MQTT-I: Achieving End-to-End Data Flow Integrity in MQTT

Francesco Buccafurri, *Member, IEEE,* Vincenzo De Angelis, and Sara Lazzaro

Abstract—MQTT has become the de facto standard in the IoT. Although standard MQTT lacks built-in security features, several proposals have been made to address this gap. Unfortunately, no existing proposal aims to offer end-to-end data flow integrity in the threat model of untrusted broker. Consider that, the broker has a privileged role, since it is in the middle of communication between publishers and subscribers. Our paper attempts to bridge this gap by introducing a new protocol called MQTT-I, which achieves end-to-end data flow integrity. Our solution is inspired by approaches based on Merkle Hash Trees, commonly used in the context of outsourced data to guarantee data integrity. Our solution aligns with the specific nature of MQTT, in which: (1) publishers and subscribers dynamically join and leave the system, (2) the decoupling principle holds, meaning that publishers and subscribers do not establish any form of agreement, and (3) data, whose integrity should be protected, are multi-topic streams. Moreover, the proposed solution allows us to find the right balance between performance and security. We perform both theoretical and experimental analysis to demonstrate that the introduced security features come with an acceptable overhead in terms of computational and energy cost.

Index Terms—MQTT security, data flow integrity, IoT security, Merkle Hash Tree, malicious MQTT broker

1 Introduction

The Message Queuing Telemetry Transport (MQTT) protocol [1] has become the de-facto standard in the Internet of Things (IoT). It finds application in various domains [2], such as asset tracking and management [3], smart city [4], home automation [5], eHealth [6], in-vehicle infotainment [7], and so on. MQTT is based on the publish/subscribe model. An MQTT broker acts as a central server, collecting messages sent on various topics by publishers and distributing them to interested subscribers. This model keeps the MQTT client implementation simple, shifting system complexities to the broker [8]. An important property of the publish/subscribe model is the *decoupling* of senders and receivers [9], [10] allowing publishers to send messages without knowing the interested subscribers.

Despite its popularity, MQTT lacks built-in security functionalities. Indeed, MQTT is designed for devices with low processing power, and by default, the protocol aims to minimize the overhead needed to exchange messages [11].

While a considerable effort has been devoted to providing client-to-broker protection, to the best of our knowledge, no approach has been proposed in the literature to achieving end-to-end data flow integrity in the threat model of untrusted broker. Nevertheless, this issue is highly relevant, as the architecture of MQTT itself induces a realistic threat model. The MQTT broker sits in the middle between clients and could potentially access, drop, or tamper with all the exchanged messages, which, by design, are transmitted in the clear.

- *F. Buccafurri and S. Lazzaro are with the DIIES Dept., University Mediterranea of Reggio Calabria, Via dell'Universita 25, 89124 Reggio Calabria, Italy*
  E-mail: {bucca, sara.lazzaro}@unirc.it
- *V. De Angelis is with the DIMES Dept., University of Calabria, Via P.Bucci, 87036 Arcavacata di Rende, Cosenza, Italy*
  E-mail: vincenzo.deangelis@dimes.unical.it

In this paper, we propose a new protocol, called *MQTT-I(ntegrity)*, to achieve end-to-end data flow integrity. By data flow integrity, we mean not only ensuring the integrity of individual messages but also guaranteeing the completeness, correctness, and liveness [12] of the client-to-client exchanged data flow as a whole.

It is worth noting that although the problem addressed in this paper shares similarities with the data integrity problem studied in the context of outsourced data [13], there are specific features in the MQTT context that differentiate the two problems, preventing a direct application of the solutions from the former to the latter.

Specifically, as we analyze in detail in Section 5, the combination of the following three aspects make previous approaches not applicable:
(1) The dynamicity of the considered scenario, in which clients exchange a data stream between them and they can leave and join the system at any time.
(2) The existence of the decoupling principle, according to which no agreement can be required between publishers and subscribers.
(3) The fact that data are multiplexed (and then shuffled) across multiple topics.

