future internet

MDPI

Article

Achieving Accountability and Data Integrity in Message Queuing Telemetry Transport Using Blockchain and Interplanetary File System

Sara Lazzaro and Francesco Buccafurri *

Department DIIES, University Mediterranea of Reggio Calabria, Via Università 25, 89122 Reggio Calabria, Italy; sara.lazzaro@unirc.it

* Correspondence: bucca@unirc.it

Abstract: Ensuring accountability and integrity in MQTT communications is important for enabling several IoT applications. This paper presents a novel approach that combines blockchain technology and the interplanetary file system (IPFS) to achieve non-repudiation and data integrity in the MQTT protocol. Our solution operates in discrete temporal rounds, during which the broker constructs a Merkle hash tree (MHT) from the messages received. Then the broker publishes the root on the blockchain and the MHT itself on IPFS. This mechanism guarantees that both publishers and subscribers can verify the integrity of the message exchanged. Furthermore, the interactions with the blockchain made by the publishers and the broker ensure they cannot deny having sent the exchanged messages. We provide a detailed security analysis, showing that under standard assumptions, the proposed solution achieves both data integrity and accountability. Additionally, we provided an experimental campaign to study the scalability and the throughput of the system. Our results show that our solution scales well with the number of clients. Furthermore, from our results, it emerges that the throughput reduction depends on the integrity check operations. However, since the frequency of these checks can be freely chosen, we can set it so that the throughput reduction is negligible. Finally, we provided a detailed analysis of the costs of our solution showing that, overall, the execution costs are relatively low, especially given the critical security and accountability benefits it guarantees. Furthermore, our analysis shows that the higher the number of subscribers in the system, the lower the costs per client in our solution. Again, this confirms that our solution does not present any scalability issues.

Keywords: MQTT; accountability; end-to-end; data integrity

Citation: Lazzaro, S.; Buccafurri, F. Achieving Accountability and Data Integrity in Message Queuing Telemetry Transport Using Blockchain and Interplanetary File System. *Future Internet* **2024**, *16*, 246. https://doi.org/10.3390/fi16070246

Academic Editor: Gianluigi Ferrari

Received: 16 June 2024
Revised: 9 July 2024
Accepted: 11 July 2024
Published: 13 July 2024

1. Introduction

The Internet of Things (IoT) has enhanced several fields, from smart homes [1,2] to industrial automation [3–5], by enabling communication between low-end devices. Message queuing telemetry transport (MQTT) [6] is one of the main communication protocols adopted by such devices. MQTT is a lightweight, publish–subscribe messaging protocol suited for constrained environments characterized by low network and computational performance [7]. In MQTT, the communication between clients (publishers and subscribers) is mediated by a broker. This architecture is efficient since it leaves all the complexities to the broker [8]. However, relying on a third party may introduce significant security challenges.

Indeed, in many MQTT applications, the broker could potentially be compromised or untrusted. This can lead to the risk of unauthorized tampering with messages [9]. For example, if patient data are maliciously altered in healthcare applications, the consequences could be severe. Similarly, in smart city infrastructures, it is important to ensure the integrity of the transmitted data in order to guarantee reliable service delivery.

In addition to integrity, ensuring accountability for both publishers and brokers in MQTT is important so that they both cannot deny having sent a flow of messages. For

example, if a publisher sends wrong information, it must be held liable for any damage caused by the sent data. Similarly, on the broker's side, accountability is also important. Brokers often should enforce access control and ensure that only verified publishers are allowed to publish on specific topics. For example, subscribers relying on critical information, such as in financial or health-related applications, must be assured that their data come from verified publishers. By providing accountability on the broker side, our solution guarantees that brokers perform these verification checks.

Addressing these security concerns requires robust mechanisms to guarantee that messages are not tampered with and that the involved parties, i.e., publishers and brokers, cannot deny having sent their messages. This paper proposes a novel approach to achieving both accountability and data integrity in MQTT.

Our proposal integrates blockchain technology [10–12] and the interplanetary file system (IPFS) [13,14] into the MQTT protocol. In particular, the immutability property of the blockchain ledger is fundamental to achieving accountability. However, storing large amounts of data in the blockchain has a non-negligible economic cost, as well as leading to scalability problems in the blockchain [15]. IPFS, on the other hand, provides an efficient and decentralized way to store message data off the blockchain. By storing only the hash on the blockchain and the actual file on IPFS, the blockchain scalability issues are mitigated, as only the critical data are stored on-chain [16].

Our solution is as follows. The broker builds a Merkle hash tree (MHT) of the messages received from the publisher during discrete temporal rounds. It then publishes the root of the MHT on the blockchain and stores the tree itself on IPFS. This allows both publishers and subscribers to verify the integrity of the exchanged messages. It also ensures that both publishers and the broker cannot repudiate the messages they have sent.

In our study, we performed a security analysis to demonstrate that our approach achieves accountability and data integrity in MQTT communications under standard assumptions. Additionally, we conducted an experimental campaign to examine the scalability and throughput of the system. Our findings indicate that our solution scales well with the number of clients. Moreover, the results reveal that the reduction in throughput is influenced by the integrity check operations. However, since the frequency of these checks can be adjusted as needed, we can configure it to ensure that the throughput reduction remains negligible. Finally, we conducted a detailed analysis of the costs of executing smart contracts on the Ethereum blockchain. Our analysis shows that the overall costs of this solution are affordable. Furthermore, our analysis demonstrates that as the number of subscribers in the system increases, the cost per client decreases. This further confirms that our solution does not encounter any scalability issues. Overall, our results show that it is feasible to implement our solution in the real world to ensure accountability and data integrity in MQTT communications.

The rest of the document is structured as follows. Section 2 provides background information on MQTT, blockchain, and IPFS technologies. Section 4 describes the motivations and potential use cases for our approach. Section 5 presents our approach. This is followed by an analysis of the costs of smart contract execution in Section 6 and an analysis of the performance of our solution in Section 7. In Section 8 we analyze the security of the proposed approach. Finally, Section 9 concludes the paper.

2. Background

In this section, we provide a brief overview of the main technologies adopted in our approach: MQTT, blockchain, and IPFS.

2.1. MQTT

MQTT [6] is a lightweight messaging protocol based on the publish–subscribe model. It is designed for constrained devices and low-bandwidth or unreliable networks.

The MQTT architecture consists of three main components:

- Broker: The broker is the central hub that mediates the communication between publishers and subscribers on given topics.
- Publisher: A publisher is a client that sends messages to the broker labeled with given topics.
- Subscriber: A subscriber is a client that receives messages from the broker. Subscribers express interest in one or more topics and receive messages related to those topics.

The message flow in MQTT is as follows: (1) a publisher sends a message to the broker on a given topic, (2) the broker receives the message and forwards it to all the subscribers interested in that topic, and (3) the subscribers receive the message from the broker.

Additionally, MQTT allows publishers and subscribers to specify the quality of service (QoS) level they want. There are three QoS levels [17], each guaranteeing message delivery, as follows:

- QoS 0: At most one delivery.
- QoS 1: At least one delivery.
- QoS 2: Exactly one delivery.

2.2. Blockchain and IPFS

In the following, we describe two technologies we exploit in our approach: blockchain and IPFS.

2.2.1. Blockchain

Blockchain [10,11] consists of a distributed ledger that records transactions in a secure, transparent, and tamper-proof manner. Through blockchain, an entity can generate a transaction towards another entity to exchange a value. This transaction is validated by other peers in a distributed manner. Thus, no third-trusted parties are required to validate the transaction. Once the data are recorded in a block and added to the blockchain, they cannot be altered or deleted. Therefore, users cannot repudiate a transaction that they have generated [18].

2.2.2. IPFS

IPFS is a peer-to-peer distributed file system that allows files to be stored and accessed across a network of nodes. By using content addressing, IPFS uniquely identifies each file in a global namespace [13], Specifically, files are broken down into smaller chunks, hashed, and stored across multiple nodes. Each file is assigned a unique cryptographic hash, making it content-addressable [19]. This ensures that files can be retrieved even if a server goes offline, providing resilience and permanence. Furthermore, IPFS supports file versioning, allowing the storage of historical versions and tracking changes over time [20]. Any user in the network can serve a file by its content address, and other peers in the network can find and request that content from any node via a distributed hash table (DHT) [21,22].

2.2.3. Integration with Blockchain

The integration of blockchain and IPFS provides a robust mechanism for secure data storage and verification. Blockchain guarantees the immutability and timestamping of the data, while IPFS provides efficient and decentralized storage. Specifically, blockchains can store IPFS content addresses within smart contracts or transaction data, allowing for off-chain decentralized storage of large files. By storing only the hash on the blockchain and the actual file on IPFS, blockchain scalability issues [15] are mitigated, as only the critical data are stored on-chain [16].

By combining these technologies, we can achieve a reliable and tamper-proof system for message accountability and integrity in MQTT. In our approach, the broker publishes the root of the Merkle hash tree (MHT) on the blockchain and the MHT itself on IPFS. This ensures that both publishers and subscribers can verify the integrity of the exchanged messages, and both the publisher and the broker cannot repudiate the messages they have sent.

3. Related Works

In this section, we review the existing literature on security problems in MQTT.

3.1. MQTT Security

Due to its lightweight nature, the MQTT standard does not include any default security and privacy features. In the literature, several works highlighted that this is currently an open problem [23–25]. In this section, we briefly survey some of the security issues.

The first issue regards confidentiality. The basic approach consists of using TLS to establish secure channels [26]. However, it negatively impacts performance and energy consumption [27]. Therefore, more advanced solutions have been proposed in the literature to achieve confidentiality [28–35]. Another aspect to consider is authorization [36,37]. A lot of works aimed at achieving both authorization and confidentiality are based on variants CP-ABE or KP-ABE tailored to constrained devices [38–41]. Authorization is also integrated with authentication in MQTT [42–44].

