unique security and privacy challenges. In this paper, we focus on the authorisation and data auditability challenges associated with MQTT brokers, and we discuss the potential of blockchain technologies to facilitate such services.

MQTT brokers facilitate data exchange between connected IoT clients and therefore offer a means to audit and manage data channels associated with complex IoT infrastructures. For this reason, such brokers are a potential target of malicious attacks [1], [2]. The MQTT standard [3] does not impose any compulsory security or logging restrictions on brokers. If an MQTT broker is operated with default settings and without intrusion detection system monitoring, it may be vulnerable to data leakage through unauthorised topic access, be susceptible to DoS attacks, and be unable to track malicious clients, or even to determine the scale of a data breach [2].

With the increasing prevalence of IoT systems, it is also important that we consider and address the various privacy challenges that may arise. These challenges are closely related to the ability of a deployed IoT system to collect or process personal data. Such systems operating in the EU are subject to the General Data Protection Regulations (GDPR)[1] that imposes various restrictions on how personal data should be used and secured by systems, and also defines rights for end users. Compliance with GDPR often necessitates advanced forms of auditing mechanisms providing the ability to trace the use and storage of personal data. We argue that MQTT brokers are an important piece of IoT middleware that is ideally suited to support such auditing mechanisms.

In this paper, we present an approach that addresses the challenges outlined above by enhancing MQTT brokers with an open source Ethereum-based proxy service called *FlyTrap*. *FlyTrap* utilises smart contracts to manage the information on access permissions of the MQTT clients, to log information about events (such as client connections) to support auditability, and supports monetization of MQTT broker services via public blockchains. The remainder of this paper is structured as follows: section II discusses in more detail our motivation for the proposed enhancements; section III includes an overview of background and related work; section IV describes architecture and implementation details of *FlyTrap*; section V reports evaluation results; and sections VI and VII conclude the paper with discussion of limitations, remaining open challenges, and future work.

II. Motivation

Our research proposes several enhancements for MQTT brokers, and we will now discuss the motivation for each of these in turn.

A. Securing and Geo-restricting MQTT Broker Access

The MQTT standard defines only a simple authorisation mechanism based on client ID and password fields which may be included as part of the connect control packet. This enables the use of a range of password or token based mechanisms including custom solutions, external systems such as OAuth[2], and mechanisms provided by the operating system [3]. The recent version of the standard (MQTT 5.0) has introduced options for additional authentication mechanisms based on challenge/response flows using the new AUTH packets[3]. However, support for client authorisations by broker implementations

[1]https://gdpr-info.eu/

[2]https://tools.ietf.org/html/rfc6749

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

remains optional. Users deploying MQTT brokers should have access to unobtrusive authorisation frameworks for securing access to the broker, without being constrained by a specific broker implementation of optional features. In addition, in cases where sensitive (e.g. financial or personal) data are being handled by the broker, regulations or internal business policies may apply. For example, personal data originating in the EU would be protected by the GDPR, and as a consequence may be excluded from further processing (i.e. consumption by broker's clients) outside the EU. To support such restrictions, an authorisation framework should be capable of geo-restricting access to the broker.

B. Data Traceability and Transparency

MQTT broker implementations offer a variety of logging mechanisms. For example, Mosquitto[4] allows connection, subscription and message data to be saved to disk[5]. Such information can help support fault detection and debugging, and also enables reconstruction of provenance of data use. Being able to track data forwarding is important for determining the scale of data leaks, or to identify services that had access to messages containing personal data. The latter is particularly relevant in the context of the GDPR's right to be forgotten[6] that mandates personal data processors to delete all personal data relating to a particular individual if this is requested. To improve trust in such logs, we propose that these should be stored transparently using immutable storage mechanisms such as blockchains that allow for both public and permissioned access (i.e., using private blockchain networks) to audit the logging data.

III. Background & Related Work

A. The MQTT Protocol

The MQTT protocol (currently version 5.0) is an OASIS standard [3] publish/subscribe messaging transport protocol for client-server communication. Due to its small transport overhead, MQTT is a popular protocol used in IoT infrastructures and is currently supported by popular commercial products such as Azure IoT Hub [4] and AWS IoT [5]. The protocol is typically run over TCP/IP networks with three basic message delivery quality assurances: message delivered at most once (*QoS 0*); at least once (*QoS 1*); and exactly once (*QoS 2*). The protocol does not impose any restrictions on the structure of the message payload. The communication between clients is managed by the MQTT broker which manages a list of topics through which clients (e.g., IoT devices) can either send (i.e., publish) or receive (i.e., subscribe) arbitrary types of data. The MQTT protocol defines a series of control packets which direct the interactions between the broker and clients. Figure 1 illustrates a basic message workflow between the MQTT broker, a publisher sending data to a topic, and a topic subscriber receiving the data.

[4]https://mosquitto.org

[5]https://mosquitto.org/man/mosquitto-conf-5.html

[6]https://gdpr-info.eu/issues/right-to-be-forgotten/

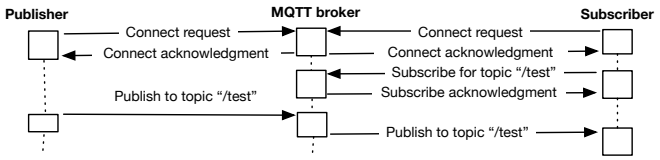

Fig. 1. An overview of a basic message workflow between the MQTT broker, publisher, and subscriber.

B. Ethereum

A blockchain network is a type of distributed ledger technology that provides immutable trusted data storage of events/transactions based on cryptographic proofs and was initially popularised by the online crypto currency market [6]. *Ethereum*[7] refers to the technology used by the popular public blockchain network of the same name and offers a Turing-complete scripting language for creating arbitrary smart-contracts. *Ethereum* was the first and most successful blockchain technology that popularised creation of decentralised applications (Dapps) built on such contracts [7]. While other blockchain technologies such as Hyperledger[8] also support smart contracts, we have chosen to work with *Ethereum* due to its widespread popularity and maturity. Smart contracts are programs that can execute on the network and define, for example, entities that can access the functionalities of a particular Dapp, associated costs, event information logged by the network, etc. This technology is critical to support utilisation of public blockchain networks in contexts that go beyond the cryptocurency markets.

In addition, *Ethereum* can be deployed with a Proof-of-Authority [8] consensus algorithm. This approach offers no rewards for adding new blocks to the chain, so it is not used in public blockchains. Such an approach is more suitable for private permissioned consortia networks with different business entities acting as network nodes similar to the networks supported by Hyperledger. This extends the applicability of *Ethereum*-based solutions to domains such as closed supply chains, which would require private networks to protect commercially sensitive data and to shield the solution from the volatility of prices associated with operations on the public blockchain network (see Section VI for further discussion).

C. Security Frameworks for MQTT

To date, various approaches to enhance MQTT security have been proposed in the literature [9]–[11]. For example, Refaey et al. [9] propose a single packet authorisation approach based on the software defined perimeter (SDP) framework[9]. The SDP layer shields the MQTT broker from interaction with unauthorised or malicious clients, and is demonstrated to be effective against DoS attacks. Shin et al. [10] discuss a security framework incorporating the AugPAKE protocol for securing communication between clients and the broker without the

[7]https://ethereum.org/

[8]https://www.hyperledger.org/

[9]https://cloudsecurityalliance.org/artifacts/sdp-architecture-guide-v2/

need for certificate validation and certificate revocation checks. More recently, Buccafurri et al. [11] propose the use of *Ethereum* blockchain and smart contracts as a trusted third party for validation of one-time-passwords (OTP) used to authenticate MQTT clients. They argue that the approach is suitable for unencrypted communication channels and the MQTT broker is assumed to bear the costs of the smart contract operation.

D. Transparency in IoT

The concept of *transparency* in the context of the IoT is closely linked to privacy of users affected by such technologies. Privacy may be impacted if the collected IoT data include personal information or if such details can be inferred by the exposure of behaviour patterns, or by combining a variety of data sources [12] [13]. In our previous work, we highlighted the lack of machine-understandable transparency in IoT systems [14] and more specifically MQTT brokers [15], [16]. We argued that such transparency is necessary to automate audit of IoT systems - a key element in facilitating regulatory compliance and user's trust in such systems.

Recently, blockchains have been identified as a suitable technology for enhancing transparency of IoT systems by recording privacy-related information, for example, user consent [17], [18]. They have been also considered as means to support other aspects of IoT security. For example, Novo proposes a blockchain-based access management platform for IoT devices that addresses the scalability challenges faced by centralized solutions [19]. Gatollin et al. present a transparent key management framework for device authentication utilising public blockchains to ensure auditability of client authentication events over time [20].

E. Research Gap

A common drawback among state of the art solutions that address the authorisation of MQTT client connections is the requirement to adopt additional security layers (and modified middleware components) - something that we believe may hinder future adoption. In contrast, *FlyTrap* is designed to seamlessly integrate with message brokers based on the well established MQTT standard without further extensions to the broker's implementation. Moreover, in the current state of the art security and privacy of MQTT brokers tends to be considered in isolation. We argue that these two challenges share similarities such as the requirement for auditable transparent access management to topics registered by the broker. We therefore considered both of these challenges during the design of *FlyTrap*.