Privacy aspects should also be considered in MQTT. Actually, there are a few proposals aimed at achieving privacy. Among these, the study in [45] has the goal of obfuscating the topics with respect to public brokers. Another way to achieve privacy is by guaranteeing the network anonymity of publishers and subscribers by employing known anonymity protocols proposed in the field of anonymous communication networks (ACN) [46–49]. Among these proposals, we mention [8,50]. Ref. [8] proposes a crowds-based protocol in which brokers form a P2P network to relay clients' messages. In [50], the authors propose a solution based on the Tor protocol, in which messages, encrypted in an onion fashion, are routed by brokers.

3.2. Lack of End-to-End Security in MQTT

Concerning security in MQTT, the studies in the literature mostly focus on the client-to-broker segment, while the client-to-client (end-to-end) security in MQTT is typically neglected. Some of the few proposals in this area primarily focus on ensuring end-to-end confidentiality, whose goal is to protect the messages so that an untrusted broker, mediating the communication, cannot access their content.

Among these, some works leverage symmetric key cryptography for securing MQTT communications [32,51]. These solutions often involve protocols that enable authorized MQTT clients to exchange symmetric keys securely. For instance, [32] introduces a lightweight protocol (designed for constrained devices) that achieves both message confidentiality and client authentication. However, [51] focuses on establishing secure end-to-end key exchange mechanisms. This protocol allows MQTT clients to securely exchange keys to ensure that messages can be encrypted and decrypted without relying on a trusted broker.

Several studies have proposed comprehensive frameworks to address both end-to-end authorization and message confidentiality. Refs. [52–55] provide solutions that ensure only authorized clients can access the messages they are authorized to receive. These frameworks often involve complex mechanisms to manage and verify client authorizations, along with a secure key exchange process to guarantee the end-to-end confidentiality of the exchanged messages.

Despite this, these solutions commonly rely on third-party trusted entities to manage keys and enforce authorization policies. Such dependencies can often hinder the adoption of these solutions in real-world scenarios. Furthermore, third-party entities can represent single points of failure [56]. Indeed, if these entities are compromised or unavailable, the client communications may be potentially exposed to unauthorized access [57]. Finally, these solutions may suffer from scalability issues, especially in large-scale IoT deployments where thousands of devices may be interacting simultaneously [58].

On the other hand, the solution we propose in this paper offers end-to-end security guarantees without relying on third-party parties. The only proposal in the literature not relying on such parties is [9]. Similar to [9], our proposal is also aimed at achieving

end-to-end integrity of the flow of messages exchanged via an MHT-based approach. However, differently from [9], our solution also achieves accountability guarantees both on the publisher and the broker sides.

4. Motivations and Use Cases

In this section, we outline the motivations behind our work and describe the use cases where our solution can be adopted. Specifically, our aim is twofold.

- We aim to achieve the integrity of the flow of messages exchanged between publishers and subscribers. This property should be preserved even in the case of an attacker being able to intercept the communication and tamper with it. This attacker can also be placed at the broker.
- We aim to achieve accountability guarantees. Specifically, in our solution, once the integrity of the exchanged flow is proven, the sender of a flow of messages (either a publisher or the broker) cannot deny having sent it. This property, known as non-repudiation, is essential for maintaining accountability.

In MQTT systems, it is important to guarantee the end-to-end integrity of the data flow. This is due to the fact that the presence of the broker in the middle of the communication between publishers and subscribers represents an inherent vulnerability of the system. Therefore, if the broker is compromised or untrusted, it exposes the system to relevant risks.

In many deployment scenarios, brokers may not be controlled by the communicating parties. Instead, they might be located in third-party, potentially untrusted environments (such as cloud providers).

Therefore, an untrusted or malicious broker can intercept, modify, or inject messages between publishers and subscribers. If these attacks go undetected, they could potentially cause severe consequences. Indeed, if the integrity of the message flow is not preserved, subscribers might make decisions based on manipulated data.

For example, MQTT can find applications in the healthcare domain to send patient data from medical devices to monitoring systems. Clearly, the integrity of these data is important, as any modification by a malicious broker could result in incorrect treatment decisions. Another critical application of MQTT is represented by smart cities. In this context, MQTT can be used to enable several services, such as traffic management, environmental monitoring, and so on. Data integrity guarantees that decisions based on this information are accurate and reliable.

We now discuss accountability. In our approach, subscribers who receive data from the broker may want to obtain proof that the received messages are indeed sent by the publisher/broker. With this proof, the subscribers can maintain a verifiable storage for the data.

Without accountability mechanisms, a broker or a publisher could deny having sent certain messages to subscribers to avoid liability or responsibility for the consequences of those messages. For instance, in a stock trading application, a publisher could send fraudulent market data and later deny its actions, causing significant financial losses to subscribers relying on that data. Without accountability, the publisher could deny having sent the data to avoid backlash or financial penalties.

Similarly, on the broker's side, accountability is also important. Brokers often should enforce access control and ensure that only verified publishers are allowed to publish on specific topics. For example, subscribers relying on critical information, such as in financial or health-related applications, need assurances that their data come from verified publishers. By providing accountability on the broker side, our solution guarantees that brokers perform these verification checks. Therefore, they cannot deny having sent a flow of messages produced by unauthorized publishers.

5. The Proposed Approach

In this section, we present a novel approach to achieve accountability and integrity in the MQTT protocol. Specifically, the main goal of this approach is twofold: (1) a

broker/publisher cannot deny having sent messages to subscribers on certain topics; (2) publishers/subscribers can verify the integrity of the data they have sent/received.

Our solution leverages a combination of blockchain technology and the interplanetary file system (IPFS) to provide a verifiable and tamper-proof mechanism for the verification of message integrity.

5.1. Notation

We designate P a publisher and S a subscriber. The broker B manages the topics $T_1, T_2, \ldots, T_n$ to which P publishes messages. The solution operates in temporal rounds R, i.e., time intervals. During a temporal round R_i, P publishes over time messages m on a given topic, say T_j.

5.2. Setup Phase

In this section, we describe the setup phase, along with the smart contract functionalities we employ in our approach.

This smart contract ensures that both publishers and subscribers can verify that the sent/received messages have not been tampered with and that the broker/publisher cannot repudiate having sent those messages. We report the code of the smart contract in Listing 1.

In the following, we refer to the notation described in the previous section. To initialize the whole system, B deploys the smart contract. Thus, B represents the owner of the smart contract. Upon authenticating P, B adds its address to the smart contract via the `addClient` function. This way, the list of all the publishers in the system is stored on the `clients` array. Similarly, the list of all the subscribers interested in the topics of the publisher is stored on this array. When a subscriber wants to join a topic, it sends its Ethereum address to the broker, who then adds it to the `clients` array using the `addClient` function.

The `addRootIPFSAddress` function allows B to map an MHT root to its corresponding IPFS address. This function can be invoked only by the contract owner (i.e., B) who locally computes the MHT root of the data received/sent.

The `getIPFSAddress` function allows for retrieving the IPFS address associated with a given MHT root. If the MHT root, provided as input, is not found in the mapping, then this function returns 0 as the default value.

As we explain in the following sections, this function can be invoked by both publishers and subscribers to check the integrity of the data published/received. Specifically, a publisher can input an MHT root to this function that is locally computed on all the data sent during a temporal round. Subscribers input this function to an MHT root that is provided by the broker. If this function returns the default value, this is a clear indication that the integrity of the data was violated in a temporal round, either in the publisher–broker segment or in the subscriber–broker segment.

Then, our contract provides a function, called `pullTheAlarm`, through which both authorized publishers and/or subscribers can pull the alarm to let others know about the data flow integrity violation.

Both publishers and subscribers can implement mechanisms to capture and react to these events emitted by the smart contract. This can be achieved using libraries such as `web3j` (for Java) or `web3js` (for JavaScript), which provide tools for interacting with smart contracts on the Ethereum blockchain [59].

As explained above, when the `pullTheAlarm` emits an event, both publishers and subscribers are alerted in real-time.

Additionally, in this case, if this function is called by a legitimate publisher or subscriber, the function emits a `ClientAlert` event. To be considered legitimate, the blockchain address of a publisher/subscriber should be present in the `clients` array. To perform this check, the `pullTheAlarm` function invokes the `isClientAddress` internal function.

Listing 1. Solidity Smart Contract for handling the mapping between the root of the MHT and the IPFS address.

```
pragma solidity >=0.6.12 <0.9.0;

contract Root_IPFSAddress_Mapping {
// Address of the contract owner, i.e.,~the broker
address public owner;

// Array to store the addresses of publishers/subscribers
address[] public clients;

// Mapping to store the roots and the corresponding IPFS address
mapping(bytes32 => bytes32) private rootToIPFSAddress;

// Event emitted when a lookup made by a publisher/subscribers fails
event ClientAlert(address indexed requester, bytes32 root);

// Modifier to restrict access to the contract owner
modifier onlyOwner() {
require(msg.sender == owner, "Not authorized");
_;
}

// Constructor to set the owner of the contract
constructor() {
owner = msg.sender;
}

// Function to add a new root-IPFSAddress mapping (only callable by the owner)
function addRootIPFSAddress(bytes32 root, bytes32 IPFSAddress) public onlyOwner {
rootToIPFSAddress[root] = IPFSAddress;
}

// Function to add a new publisher/subscriber address (only callable by the owner)
function addClient(address client) public onlyOwner {
clients.push(client);
}

// View function to retrieve the IPFSAddress associated with a given root
function getIPFSAddress(bytes32 root) public view returns (bytes32) {
bytes32 IPFSAddress = rootToIPFSAddress[root];
return IPFSAddress;
}

// Function to emit an event signaling the integrity violation
function pullTheAlarm(bytes32 root) public {
if (isClientAddress(msg.sender)) {
emit ClientAlert(msg.sender, root);
}
}

// Internal function to check if an address is in the clients array
function isClientAddress(address addr) internal view returns (bool) {
for (uint i = 0; i < clients.length; i++) {
if (clients[i] == addr) {
return true;
}
}
return false;
}
}
```

5.3. Communication Phase

For each topic T_j on which P publishes, both P and B construct a *hashchain* as follows. We denote by $H_{t-1}^{T_j}$ the hashchain built so far. When P publishes a message m_t on topic T_j, the broker concatenates m_t with the current hashchain $H_{t-1}^{T_j}$ (the hash value of the previous chain) and computes the hash:

$$H_t^{T_j} = \text{Hash}(m_t \| H_{t-1}^{T_j})$$

Here, $\|$ denotes the concatenation operation, and $H_t^{T_j}$ represents the current hash value after including m_t.