IV. *FlyTrap* Architecture & Implementation

A. Architecture Overview

FlyTrap is designed as a proxy service that enhances the standard MQTT operations with *Ethereum*-based access control and topic management as well as tools for additional logging and auditing. The *FlyTrap* is assumed to be deployed alongside an MQTT broker, where the service is acting as a secure proxy intercepting all communications between the clients and the broker (both TCP and TLS connections are supported). All other clients then interact with the MQTT broker via the *FlyTrap* proxy service, with client identities handled by the *Ethereum* blockchain.

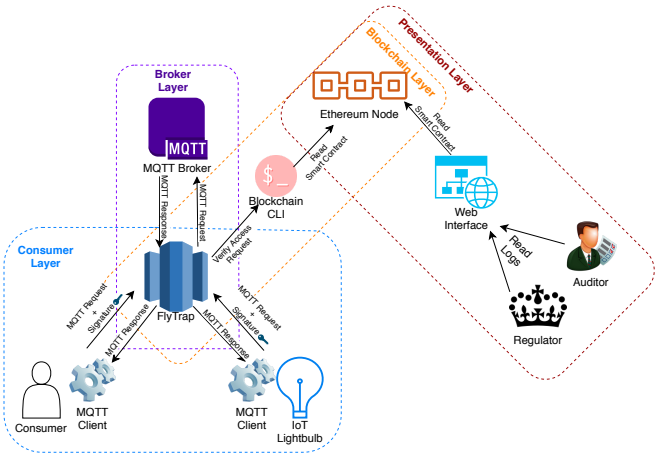

Fig. 2. *FlyTrap* high-level architecture overview.

Figure 2 illustrates the four-tier architecture of *FlyTrap*. The *Consumer Layer* is responsible for interacting with consumer end-devices (i.e. MQTT clients) via MQTT v5.0 compliant TCP/TLS packets. The *Broker Layer* is responsible for facilitating communication (e.g., forwarding authorised MQTT messages) between *FlyTrap* and any MQTT v5.0 compliant message broker. The *Blockchain Layer* manages communication with the *Ethereum* node by creating, reading, and modifying smart contracts. The *Presentation Layer* contains a set of tools for accessing and visualising information stored on the blockchain and thus enabling the audit of the broker's set up (e.g., list of allowed topics and associated restrictions) and its data forwarding events.

The core of the *FlyTrap* framework has been implemented using *Golang*[10] (v1.14), the interaction with the Ethereum smart contracts is handled using *Solidity*[11] (v0.6.6) and the web-based user front end has been implemented using a standard combination of HTML5, CSS3 and JavaScript. The framework is available as an opensource project hosted on GITHUB[12]. The remainder of this section describes in more detail the individual layers of the *FlyTrap* framework and provides examples of its use.

B. Consumer Layer

This layer handles the connection requests and forwards all authorised PUBLISH/SUBSCRIBE messages. To communicate with *FlyTrap*, an MQTT client has to provide additional authorisation values as part of the MQTT packet. These are provided as part of *user properties* [3] which were introduced in MQTT v5.0 and allow clients to include key-value pairs

[10]https://golang.org/
[11]https://solidity.readthedocs.io/en/latest/
[12]https://github.com/kdryja/FlyTrap

in the MQTT packets. *FlyTrap* expects two custom properties set as part of the CONNECT message, plain-text public key and public key signed with the private-key (see Figure 3). A client has to be able to produce the required signature itself or in case of resource constrained devices has access to pre-computed signatures.

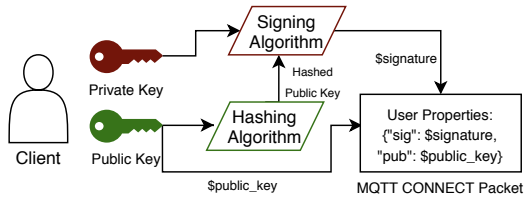

Fig. 3. Client signing public key and attaching required values to CONNECT message as user properties.

FlyTrap utilises Elliptic Curve Digital Signature [21] and Keccak-256 hashing algorithms to process the public keys and signatures (see Figure 4). The public key has exactly 160 bits and is also used as an addresses on the Ethereum blockchain [22].

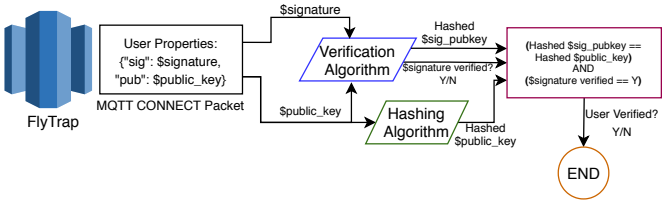

Fig. 4. *FlyTrap* verifying a signature.

C. Broker Layer

This layer handles communication between *FlyTrap* and the MQTT broker (e.g. Mosquitto). To maximise performance, *FlyTrap* and the broker should be hosted on the same machine or at least the same network, to minimise the latency of TCP/TLS connections between the broker and the proxy. The broker layer is bidirectional - i.e. it forwards the authorised MQTT packets from the client to the broker and also all packets from the broker to the client. The layer handles each client connection through a separate port.

D. Blockchain Layer

Each *Flytrap* instance requires a smart contract deployed on the *Ethereum* network. The smart contract is specified using Solidity. *FlyTrap*'s smart contract should be configured by the organisation operating the *FlyTrap* instance to associate the contract with the organisation's *Ethereum* private key. Depending on the contract configuration, *FlyTrap* provides an opportunity to monetise the operations supported by the MQTT broker. For example, the topic owner can specify additional costs associated with joining topics as publishers or subscribers (i.e. adding *Ethereum* public keys) to the contract. Any person wishing to join the topic then pays the required fee which will be transferred to the topic owner (after deduction of the cost of the transaction).

FlyTrap's smart contract contains chain code capable of verifying connecting clients and relevant data structures for managing topics and recording events. A variable *topic* maps string values representing the names of the topics to a *Topic* structure capturing all the required metadata used to mange the topic. This includes a flag to indicate whether the topic is sensitive, the topic name, owner, a country code detailing access locations for geo-restricted topics, fees for publishing and subscribing, and a list of publishers and subscribers (see Figure 5).

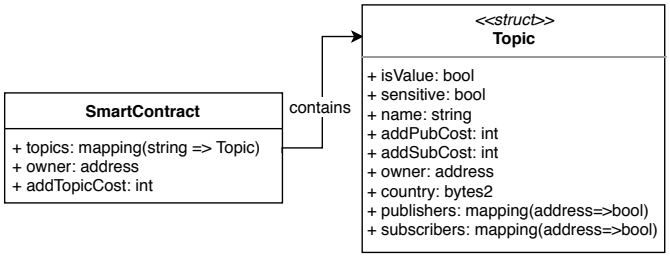

Fig. 5. Topic structure stored on blockchain.

FlyTrap also utilises *Event*, a structure used in Solidity, which attaches itself to the transaction log. This makes it possible to append information such as a user-specified reason or timestamps to all smart contract operations. *Event* information is encoded in Application Binary Interface (ABI)[13] JSON and sent along with the transaction. Figure 6 illustrates the *Event* structure used in *FlyTrap*.

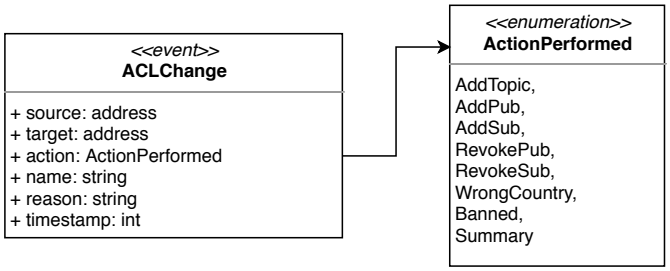

Fig. 6. *Event* structure as stored in transaction log.

FlyTrap also supports enhanced logging for topics marked as *sensitive*. For each sensitive topic *FlyTrap* maintains an in-memory list containing *Ethereum* public keys of all clients that posted or received data, which is periodically saved on the blockchain according to the reporting frequency settings set by the user. For example, if the reporting frequency is set at 30 minutes 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* and client *B* published to topic *X* between 12:00 and 12:30.

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

The aggregated reporting feature was introduced due to a specific design characteristic of the blockchain network; every transaction placed on a blockchain with the *Proof-of-Work* consensus algorithm has an embedded gas price[14]. The owner of the system would need to either accept the increased costs associated with more detailed reporting or a decreased reporting frequency.

To improve the latency of information retrieval from the network and to further reduce the price of blockchain operations, *FlyTrap* supports caching mechanisms for some of its operations. This includes an in-memory map with mutex lock (to avoid race conditions between different goroutines), which stores information for repeated requests (e.g., client connections).

E. Presentation Layer

FlyTrap provides a simple Web based user interface for viewing the data stored on the blockchain network. The Web app only supports read operations over the blockchain which are free and therefore do not require an *Ethereum* wallet.

The user interface allows various activities performed on the smart contract to be managed, such as addition of authorised publishers to a topic; it also provides reports on failed connection attempts, such as connections from a disallowed country. The tool also allows topics linked to specific clients to be identified by their public *Ethereum* keys, and aggregated reports to be generated based on user-defined time windows (as discussed in the previous section).

Since the blockchain follows the design of a linked-list, it is not possible to look up a transaction from a specific time directly. Instead, look up of all elements is required resulting in the worst case complexity of $O(n)$, where n is the number of transactions in the smart contract. To improve performance, the Web app maintains in-memory track of transaction hashes in 5-day intervals[15], updating as further requests are performed.