At the end of each temporal round R_i, both P and B proceed with the construction of a Merkle hash tree (MHT) M_i using the hashchains of all topics on which P has published during R_i. We explain this process in the following.

For each topic T_j, the final hash value of the hashchain at the end of the round, say $H_{\text{end}}^{T_j}$, is used as a leaf node in the MHT.

Therefore, given the set of the topics $\{T_1, T_2, \ldots, T_n\}$ on which P publishes, the hash values $H_{\text{end}}^{T_1}, H_{\text{end}}^{T_2}, \ldots, H_{\text{end}}^{T_n}$ are the leaves of the MHT.

To build the MHT, consecutive pairs of hash values are hashed together to form parent nodes. For example, two consecutive leaves $H_{\text{end}}^{T_i}$ and $H_{\text{end}}^{T_{i+1}}$ are concatenated and hashed to form their parent node, say PN_i:

$$PN_i = \text{Hash}(H_{\text{end}}^{T_i} \| H_{\text{end}}^{T_{i+1}})$$

This process continues iteratively, building up the tree until a single root hash Root_{M_i} is obtained at the top of the tree.

In the meantime, all the subscribers proceed as follows. Consider a subscriber S interested in a subset of topics published by P, denoted as $\{T_j, T_{j+1}, \ldots, T_{j+k}\} \subseteq \{T_1, T_2, \ldots, T_n\}$. During the round R_i, S receives messages on these topics from the broker B and constructs the corresponding hashchain. At the end of the round, S receives the root of the MHT from B.

In the next section, we describe how the publisher and the subscribers can detect any integrity violation in the published flow of messages.

5.4. Verification Phase

Once the communication phase is completed, the verification of message integrity for the current temporal round R_i begins. This phase involves several steps performed by the broker B and the clients (the publisher P and all the subscribers S).

B proceeds as follows. At the end of the round R_i, it first publishes the Merkle hash tree M_i on IPFS. As described in Section 2, in IPFS, each stored object is identified by a unique hash. We call this address IPFS_Addr_{M_i}. Then, B invokes the `addRootIPFSAddress` function on the smart contract with the following inputs: Root_{M_i} and IPFS_Addr_{M_i}. The smart contract stores this pair, thus creating a permanent link between the MHT root and its storage location on IPFS.

P proceeds as follows. At the end of the round R_i, P independently constructs a Merkle hash tree M_i^* of the messages exchanged during R_i with B. Thus, P locally computes its root hash $\text{Root}_{M_i}^*$ and verifies it against the roots recorded in the smart contract. Specifically, P invokes the `getIPFSAddress` function on the smart contract with the locally computed root $\text{Root}_{M_i}^*$. The function returns the related IPFS address if a matching root is found. If the returned IPFS address is 0, it indicates that the root $\text{Root}_{M_i}^*$ computed by P does not match any root stored by the broker. In this case, the smart contract emits a `PublisherAlert` event to signal there is an integrity violation, i.e., the messages published by the broker are different from those received by the subscribers. This way, all the subscribers who monitor these events in real-time are immediately notified of the integrity violation in the message stream for the round R_i.

Then, also, each subscriber S proceeds with the verification phase. S verifies that the root hash $\text{Root}_{M_i}^{**}$ received from B is stored in the blockchain by invoking the `getIPFSAddress` function on the smart contract. If no match is found on the blockchain, the smart contract emits the `SubscriberAlert` event notifying all participants of this. If the root hash matches (i.e., $\text{Root}_{M_i}^{**} = \text{Root}_{M_i}$), S proceeds to verify the integrity of the computed hashchains. S requests from B the missing hash values necessary to construct the path from the final hash value of the local hashchain to the root Root_{M_i}. In the case that S cannot reconstruct at least one of these paths, it invokes the `getIPFSAddress` function with a deliberately incorrect root hash (e.g., 0) to trigger an alert for all participants, ensuring they are aware of the issue. This

way, all monitoring participants (publishers and subscribers) are immediately notified of the integrity violation.

This verification phase ensures robust end-to-end integrity guarantees within the MQTT framework. Indeed, during each round time, both publishers and subscribers can check whether the messages sent by the publisher have not been tampered with while in transit.

5.5. Final Remarks

As final remarks, we highlight the main advantages of our solution by motivating the design choices we adopted.

We start making some considerations regarding the storage required by our solution.

The main information that our solution needs to store is the hashchains built at each round. We recall that, during each round, publishers (subscribers, respectively) compute a hashchain for each topic they publish to (are interested in). Similarly, the broker computes a hashchain for each topics to which publishers are publishing. Once an entity in the system sends or receives a message, it computes the current value of the hashchain by computing the hash of this message with the last stored hashchain. At this point, the message is no longer needed and thus can be discarded, as well as the older value of the hashchain. Therefore, at the end of the round, an entity in the system needs to locally store just a digest per topic. Additionally, after performing the integrity checks, the digests can be discarded. Just as an example, if the entities manage 1000 topics, only 32 KB of storage are necessary.

We highlight, as analyzed in Section 7, that computing the hashchain on the fly does not affect the throughput of the system. Indeed, the time typically required to compute the hash (which is needed to update the hashchain value) is much lower than the time between the generation of two consecutive packets. Therefore, the computation of the hashchain does not affect the sending rate and thus the throughput of the system.

To conclude this section, we discuss the benefits of combining hashchains with MHT in our solution rather than building an MHT directly on the messages.

We observe that in our solution, the leaves of the MHT are represented by the hashchains, which are in the same number as the topics of a publisher. On the other hand, if the MHT is built directly on the messages, the leaves would be in the same number as all the messages published in a round time (by such a publisher). This would result in a higher number of hashes necessary to build the MHT. Additionally, this would result in a higher number of hashes when verifying the path from the leaves to the MHT's root.

More formally, we denote by $|T|$ the number of topics of a publisher and $|M|$ the number of messages that it publishes during a time round. Clearly, it holds that $|T| < |M|$. Therefore, if the leaves are the number of topics, the MHT computation requires the computation of $2 \times |T| - 1$ hashes. However, when the leaves are the number of messages, the MHT computation requires the computation of a higher number of hashes, i.e., $2 \times |M| - 1$ hashes.

Additionally, if the leaves are in the number of topics, the height of the MHT is $log|T|$. While, when the leaves are in the number of messages, the height of the MHT is $log|M|$. This impacts the verification process (performed by subscribers) since, in the second case, the number of hashes that need to be computed to verify whether a leaf is part of an MHT is higher. Therefore, combining hashchains with MHT is computationally more efficient. Additionally, for each topic, the subscribers are interested in verifying that the flow of messages published on that topic is intact. As such, verifying the path from the hashchain on that topic to the MHT's root is the fastest way to verify the entire flow on that topic, since it requires just one verification. On the other hand, if the MHT is built on messages, the verification process requires verifying the path from each message on that topic to the MHT's root. Hence, while in the previous case, the number of paths to verify is the number of the topics to which the subscriber is interested, in the second case, the number of paths to verify is the number of messages on those topics, thus making the verification process much more inefficient.

6. Cost Analysis of Smart Contract Execution

In this section, we analyze the costs associated with the functions of our smart contract on the Ethereum blockchain. Additionally, we discuss the economic feasibility and potential business model of our proposed solution.

6.1. Gas Costs Overview

Through this section, we examine the costs of smart contract deployment and the execution of the smart contract functions. We proceeded as follows. We first computed the gas necessary to execute them. We recall that Ethereum transactions and smart contract executions require a fee known as "gas", which measures the computational effort required. Then we computed the cost in US dollars with the following formula:

$$\text{Cost in USD} = \text{Gas} \times \text{Gas price (Gwei)} \times 10^{-9} \times \text{Ether/USD exchange rate}$$

We report in Table 1 the gas price required for executing the deployment and the functions of the smart contract, along with the related price in US dollars (USD). The gas usage is the estimation provided by Remix Compiler (https://remix.ethereum.org/ (accessed on 10 July 2024)). To estimate the cost in USD, we used the above formula considering the variation in the year 2023 gas price and the Ether/USD exchange rate. We then calculated the median values and reported them in Table 1. For a complete view of the fluctuation of the costs (in USD) over the year 2023, the reader can refer to Figures 1 and 2, which report how the costs related to the execution of the smart contract deployment and the smart contract functionalities, respectively, change over the year 2023. We observe that the costs present some fluctuations throughout the year, with a peak in the months of May and June. Overall, excluding this temporary spike, the median cost of executing the smart contract deployment is around USD 23. In contrast, the cost of executing the smart contract functions is around USD 2.

In the following sections, we discuss in detail all the costs related to the execution of our solution.

Table 1. Costs for executing the smart contract functions on the Ethereum blockchain.

Function	Gas Usage	Cost in USD (2023 Median)
`Deploy Contract`	461,322 gas	USD 23
`addClient`	68,401 gas	USD 3.41
`addRootIPFSAddress`	47,016 gas	USD 2.34
`pullTheAlarm`	26,763 gas	USD 1.33
`getIPFSAddress` (view)	-	USD 0

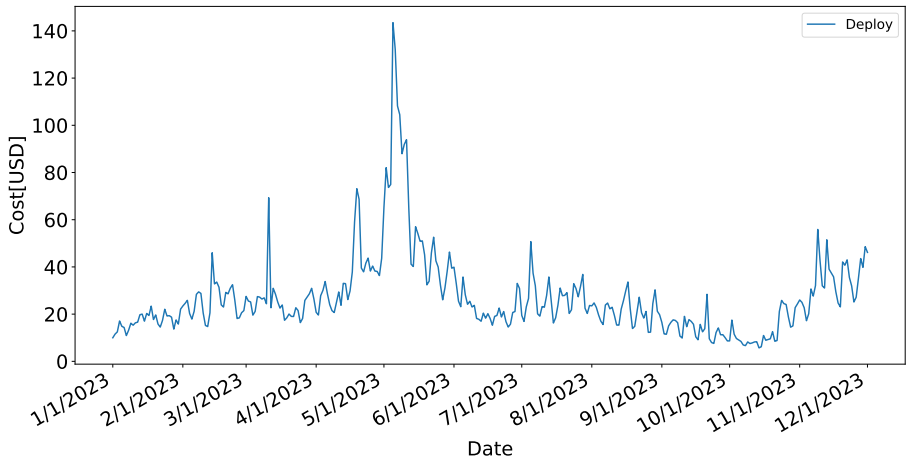

Figure 1. Cost of deploying the smart contract in US dollars over the year 2023.

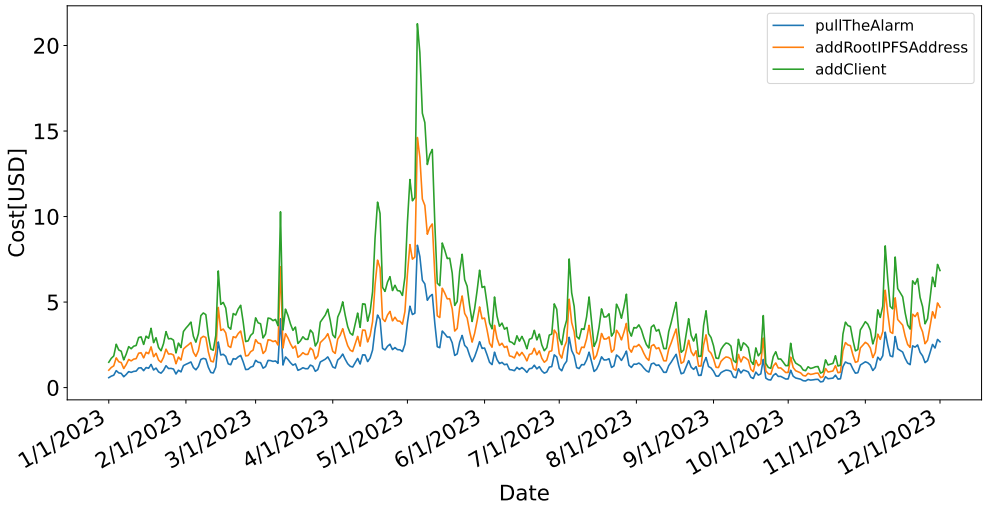

Figure 2. Cost of invoking the smart contract functions in US dollars over the year 2023.

6.2. Cost Breakdown and Business Model Implications

In the following, we discuss the cost of each smart contract function along with its deployment. These costs are reported in Table 1.

Deployment Cost: The cost for deploying the smart contract is 461,322 gas, which translates to approximately USD 23. This deployment is a one-time cost incurred by the broker. Given its single occurrence, it is a manageable and reasonable investment to set up the system.

Adding Clients (`addClient` function): We recall that this function can only be invoked by the broker to include a new publisher or subscriber in the system. Each time this function is invoked, the broker incurs a cost of about 68,401 gas, equivalent to USD 3.41. In our business model, we propose that this cost be covered by customers (publishers or subscribers) as a one-time fee upon joining the system. This cost is not prohibitive, so it is a feasible expense for new users.

Storing Root Hash and IPFS Address (`addRootIPFSAddress` function): We recall that the `addRootIPFSAddress` function can only be invoked by the broker. Specifically, the broker invokes it at the end of each temporal round to store the root hash of the Merkle Hash Tree (MHT) along with its corresponding IPFS address. This function consumes approximately 47,016 gas, resulting in a cost of around USD 2.34 per execution. The broker bears this recurring expense since this function is called at the end of each round. The frequency of these calls depends on the length of the time round: shorter rounds mean more frequent costs, while longer rounds reduce the frequency but delay the validation of message integrity for subscribers.

Integrity Violation Alerts (`pullTheAlarm Function`): We recall that the `pullTheAlarm` function signals a violation of the integrity of the data exchanged. This function costs about 26,763 gas per call, about USD 1.33. This cost can be borne by the publisher or the subscriber reporting the violation. Since this function is called only when an integrity violation is detected, it is not a regular expense, but rather an occasional one. Therefore, the cost appears sustainable

Fetching IPFS Address (`getIPFSAddress` Function): The `getIPFSAddress` function is a view function, which means that it does not alter the state of the blockchain and therefore does not incur any gas costs. This function is regularly used by publishers and subscribers to retrieve the IPFS address of the MHT for verification purposes. Thus, the integrity verification operation has no cost for publishers/subscribers.

6.3. Cost Analysis of System Maintenance per Client

In the previous section, we examined the costs of individual operations on the blockchain in terms of dollars. These operations, carried out by the broker and clients (only the `pullTheAlarm` function), incur certain costs at specific frequencies. In a realistic business model, it is reasonable to assume that the broker passes these costs onto the clients who subscribe to the service, thereby sharing the expenses to amortize the costs. This section evaluates the average annual cost that each client must bear to maintain the system for a year. This is calculated by dividing the total cost by the number of clients.

6.3.1. Cost Calculation Model

The total cost of our solution depends on four parameters:

1. Number of publishers
2. Number of subscribers
3. Frequency of round time (a higher frequency implies invoking the `addRootIPFSAddress` function more often)
4. Frequency of alarms (a higher frequency implies invoking the `pullTheAlarm` function more often)

Since these parameters cannot be fixed in advance and may vary significantly depending on the considered scenario, we present analyses showing the variation of these four parameters over wide ranges to assess the cost.

Figures 3–5 show the annual cost per client as the number of publishers and subscribers varies. Each figure represents three different alarm frequencies: one alarm per day per client, one alarm per week per client, and one alarm per month per client. Within each figure, there are three plots for three different round frequencies:

- One round per day per publisher (blue)
- One round per week per publisher (orange)
- One round per month per publisher (green)

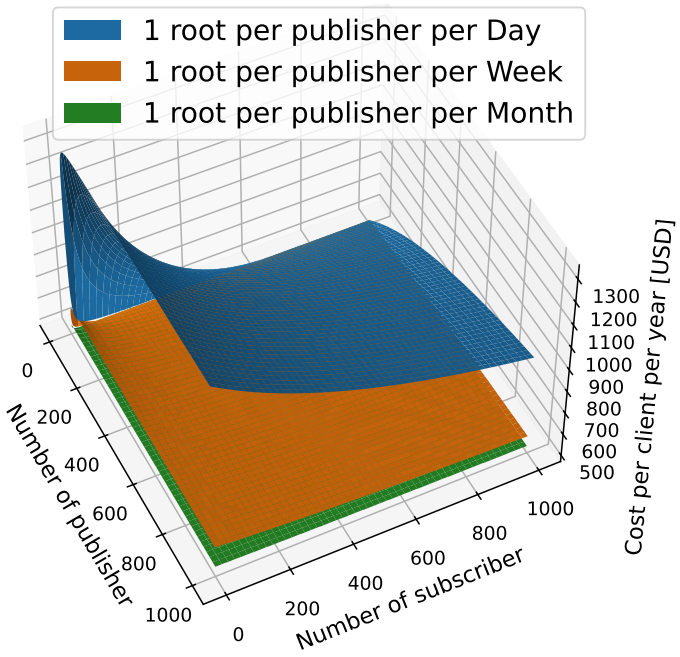

Figure 3. Annual Cost per Client with 1 Alarm per Day.

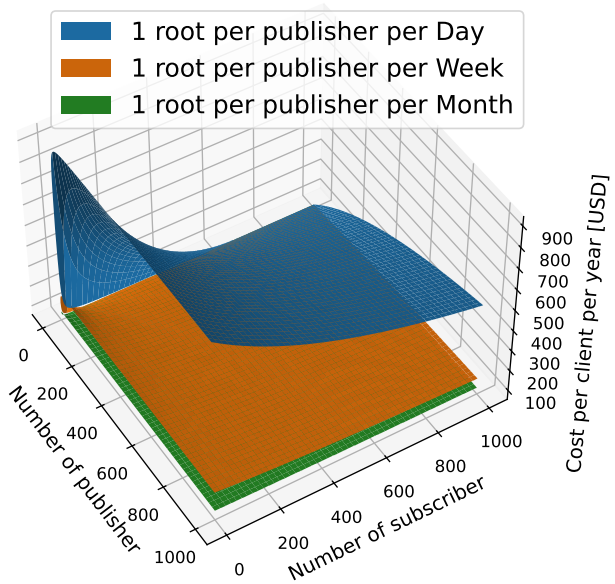

Figure 4. Annual Cost per Client with 1 Alarm per Week.