F. Example Authorisation Workflows

Figure 7 illustrates an example workfow of a client successfully publishing a message under a specific topic. The diagram illustrates an ordered set of steps that are performed by the client, *FlyTrap* instance, MQTT broker and the blockchain node to successfully publish a message.

As outlined above, a number of components including the *FlyTrap* client, *FlyTrap* proxy, MQTT broker and the *Ethereum* blockchain node must interact in order to publish a message. The list below provides a detailed description of the publishing workflow where each step is annotated with either **BL** - Blockchain Layer, **CL** - Consumer Layer or **ML** - MQTT Broker Layer to indicate the layer in which this step takes places.

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* extracts the signature from the optional field and verifies its integrity. It also checks if the client has not been attempting many unsuccessful connections. Finally, *FlyTrap* responds 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**)
3) Client now forwards the relevant PUBLISH/SUBSCRIBE packet to *FlyTrap* (as it still believes that it is a regular MQTT broker). (**CL**)
4) *FlyTrap* extracts the requested topic from the MQTT packet and communicates 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 now responds with PUBACK, indicating successful PUBLISH. (**ML**)
7) Finally, *FlyTrap* proxies the same PUBACK packet back to the initial client to let them know that the operation was successful - and gracefully terminates the connection. (**CL**)

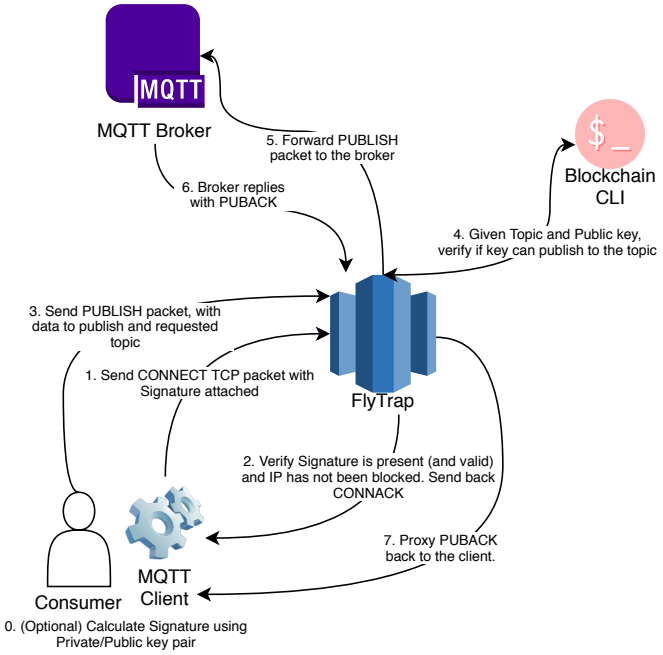

Fig. 7. Example of a successful workflow publishing a message on the broker through *FlyTrap*.

To contrast the successful message publishing workflow we now describe the steps of a failed publish workflow (also

[14]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.
[15]This may be changed according to specific organisational needs.

illustrated in Figure 8).

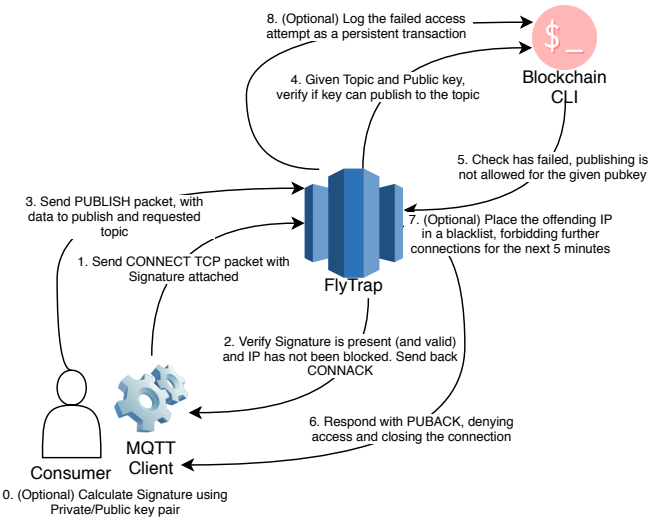

Fig. 8. Example of a workflow failing to publish a message on the broker through *FlyTrap*

0-3) Same as in Figure 7.

4-5) Having verified the authenticity of the Public Key, *FlyTrap* attempts to verify with the smart contract whether the client can access the topic. As the response from the blockchain is negative, the client is not allowed to publish on the requested topic. (**BL**)

5) *FlyTrap* sends PUBACK back to the client, setting reason code[16] to "Not authorised", at the same time terminating the connection with the client. (**CL**)

6) (optional) The framework uses a cache of previous connections to check if the originating IP has not exceeded the maximum number of allowed attempts. If it has, the IP address is placed on a blacklist, and every subsequent connection is then denied for the specified time. In this example, that is 5 minutes. (**CL**)

7) (optional) If the client is banned, *FlyTrap* registers a new transaction on the blockchain to persistently log this event to check potential attack attempts or generate reports. (**BL**)

V. Evaluation

The *FlyTrap* prototype was evaluated based on the following criteria: its ability to capture logging information on a blockchain network as set out in our objectives; the cost of operation on a public *Ethereum* network; and its impact on MQTT broker performance.