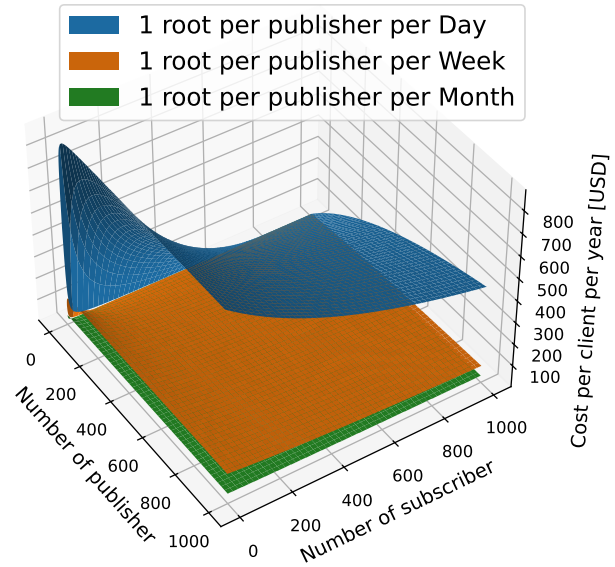

Figure 5. Annual Cost per Client with 1 Alarm per Month.

6.3.2. Discussion

Across all plots, it is evident that the solution scales very well with an increasing number of publishers and subscribers.

Subscribers: Each new subscriber introduces a new cost for the `addClient` and `pullTheAlarm` functions. However, these costs are originated by the subscribers themselves (and then can be sustained by them). The other costs (e.g., deploying and storing the root of the Merkle hash tree (MHT) for each round for each publisher) are distributed among all subscribers. Consequently, as the number of subscribers increases, the cost per individual client decreases.

Publishers: For publishers, the introduction of a new cost (due to the storage of the root) is partially offset by the subscribers. The deployment cost is always amortized by the new publishers. As the number of publishers increases, the cost remains nearly constant (with a slight increase) because the cost they introduce is almost fully offset by the publishers themselves. Conversely, as the number of subscribers increases, the per-client cost decreases significantly.

We can conclude that the solution scales very well with the number of clients.

Regarding other factors, as expected, higher round frequencies or higher root publication frequencies result in higher costs. However, even in extreme cases (high-security requirements and a high number of alarms), the cost peaks at around $1200 annually, which can be afforded by a moderately sized organization. Under average conditions, the costs are around $300 annually, akin to a standard subscription fee that a user would pay.

7. Experiments

In this section, we describe our experimental campaign and the results we obtained.

7.1. Experimental Setup

Our campaign involves the deployment of a single MQTT broker and a variable number of MQTT clients.

The MQTT broker was deployed on a standard laptop equipped with an Intel i7-8850 CPU at 1.8 GHz and 16 GB of RAM. We implemented the broker via a customized version of the HiveMQ broker, including the specific functionalities required by our solution.

To implement an MQTT client we employed a Raspberry Pi Pico W device. The latter is equipped with a dual-core ARM Cortex-M0+ processor running at 133 MHz, 256 KB of RAM, and 2 MB of flash memory. The Raspberry Pi Pico is suited to emulate typical MQTT clients due to its limited computational and networking capabilities. Then to observe the scalability of the system, given the unavailability of a high number of physical Raspberry Pi Pico units, we deployed simulated clients (publishers and subscribers) on a standard laptop. The execution times of these simulated clients were calibrated to match those of the actual Raspberry Pi Pico devices.

We considered publishers and subscribers connected to the broker through an internet connection. To simulate realistic network conditions, we incorporated network delays based on real-world data [8].

To simulate the transmission of actual data by publishers, we used a dataset reporting temperature sensor data collected from a building (Accessible at: https://www.kaggle.com/code/koheimuramatsu/iot-temperature-forecasting/input (accessed on 10 July 2024)). In this dataset, each datum was associated with the timings in which it was generated by the sensor. This allowed us to compute a realistic rate of data generation.

We used the Sepolia testnet as the blockchain network for our experiments. Specifically, we used the testnet to deploy the smart contract required by our solution. Then, we used this testnet to measure the timing required to invoke the smart contract functions. Similarly, we used the client from IPFS Desktop App (https://docs.ipfs.tech/install/ipfs-desktop/ (accessed on 10 July 2024)) to measure the timing required to store the MHT on IPFS.

To conclude this section, we summarize the times we obtained in our setup both client-side and broker-side in Table 2.

Table 2. Execution times client-side and broker-side.

Operation	Client-Side	Broker-Side
Hash (SHA-256)	148 ms	58 µs
`addRootIPFSAddress`	–	18.088 s
`getIPFSAddress`	16.331 s	–
IPFS Upload	–	9 ms

7.2. Performance Analysis

In this section, we analyze the performance of the proposed solution by examining the throughput of publishers, subscribers, and the broker. Throughput refers to the rate at which publishers can dispatch locally generated messages to the broker and the rate at which subscribers can consume messages received from the broker. Similarly, on the broker side, the throughput is the rate at which it can send messages to the subscribers.

In our experiments, we measured the normalized throughput with respect to the maximum throughput, which is determined based on the transmitted data's generation rate. The maximum throughput is computed by observing the frequency by which data are generated in our dataset.

In Figures 6 and 7, we report how the normalized throughput varies as the frequency of round times (per day) increases and as the number of topics increases for publishers and subscribers, respectively. Concerning the number of topics, we considered the number of topics on which a publisher publishes.

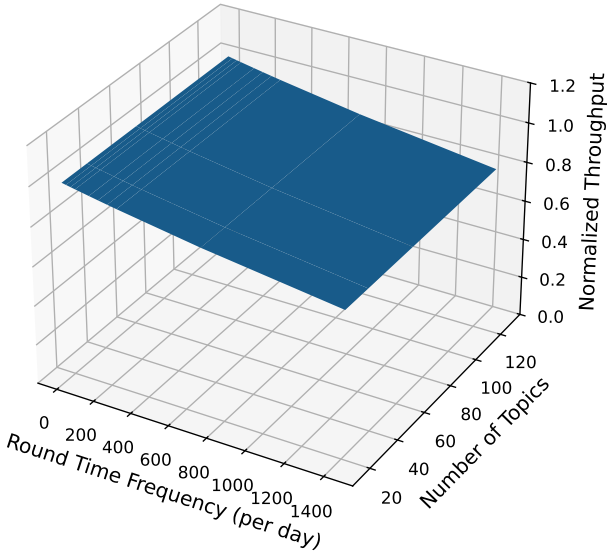

Figure 6. Normalized throughput of publishers.

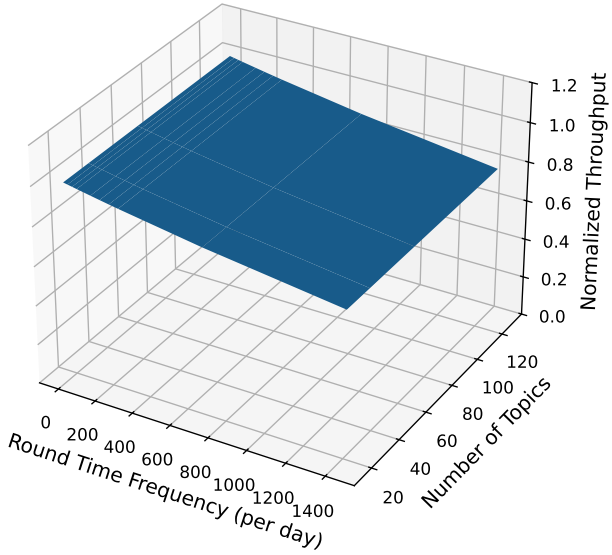

Figure 7. Normalized throughput of subscribers.

The ideal normalized throughput is 1, which represents a scenario in which the system handles all generated messages without any additional overhead. However, as expected, in our experiments, the normalized throughput deviates from 1 as the frequency of rounds increases throughout the day. This deviation occurs because additional operations are required at the end of each round.

Specifically, concerning publishers, this decrease is due to the execution of the following operations at the end of the round: (1) computation of an MHT, (2) verification of the presence of the computed root in the blockchain, to ensure there is no integrity violation on the broker side.

Concerning subscribers, this decrease is due to the execution of the following operations at the end of the round: (1) computation of a path of hashes for each topic, (2) verification of the presence of the computed root in the blockchain (as on the publisher side). Considering (1), we observe that, in general, subscribers may not be interested in the whole set of topics of the publisher. In our experiments, we considered that the number of topics in which the subscriber is interested is, on average, half of the topics of the publisher.

It can be seen from the plot that the increase in the number of topics and, thus, the increase in the time for calculating the MHT, generates differences in the normalized throughput that are not appreciable. In fact, even when the number of topics is in the hundreds, the calculation of the MHT takes less than 1 s. On the other hand, the operation that most affects the reduction in throughput is the invocation of the `getIPFSAddress` function, the execution of which takes more than 15 s on average.

Due to this end-round operation, the normalized throughput deviates from the maximum one. However, for both publishers and subscribers, we observe that the reduction in throughput is at most 20% in the worst-case scenario, where the round time frequency is as high as one round every 60 s (1440 rounds per day). We highlight that this represents an extreme case. Typically, the frequency of round times should be chosen based on the data publication frequency to ensure there are enough messages published within a round. Overall, the higher the round time, the lower the throughput reduction. In our dataset, the data are generally published every 30 s. Therefore, with a round time of 60 s, in each round, only two messages are published. Thus, the introduced overhead reduces the throughput of the clients. On the other hand, if the time round duration is higher than 5 min, we can observe that the reduction of the throughput is lower than 10%.

Figures 8 and 9 report how the normalized throughput of the broker varies as the number of publishers increases and as the frequency of round time (per day) increases, considering 16 topics (in Figure 8) and 128 topics (in Figure 9). To conduct these experiments, we considered a single broker, one subscriber, and an increasing number of publishers (up to 10,000 publishers).

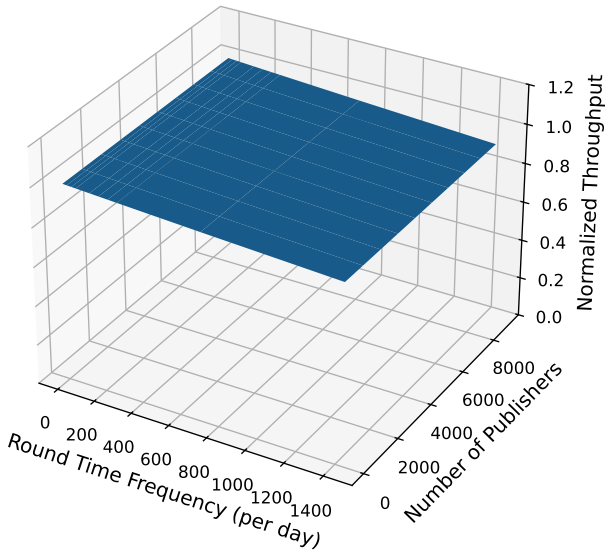

Figure 8. Normalized throughput of the broker, considering 16 topics per publisher.

In Figure 8, where 16 topics are considered, we observe that the increasing number of publishers does not significantly affect the normalized throughput. On the contrary, in Figure 9, where 128 topics are considered, we can see that when the number of publishers increases, the normalized throughput decreases. However, the reduction in normalized throughput, as the number of publishers increases, is minimal, since it always remains below 2%.

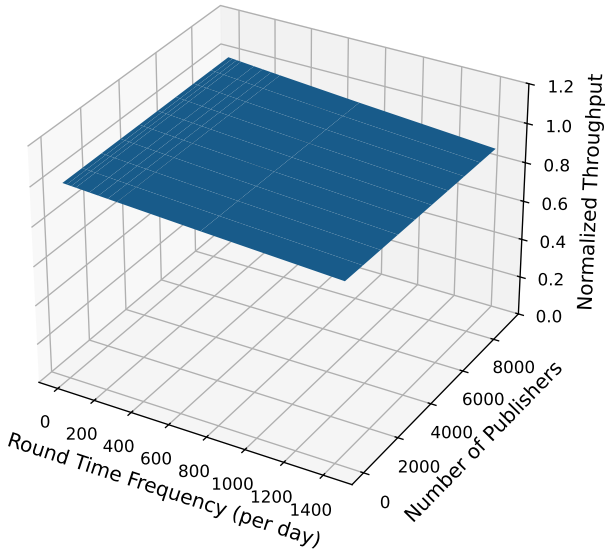

Figure 9. Normalized throughput of the broker, considering 128 topics per publisher.

As the number of publishers increases, the number of MHTs that the broker needs to calculate also increases linearly. Additionally, the time to compute an MHT increases linearly with the number of topics. However, as mentioned earlier, the time to construct an MHT for each client is significantly low due to the nanosecond order of hash calculation time. For instance, if there were 100 million publishers connected to a single broker, the time to calculate all the hashes would be approximately 50 s. However, with such a large number of publishers, the bottleneck would be managing concurrent connections (It is estimated that a HiveMQ broker can support up to 10 million concurrent connections), rather than managing the computations of MHTs.

We observe that as the number of publishers increases, the number of network operations performed on the blockchain and IPFS also increases linearly. We recall that for each publisher, the MHT root must be saved on the blockchain (via the `addRootIPFSAddress` function), and the MHT must be stored on IPFS. However, unlike the CPU-intensive MHT computation operations, network operations are I/O operations and can be managed in parallel. This explains why the linear increase in these operations does not result in a linear decrease in the broker's normalized throughput.

Despite this, the execution of the `addRootIPFSAddress` function requires more than 15 s to complete. Thus, this is the operation that causes the most throughput reduction.

Indeed, as explained before, the operations involving the blockchain generate the highest overhead. This overhead produces a reduction in the throughput that increases linearly with the increase in the time round frequency (per day). However, by choosing appropriate round frequencies, the reduction in throughput can be limited. For example, if the time round duration is higher than 5 min, we can observe that the reduction of the throughput is less than 3%.

We also experimented with how the broker-side throughput is affected by an increasing number of subscribers. Specifically, we considered a setting with a single publisher, one broker, and a growing number of subscribers. This scenario did not produce significant differences in throughput. For each subscriber, at the end of the round, the broker must send the root and the hashes that allow reconstruction of the path from the hash chain (calculated locally by the individual subscriber) to the MHT root (for each topic to which the subscriber is subscribed). However, these hashes and the root can be sent piggybacking on the last message of the round. Moreover, these are network (I/O) operations, making them easily parallelizable. As a result, no substantial variations in throughput are observed with an increasing number of subscribers.

7.3. Discussions

Our experiments show that the proposed solution scales well with the number of clients. Additionally, the experimental results indicate that the proposed solution incurs minimal overhead on the system's throughput. This overhead is primarily due to the additional integrity verification operations required at the end of each round. By choosing an appropriate round time frequency based on the data publication rate, the reduction in throughput can be negligible, ensuring the system's efficiency. We observe that a low round time frequency may limit the applicability of our solution for hard real-time applications [60,61], in which data integrity should be verified in a very short range (in the order of milliseconds). However, typical MQTT applications [62–64] do not have such strong requirements, and thus, our solution is effective in providing data integrity for these applications.

We can conclude that our experiments confirm that the proposed solution is viable for maintaining data integrity without significantly compromising performance.

8. Security Analysis

In this section, we analyze the security of our approach. We start by describing the security assumptions, the adversary model, and the security properties we aim to achieve. Then, we show that under our assumptions, the adversary cannot violate our security properties.

8.1. Security Assumptions

Our security analysis relies on the following assumptions:

- **Account Ownership:** The ownership of Ethereum accounts used by publishers, subscribers, and the broker is guaranteed. We assume there are no impersonation activities, i.e., an attacker cannot take over a participant's account.
- **Hash Function Properties:** The hash functions used for the construction of hashchains and Merkle hash trees (MHT) are collision and pre-image resistant. This means that it is computationally unlikely to find two different inputs that produce the same hash (collision resistance) or to find an input whose hash matches a given output (pre-image resistance).

8.2. Adversary Model

In our model, the adversary can be a malicious broker or any malicious party that can access and tamper with the traffic exchanged by the publishers with the interested subscribers. The latter can do so by performing a man-in-the-middle attack. We observe that the communication between a client (either a publisher or a subscriber) and the broker is often protected via well-known security protocols, such as the TLS protocol [1]. However, when the certificate validation in TLS is not properly implemented the whole communication is susceptible to man-in-the-middle attacks [4,5]. Additionally, we recall that being the communication in MQTT mediated by the broker, the latter can intrinsically perform a man-in-the-middle attack [8].

In our model, we consider that an attacker can tamper with the traffic in the following ways [9]:

- the attacker can modify at least one of the exchanged messages;
- the attacker can inject at least one message in the exchanged flow;
- the attacker can delete at least one of the exchanged messages;
- the attacker can change the order of at least one message in the exchanged flow;

Additionally, in our threat model, we consider that a publisher or a broker can deny having sent a certain flow of messages.

8.3. Security Properties

Through our approach, we aim to achieve the following two security properties:

- **End-to-end data flow integrity:** This concept was first formalized in [9]. Our aim is to achieve not only the integrity of each exchanged message but also the integrity of the entire flow of exchanged messages.
- **Accountability:** This property ensures that publishers and brokers cannot deny having sent a flow of messages after it has been sent.

Concerning the **accountability** property it implies the following:

- the flow of messages published by the publisher/broker can be traced back to it in a verifiable manner.
- the publisher/broker cannot deny the publication of a flow once it has been sent.

Concerning the **end-to-end data flow integrity** property, it was originally defined in a weak sense and in a strong sense [9]. In our analysis, we consider the weak one. Specifically, it consists of the following properties:

- **Completeness:** all the messages sent by a publisher are also received by the subscribers;
- **Correctness:** all the messages received by the subscribers are correct, in the sense that, for instance, there are no new messages added by an attacker, i.e., messages that are not present in the original message flow sent by the publisher;
- **Weak Liveness:** the order of the messages received by the subscribers is preserved.

8.4. Achieving Security Properties

In the following, given our security assumptions, we show that the adversary cannot break the identified security properties.

8.4.1. Preserving End-to-End Data Flow Integrity

The objective of our approach is to preserve end-to-end data flow integrity in the sense that when this property cannot be preserved (i.e., an attacker has violated the integrity of the exchanged flow), publishers and subscribers can acknowledge this.

As formally demonstrated in [9], this property cannot be satisfied when at least one of the attacks identified in Section 8.3 is performed. Thus the goal of an attacker is to perform one of the identified attacks without being detected.

Suppose an attacker is able to tamper with the flow of messages exchanged during a temporal round on a given topic by performing one of the attacks identified in Section 8.3. Then, this modified flow is received by the subscribers. However, once a subscriber builds the hashchain for that topic, this hashchain differs from the one computed publisher-side. Then the subscriber has to reconstruct the path from the hashchain to the root of the MHT sent by the broker to it. In order for the attack to be undetected, no event should be emitted by the smart contract. Therefore, the attacker should be able to forge a path from the modified hashchain to the root actually computed publisher-side and published on the blockchain.

However, due to the **hash function properties** assumption, an attacker cannot be able to do so. Indeed the **hash function properties** assumption ensures that the adversary cannot alter the messages in the hashchain or MHT without detection. Moreover, it cannot construct a valid path from an arbitrary hashchain to the actual MHT root.