A. Setup

The experiments were performed on a single Ubuntu 16.04.6 LTS virtual server instance with the following allocated resources: four core Intel Xeon CPU E5-2630 @ 2.30GHz; 70 GB RAM, 15 MB L3 Cache. The experimental *Ethereum* network was set up using Ganache[17]. The network was started with an account owning 100 ETH coins. An example smart contract was created containing a test topic and lists of allowed publishers and subscribers.

B. Cost of Operation

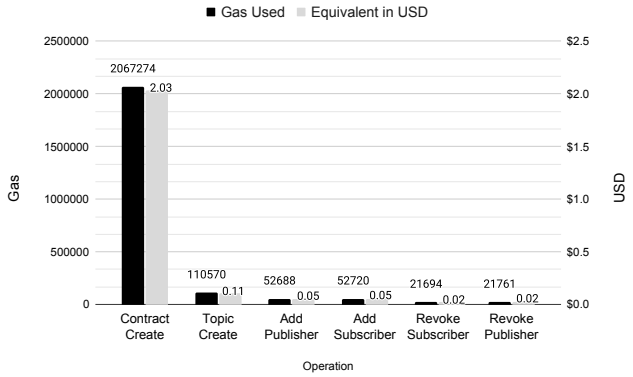

Fig. 9. Cost comparison of various operations on the blockchain for April 2020 (1 ETH = $182.29 and 1 Gas = 0.0000000054 ETH).

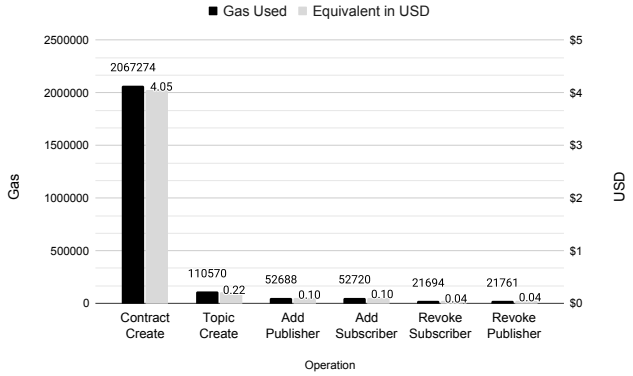

Fig. 10. Cost comparison of various operations on the blockchain for September 2021 (1 ETH = $3817.20 and 1 Gas = 0.00000026 ETH).

Six operations supported by *FlyTrap* were performed to extract their costs in Gas, which were then converted to USD. As illustrated by Figures 9 and 10 costs were calculated at specific times in 2020 and 2021. The results demonstrate how the extreme volatility of cryptocurrency values could impact on operational costs associated with a solution such as *FlyTrap* if operated on a public network. The most expensive operation is the creation of the smart contract, however, this operation is only performed once. All other subsequent operations modifying the state of the contract cost significantly less. It should also be noted that Figures 9 and 10 illustrate baseline costs that would be incurred to operate *FlyTrap* on a public *Ethereum* network at the time of each experiment.

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

[17]https://www.trufflesuite.com/ganache

These would increase further if the owner of the smart contract wished to add a profit margin to individual operations.

C. FlyTrap Performance

We focus now on the framework's influence on latency of MQTT packet transmissions and its scalability for concurrent client connections.

1) Packet Latency: This experiment evaluated the latency of MQTT packet transmissions by comparing three different configurations: a baseline (standard) MQTT broker; *FlyTrap*; and *FlyTrap* with caching functionality enabled. The latency was measured for three MQTT packets, namely, CONNECT, SUBSCRIBE, and PUBLISH. Time was observed from the point when a packet was sent by the client until a corresponding ACK packet was received back from the MQTT broker. As no caching occurs during the CONNECT stage, the caching functionality does not influence transmission of CONNECT packets. Both plain TCP and TLS connections were evaluated (see Figure 11). For each configuration 105 transmissions were executed, with the first five being discarded as *warm up* measurements.

Results indicate that *FlyTrap* increases the latency for CONNECT, SUBSCRIBE, and PUBLISH packets by $53.73\%, 61.92\%, 62.67\%$ for TLS and by $18.13\%, 43.30\%, 31.58\%$ for plain TCP connections respectively. The caching functionality reduces the increase for PUBLISH and SUBSCRIBE packets to $23.41\%, 15.55\%$ for TLS and to $18.12\%, 22.19\%$ for plain TCP connections respectively.