This ensures that a violation of the end-to-end data flow integrity property cannot go undetected.

8.4.2. Preserving Accountability

The use of blockchain ensures that the data are stored immutably and can be retrieved for verification. Once the root of the MHT is on the blockchain, it cannot be changed. Similarly, the address of the MHT stored on IPFS remains unchanged. This ensures that the integrity of the MHT is maintained and can be independently verified.

Due to this, the broker cannot deny having sent a flow of messages once it has been included in the MHT. The cryptographic proofs provided by the hashchain and MHT (see

assumption **hash function properties**) ensure that the message was part of the published set, thereby achieving accountability. Additionally, due to the assumption **account ownership**, a broker cannot deny having published a certain root.

Similarly, also, the publisher cannot deny having computed a certain root from the flows of messages sent. This is because once this root is computed independently from the broker, it has to send a transaction invoking a function of the smart contract with its computed root as input. This transaction is immutably stored on the blockchain and it can be publicly verifiable.

Therefore, if a subscriber wants to provide proof of having received a certain flow of messages (in a given temporal round) from the publisher, it simply has to provide the path from the related hash chain to the root written in the transaction executed by the publisher.

9. Conclusions

In this paper, we introduced an approach to ensure accountability and end-to-end data flow integrity in MQTT by leveraging blockchain and IPFS technologies. The core idea consists of building an MHT of the data exchanged between publishers and subscribers and then verifying the computed root of the MHT. The integration of blockchain with IPS enhances the scalability of the proposed approach since only the essential data (i.e., the root of the MHT) are stored on the blockchain, while the MHT itself is stored on IPFS.

The blockchain effectively prevents brokers and publishers from repudiating messages to create immutable and verifiable logs of all communications. Additionally, our solution ensures that when an attacker is able to modify the flow of messages in transit, publishers and subscribers can detect this integrity violation. Differently from the majority of the solutions in the literature concerning end-to-end security in MQTT, our proposal does not rely on trusted third parties. Additionally, it enhances the MQTT protocol with accountability guarantees.

In our study, we provided a security analysis to show that under standard cryptographic assumptions, our approach guarantees both accountability and end-to-end data flow integrity. Additionally, we performed a thorough experimental campaign to study the impact of our solution on MQTT. Our experimental results indicate that the proposed solution incurs minimal overhead on the system's throughput. The overhead is due to the additional integrity verification operations required at the end of each round. However, by choosing an appropriate round time frequency, the reduction in throughput can be negligible. Our results also show that our solution scales well with an increase in the number of clients and, thus, does not introduce any scalability issue.

Finally, we provided a detailed analysis of the costs of our solution by also showing an estimation of the costs per year required by our solution. Overall, the execution costs are relatively low, especially given the critical security and accountability benefits our solution provides. Additionally, from our analysis, it emerges that the more subscribers in the system, the lower the costs per client. This demonstrates that our solution scales very well with an increasing number of clients in the system.

The future work will focus on optimizing the system by incorporating a methodology to estimate the best round time frequency into the solution. This could be achieved by dynamically adjusting the round time frequency (during communication between clients) to better match the frequency of data exchange.

Author Contributions: Conceptualization, F.B. and S.L.; methodology, F.B. and S.L.; software, S.L.; validation, S.L.; formal analysis, S.L.; investigation, F.B. and S.L.; resources, S.L.; data curation, S.L.; writing—original draft preparation, S.L.; writing—review and editing, S.L.; visualization, S.L.; supervision, F.B.; project administration, F.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MQTT	Message queuing telemetry transport
IPFS	Interplanetary file system
QoS	Quality of service
MHT	Merkle hash tree
DHT	Distributed hash table
P2P	Peer-to-peer
TLS	Transport layer security
ACN	Anonymous communication network
CP-ABE	Cipertext-policy attribute-based encryption
KP-ABE	Key-policy attribute-based encryption
USD	United States dollars

References

1. Lazzaro, S.; De Angelis, V.; Mandalari, A.M.; Buccafurri, F. Is Your Kettle Smarter Than a Hacker? A Scalable Tool for Assessing Replay Attack Vulnerabilities on Consumer IoT Devices. In Proceedings of the 2024 IEEE International Conference on Pervasive Computing and Communications (PerCom), Biarritz, France, 11–15 March 2024; pp. 114–124. [CrossRef]
2. Maskeliūnas, R.; Damaševičius, R.; Segal, S. A Review of Internet of Things Technologies for Ambient Assisted Living Environments. *Future Internet* **2019**, *11*, 259. [CrossRef]
3. Li, J.; Maiti, A.; Fei, J. Features and Scope of Regulatory Technologies: Challenges and Opportunities with Industrial Internet of Things. *Future Internet* **2023**, *15*, 256. [CrossRef]
4. Lupia, F.; Lucchese, M.; Merro, M.; Zannone, N. ICS Honeypot Interactions: A Latitudinal Study. In Proceedings of the 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy, 15–18 December 2023; pp. 3025–3034.
5. Lucchese, M.; Lupia, F.; Merro, M.; Paci, F.; Zannone, N.; Furfaro, A. HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems. In Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES '23, New York, NY, USA, 29 August–1 September 2023. [CrossRef]
6. OASIS. MQTT Version 5.0 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed on 10 July 2024).
7. Gündoğan, C.; Kietzmann, P.; Lenders, M.; Petersen, H.; Schmidt, T.C.; Wählisch, M. NDN, CoAP, and MQTT: A comparative measurement study in the IoT. In Proceedings of the 5th ACM Conference on Information-Centric Networking, ICN '18, New York, NY, USA, 21–23 September 2018; pp. 159–171. [CrossRef]
8. Buccafurri, F.; De Angelis, V.; Lazzaro, S. MQTT-A: A Broker-Bridging P2P Architecture to Achieve Anonymity in MQTT. *IEEE Internet Things J.* **2023**, *10*, 15443–15463. [CrossRef]
9. Buccafurri, F.; De Angelis, V.; Lazzaro, S. MQTT-I: Achieving End-to-End Data Flow Integrity in MQTT. *IEEE Trans. Dependable Secur. Comput.* **2024**, 1–18. [CrossRef]
10. Panwar, A.; Bhatnagar, V. Distributed Ledger Technology (DLT): The Beginning of a Technological Revolution for Blockchain. In Proceedings of the 2nd International Conference on Data, Engineering and Applications (IDEA), Bhopal, India, 28–29 February 2020; pp. 1–5. [CrossRef]
11. Hamilton, M. Blockchain distributed ledger technology: An introduction and focus on smart contracts. *J. Corp. Account. Financ.* **2020**, *31*, 7–12. [CrossRef]
12. Kushwaha, S.S.; Joshi, S.; Singh, D.; Kaur, M.; Lee, H.N. Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access* **2022**, *10*, 6605–6621. [CrossRef]
13. Benet, J. Ipfs-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
14. Daniel, E.; Tschorsch, F. IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 31–52. [CrossRef]
15. Sanka, A.I.; Cheung, R.C. A systematic review of blockchain scalability: Issues, solutions, analysis and future research. *J. Netw. Comput. Appl.* **2021**, *195*, 103232. [CrossRef]
16. Zheng, Q.; Li, Y.; Chen, P.; Dong, X. An Innovative IPFS-Based Storage Model for Blockchain. In Proceedings of the 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 704–708. [CrossRef]
17. Toldinas, J.; Lozinskis, B.; Baranauskas, E.; Dobrovolskis, A. MQTT Quality of Service versus Energy Consumption. In Proceedings of the 2019 23rd International Conference Electronics, Palanga, Lithuania, 17–19 June 2019; pp. 1–4. [CrossRef]
18. Sun, J.; Yao, X.; Wang, S.; Wu, Y. Non-Repudiation Storage and Access Control Scheme of Insurance Data Based on Blockchain in IPFS. *IEEE Access* **2020**, *8*, 155145–155155. [CrossRef]