2) Scalability: In this experiment, a single PUBLISH packet was transmitted through *FlyTrap* with TLS enabled from 10, 100, 1000, and 10,000 concurrently connected MQTT clients. Every client sending the PUBLISH packet at the same time used a separate process, thus ensuring that they did not interfere with each other. Each client was identified with a different public key, forcing *FlyTrap* to perform authorisation flow. The measured variable includes the time between issuing the PUBLISH packet and receiving a PUBACK packet back from the server. We did not observe any impact on the speed when varying the number of simultaneous client connections between 10 and 10,000 with the average broker response time remaining at 0.1 ms. The results therefore suggest that introduction of the *FlyTrap* proxy has no impact on the scalability of the MQTT broker.

VI. Limitations & Future Challenges

The decentralised nature of public blockchain networks makes the *FlyTrap* data layer resilient against hardware failures and offers 100% up time. However, using a public blockchain network means that the *FlyTrap* framework introduces some direct costs for service operators. For example, repeated failed connection attempts from a client would result in a persistent log on the blockchain, which is associated with blockchain costs and is incurred by the broker's owner. This creates an opportunity for an attack where an adversary could attempt to increase costs associated with operating *FlyTrap* through malicious connection attempts. Given the nature of current blockchain currencies it would be also difficult to predict fees for transactions, which may make apps based on an open blockchain network unstable.

Identities of individual clients are recorded using their public *Ethereum* keys which only provides pseudonymous protection. Previous research has demonstrated that the degree of privacy protection offered by the public key mechanisms on the Bitcoin network may be insufficient and that as a result, activities of certain users may be observed [23]. Further work is therefore needed to assess whether exposing interactions of IoT clients and brokers on a public blockchain network would lead to potential security or privacy risks.

Public blockchain networks have also been criticised for their extremely high carbon footprint due to the *Proof-of-Work* consensus algorithm used to validate transactions [24]. Such algorithms are dependent on the use of resources to solve difficult cryptography puzzles also referred to as *mining*. Currently, *Ethereum* uses the *Proof-of-Work* consensus algorithm but in the future the network is anticipated to move towards a more efficient *Proof-of-Stake* algorithm which eliminates the need to run energy inefficient mining operations [25]. Further research will be needed to better understand the environmental impact of blockchain-based solutions in the IoT context.

Finally, the approach presented in this paper relies on the existence of a trusted channel between the broker, *FlyTrap*, and clients. To achieve this, *FlyTrap* supports TLS connections, however, plain TCP connections are also supported for scenarios where using TLS is not feasible. It is therefore possible that man-in-the middle attacks may occur. *FlyTrap* would be also ineffective if it is possible to bypass the proxy and communicate directly with the broker.

VII. Conclusions & Future Work

In this paper, we have presented a novel *Ethereum*-based proxy service to support authorisation, monitoring and monetisation of MQTT broker client connections. We have demonstrated the feasibility of this approach via an implementation and evaluation of the *FlyTrap* framework.

In future, we aim to evaluate the suitability of other popular blockchain networks for integration with the *FlyTrap* framework. We will also seek to evaluate the solution in a real world deployment scenario (e.g., food supply chain monitoring) to explore its effectiveness with real users. Finally, as the most recent versions of popular MQTT Brokers (such as Mosquitto) have introduced support for webSockets, we will explore how this would enable us to mitigate the current *FlyTrap* limit on maximum concurrent connections, which is the maximum number of ports on Linux (i.e., 65535).

References

[1] S. Andy, B. Rahardjo, and B. Hanindhito, "Attack scenarios and security analysis of mqtt communication protocol in iot system," in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017, pp. 1–6.

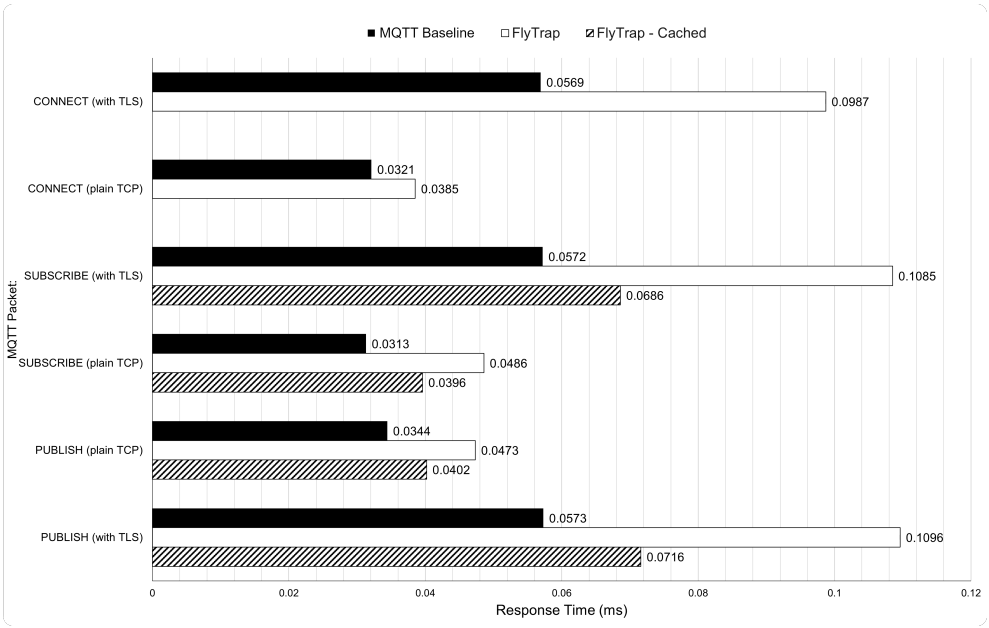

Fig. 11. A comparison of latency observed for different MQTT packets with and without TLS enabled.

[2] J. J. Anthraper and J. Kotak, "Security, privacy and forensic concern of mqtt protocol," in *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*, 2019.

[3] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "Mqtt version 5.0," *OASIS Standard*, 2019, https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901000.

[4] S. Klein, *IoT Solutions in Microsoft's Azure IoT Suite*, 2017.

[5] S. Bhatt, F. Patwa, and R. Sandhu, "Access control model for aws internet of things," in *International Conference on Network and System Security*. Springer, 2017, pp. 721–736.

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

[7] V. Buterin, "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform," *Etherum*, 2014, https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[8] P. Network, "Proof of authority: consensus model with identity at stake," 2017.

[9] A. Refaey, A. Sallam, and A. Shami, "On iot applications: a proposed sdp framework for mqtt," *Electronics Letters*, vol. 55, no. 22, pp. 1201–1203, 2019.

[10] S. Shin, K. Kobara, Chia-Chuan Chuang, and Weicheng Huang, "A security framework for mqtt," in *2016 IEEE Conference on Communications and Network Security (CNS)*, 2016, pp. 432–436.

[11] F. Buccafurri, V. De Angelis, and R. Nardone, "Securing MQTT by blockchain-based otp authentication," *Sensors (Switzerland)*, 2020.

[12] L. Urquhart, T. Lodge, and A. Crabtree, "Demonstrably doing accountability in the internet of things," *International Journal of Law and Information Technology*, vol. 27, no. 1, pp. 1–27, 2018.

[13] S. R. Peppet, "Regulating the internet of things: first steps toward managing discrimination, privacy, security and consent," *Tex. L. Rev.*, vol. 93, p. 85, 2014.

[14] M. Markovic, D. Garijo, P. Edwards, and W. Vasconcelos, "Semantic modelling of plans and execution traces for enhancing transparency of iot systems," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2019, pp. 110–115.

[15] M. Markovic and P. Edwards, "Enhancing transparency of mqtt brokers for iot applications through provenance streams," in *Proceedings of the 6th International Workshop on Middleware and Applications for the Internet of Things*, ser. M4IoT '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 17–20. [Online]. Available: https://doi.org/10.1145/3366610.3368099

[16] M. Markovic, D. Corsar, W. Asif, P. Edwards, and M. Rajarajan, "Towards transparency of iot message brokers," in *Provenance and Annotation of Data and Processes*, K. Belhajjame, A. Gehani, and P. Alper, Eds. Cham: Springer International Publishing, 2018, pp. 200–203.

[17] K. Rantos, G. Drosatos, K. Demertzis, C. Ilioudis, A. Papanikolaou, and A. Kritsas, "Advocate: A consent management platform for personal data processing in the iot using blockchain technology," in *Innovative Security Solutions for Information Technology and Communications*, J.-L. Lanet and C. Toma, Eds. Cham: Springer International Publishing, 2019, pp. 300–313.

[18] S. Cha, T. Tsai, W. Peng, T. Huang, and T. Hsu, "Privacy-aware and blockchain connected gateways for users to access legacy iot devices," in *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, 2017, pp. 1–3.

[19] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[20] A. Gattolin, C. Rottondi, and G. Verticale, "Blast: Blockchain-assisted key transparency for device authentication," in *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*. IEEE, 2018, pp. 1–6.

[21] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[22] M. Dameron, "Beigepaper: An ethereum technical specification," 2017.

[23] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*. New York, NY: Springer New York, 2013, pp. 197–223.

[24] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, 2014, pp. 280–285.

[25] F. Saleh, "Blockchain without waste: Proof-of-stake," *Review of Financial Studies, Forthcoming*, 2020, available at SSRN:https://ssrn.com/abstract=3183935.