19. Anthal, J.; Choudhary, S.; Shettiyar, R. Decentralizing File Sharing: The Potential of Blockchain and IPFS. In Proceedings of the 2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT), Gharuan, India, 5–6 May 2023; pp. 773–777. [CrossRef]
20. Khatal, S.; Rane, J.; Patel, D.; Patel, P.; Busnel, Y. FileShare: A Blockchain and IPFS Framework for Secure File Sharing and Data Provenance. In *Proceedings of the Advances in Machine Learning and Computational Intelligence*; Patnaik, S., Yang, X.S., Sethi, I.K., Eds.; Springer: Singapore, 2021; pp. 825–833.
21. Bawane, H.; Shinde, T.; Kadam, A.; Budukh, Y.; Mundhe, P. Ethegram-an ethereum and ipfs-based decentralized social network system. *Int. Res. J. Eng. Technol.* **2020**, *7*, 1978–1982.
22. Buccafurri, F.; De Angelis, V.; Idone, M.F.; Labrini, C. A protocol for anonymous short communications in social networks and its application to proximity-based services. *Online Soc. Netw. Media* **2022**, *31*, 100221. [CrossRef]
23. Patel, C.; Doshi, N. A novel MQTT security framework in generic IoT model. *Procedia Comput. Sci.* **2020**, *171*, 1399–1408. [CrossRef]
24. Perrone, G.; Vecchio, M.; Pecori, R.; Giaffreda, R. The Day After Mirai: A Survey on MQTT Security Solutions After the Largest Cyber-attack Carried Out through an Army of IoT Devices. In Proceedings of the IoTBDS, Porto, Portugal, 24–26 April 2017; pp. 246–253.
25. Mendez Mena, D.; Papapanagiotou, I.; Yang, B. Internet of things: Survey on security. *Inf. Secur. J. Glob. Perspect.* **2018**, *27*, 162–182. [CrossRef]
26. Dierks, T.; Allen, C. The TLS Protocol Version 1.0. *RFC* **1999**, *2246*, 1–80.
27. Prantl, T.; Iffländer, L.; Herrnleben, S.; Engel, S.; Kounev, S.; Krupitzer, C. Performance impact analysis of securing MQTT using TLS. In Proceedings of the ACM/SPEC International Conference on Performance Engineering, Virtual Event, France, 19–23 April 2021; pp. 241–248.
28. Mathews, S.P.; Gondkar, R.R. Protocol Recommendation for Message Encryption in MQTT. In Proceedings of the 2019 International Conference on Data Science and Communication (IconDSC), Bangalore, India, 1–2 March 2019; pp. 1–5. [CrossRef]
29. Su, W.T.; Chen, W.C.; Chen, C.C. An Extensible and Transparent Thing-to-Thing Security Enhancement for MQTT Protocol in IoT Environment. In Proceedings of the 2019 Global IoT Summit (GIoTS), Aarhus, Denmark, 17–21 June 2019; pp. 1–4. [CrossRef]
30. Ahamed, J.; Zahid, M.; Omar, M.; Ahmad, K. AES and MQTT based security system in the internet of things. *J. Discret. Math. Sci. Cryptogr.* **2019**, *22*, 1589–1598. [CrossRef]
31. Shin, S.; Kobara, K.; Chuang, C.C.; Huang, W. A security framework for MQTT. In Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS), Philadelphia, PA, USA, 17–19 October 2016; pp. 432–436. [CrossRef]
32. Sadio, O.; Ngom, I.; Lishou, C. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 119–123. [CrossRef]
33. Dinculeană, D.; Cheng, X. Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices. *Appl. Sci.* **2019**, *9*, 848. [CrossRef]
34. Oak, A.; Daruwala, R. Assessment of Message Queue Telemetry and Transport (MQTT) protocol with Symmetric Encryption. In Proceedings of the 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Jalandhar, India, 15–17 December 2018; pp. 5–8. [CrossRef]
35. Iyer, S.; Bansod, G.V.; Naidu, P.; Garg, S. Implementation and Evaluation of Lightweight Ciphers in MQTT Environment. In Proceedings of the 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Msyuru, India, 14–15 December 2018; pp. 276–281. [CrossRef]
36. Niruntasukrat, A.; Issariyapat, C.; Pongpaibool, P.; Meesublak, K.; Aiumsupucgul, P.; Panya, A. Authorization mechanism for MQTT-based Internet of Things. In Proceedings of the 2016 IEEE International Conference on Communications Workshops (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; pp. 290–295.
37. Michaelides, M.; Sengul, C.; Patras, P. An Experimental Evaluation of MQTT Authentication and Authorization in IoT. In Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, New Orleans, LA, USA, 31 January–4 February 2022; pp. 69–76.
38. Gupta, V.; Khera, S.; Turk, N. MQTT protocol employing IOT based home safety system with ABE encryption. *Multimed. Tools Appl.* **2021**, *80*, 2931–2949. [CrossRef]
39. Bisne, L.; Parmar, M. Composite secure MQTT for Internet of Things using ABE and dynamic S-box AES. In Proceedings of the 2017 Innovations in Power and Advanced Computing Technologies (i-PACT), Vellore, India, 21–22 April 2017; pp. 1–5.
40. Mendoza-Cardenas, F.; Leon-Aguilar, R.S.; Quiroz-Arroyo, J.L. CP-ABE encryption over MQTT for an IoT system with Raspberry Pi. In Proceedings of the 2022 56th Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, USA, 9–11 March 2022; pp. 236–239. [CrossRef]
41. Liao, T.L.; Lin, H.R.; Wan, P.Y.; Yan, J.J. Improved Attribute-Based Encryption Using Chaos Synchronization and Its Application to MQTT Security. *Appl. Sci.* **2019**, *9*, 4454. [CrossRef]
42. Calabretta, M.; Pecori, R.; Vecchio, M.; Veltri, L. MQTT-Auth: A token-based solution to endow MQTT with authentication and authorization capabilities. *J. Commun. Softw. Syst.* **2018**, *14*, 320–331. [CrossRef]
43. Bhawiyuga, A.; Data, M.; Warda, A. Architectural design of token based authentication of MQTT protocol in constrained IoT device. In Proceedings of the 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), Lombok, Indonesia, 26–27 October 2017; pp. 1–4.

44. Bali, R.S.; Jaafar, F.; Zavarasky, P. Lightweight authentication for MQTT to improve the security of IoT communication. In Proceedings of the 3rd International Conference on Cryptography Security, and Privacy, Kuala Lumpur, Malaysia, 19–21 January 2019; pp. 6–12.
45. Fischer, M.; Kümper, D.; Tönjes, R. Towards improving the Privacy in the MQTT Protocol. In Proceedings of the 2019 Global IoT Summit (GIoTS), Aarhus, Denmark, 17–21 June 2019; pp. 1–6. [CrossRef]
46. Dingledine, R.; Mathewson, N.; Syverson, P. *Tor: The Second-Generation Onion Router*; Technical Report; Naval Research Laboratory: Washington, DC, USA, 2004.
47. Piotrowska, A.M.; Hayes, J.; Elahi, T.; Meiser, S.; Danezis, G. The loopix anonymity system. In Proceedings of the 26th Usenix Security Symposium (Usenix Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 1199–1216.
48. Buccafurri, F.; De Angelis, V.; Idone, M.F.; Labrini, C.; Lazzaro, S. Achieving Sender Anonymity in Tor against the Global Passive Adversary. *Appl. Sci.* **2022**, *12*, 137. [CrossRef]
49. Reiter, M.K.; Rubin, A.D. Crowds: Anonymity for Web transactions. *ACM Trans. Inf. Syst. Secur.* **1998**, *1*, 66–92. [CrossRef]
50. Protskaya, Y.; Veltri, L. Broker Bridging Mechanism for Providing Anonymity in MQTT. In Proceedings of the 2019 10th International Conference on Networks of the Future (NoF), Rome, Italy, 1–3 October 2019; pp. 110–113. [CrossRef]
51. Lee, H.; Lim, J.; Kwon, T.T. MQTLS: Toward Secure MQTT Communication with an Untrusted Broker. In Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea, 16–18 October 2019; pp. 53–58. [CrossRef]
52. Dahlmanns, M.; Pennekamp, J.; Fink, I.B.; Schoolmann, B.; Wehrle, K.; Henze, M. Transparent End-to-End Security for Publish/Subscribe Communication in Cyber-Physical Systems. In Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems, SAT-CPS '21, New York, NY, USA, 28 April 2021; pp. 78–87. [CrossRef]
53. Hamad, M.; Finkenzeller, A.; Liu, H.; Lauinger, J.; Prevelakis, V.; Steinhorst, S. SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation. *IEEE Internet Things J.* **2023**, *10*, 3384–3406. [CrossRef]
54. Pallickara, S.; Pierce, M.; Gadgil, H.; Fox, G.; Yan, Y.; Huang, Y. A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems. In Proceedings of the 2006 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 28–29 September 2006; pp. 215–222. [CrossRef]
55. Borcea, C.; Gupta, A.B.D.; Polyakov, Y.; Rohloff, K.; Ryan, G. PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption. *Future Gener. Comput. Syst.* **2017**, *71*, 177–191. [CrossRef]
56. Yang, H.; Guo, Y.; Guo, Y. Blockchain-based cloud-fog collaborative smart home authentication scheme. *Comput. Netw.* **2024**, *242*, 110240. [CrossRef]
57. Küpçü, A. Distributing trusted third parties. *SIGACT News* **2013**, *44*, 92–112. [CrossRef]
58. Luntovskyy, A.; Globa, L. Performance, Reliability and Scalability for IoT. In Proceedings of the 2019 International Conference on Information and Digital Technologies (IDT), Zilina, Slovakia, 25–27 June 2019; pp. 316–321. [CrossRef]
59. Buccafurri, F.; De Angelis, V.; Lazzaro, S. A Blockchain-Based Framework to Enhance Anonymous Services with Accountability Guarantees. *Future Internet* **2022**, *14*, 243. [CrossRef]
60. Profanter, S.; Tekat, A.; Dorofeev, K.; Rickert, M.; Knoll, A. OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. In Proceedings of the 2019 IEEE International Conference on Industrial Technology (ICIT), Melbourne, VIC, Australia, 13–15 February 2019; pp. 955–962. [CrossRef]
61. Lazzaro, A.; D'Addona, D.M.; Merenda, M. Comparison of Machine Learning Models for Predictive Maintenance Applications. In *Proceedings of the Advances in System-Integrated Intelligence*; Valle, M., Lehmhus, D., Gianoglio, C., Ragusa, E., Seminara, L., Bosse, S., Ibrahim, A., Thoben, K.D., Eds.; Springer: Cham, Switzerland, 2023; pp. 657–666.
62. Lakshminarayana, S.; Praseed, A.; Thilagam, P.S. Securing the IoT Application Layer from an MQTT Protocol Perspective: Challenges and Research Prospects. *IEEE Commun. Surv. Tutor.* **2024**. [CrossRef]
63. Lachtar, A.; Val, T.; Kachouri, A. Elderly monitoring system in a smart city environment using LoRa and MQTT. *IET Wirel. Sens. Syst.* **2020**, *10*, 70–77. [CrossRef]
64. Georgi, N.; Le Bouquin Jeannès, R. Proposal of a health monitoring system for continuous care. In Proceedings of the 2017 Fourth International Conference on Advances in Biomedical Engineering (ICABME), Beirut, Lebanon, 19–21 October 2017; pp. 1–4. [CrossRef]