In fact, the above aspects become the design challenges of our solution. It is based on a non-trivial application of an existing data structure, called Merkle Hash Tree [14], [15], typically used in the context of data outsourcing. To capture the dynamic nature of data, we split the data stream in *rounds*, identified by a suitable time slot. For each round, the broker builds an Merkle hash tree, and, once has provided each subscriber with the right portion (which depends on the topics to which the subscriber is registered) of the Merkle hash tree needed for integrity verification of this round, the broker discards the Merkle Hash Tree, but keeps and publishes only the signed root concatenated with the root of the previous Merkle hash tree, to ensure the

integrity of the entire data stream. Decoupling of publishers and subscribers (possibly joining and leaving the system) is supported, as neither key exchange is required nor separate communication channels. Finally, multiple topics are managed by aggregating the data labeled with the same topic in a single digest forming a leaf of the Merkle Hash Tree. This way, the Merkle Hash Tree is smaller than a single multiple-topic Merkle Hash Tree (in which each message should be verified individually), and all the data labeled with the same topic can be verified by just computing the root of the Merkle Hash Tree. In particular, the size of the Merkle Hash Tree is independent of the number of messages (it depends only on the number of topics).

We carefully analyze the security provided by our solution by formalizing the concept of data flow integrity. Moreover, we evaluate the overhead introduced by our solution (compared to standard MQTT) both theoretically and experimentally. Interestingly, our solution allows us to solve the trade-off between performance (in terms of throughput and energy consumption) and security, by suitably modulating the round time (i.e., the time slot duration).

The structure of the paper is the following. In Section 2, we provide an overview of the literature about end-to-end security in MQTT and data outsourced integrity approaches. In Section 3, we provide some background knowledge concerning the MQTT protocol and Merkle Hash Tree data structure. In Section 4, we formally describe the adversarial model and the security properties we aim to achieve with our solution. The motivations of our work are presented in Section 5 in which we discuss why other approaches are not applicable in our scenario, how our proposal addresses the design challenges, and a real-life example of our solution. In Section 6, we first provide an approach guaranteeing data flow integrity but at a price in terms of efficiency. Then, in Section 7 we discuss an enhancement of the previous approach. Section 8 and Section 9, are devoted to a theoretical analysis of MQTT-I and a comparison with other approaches, respectively. In Section 10, we analyze the performances of our proposal. Then, we analyze the security of the proposed approach in Section 11. Finally, in Section 12, we draw our conclusions.

2 Related Work

In this section, we provide an overview of the main approaches in the literature related to our work.

End-to-end security in MQTT. In the following, we discuss the main proposals aimed at guaranteeing end-to-end security in MQTT. This is still an open research problem, and very few proposals in the literature aim to address it. One of the main goals of these works is to ensure end-to-end confidentiality [10], [16], [17], [18], [19], [20]. These solutions require the provision of a protocol to enable symmetric key exchange between authorized publishers and subscribers. For example, [16] focuses on proposing lightweight solutions to enable message confidentiality and node authentication for constrained devices. Conversely, [19] focuses on describing an end-to-end key exchange protocol. [10], [17], [18], [20] provide a framework to guarantee both end-to-end authorization and message confidentiality. The solutions proposed in these papers leverage third trusted parties to ensure that only authorized subscribers can access the content of exchanged messages. Besides confidentiality, access control is also a topic investigated in the context of MQTT. Among other solutions, a relevant class of proposals is aimed at achieving access control through ciphertext-policy attribute-based encryption (CP-ABE) schemes [21], [22], [23], [24]. Specifically, such schemes allow a publisher to encrypt messages so that only subscribers who meet a certain policy can decrypt them. The main advantage of this approach is that publishers themselves can enforce the policy, without relying on third parties to do so. Moreover, the set of subscribers allowed to decrypt messages does not have to be known in advance, offering the decoupling requirements of the publish/subscribe model. In [25], the authors face the problem of integrity in CP-ABE in the case of revocation of the users from the system. We observe that the solutions employing CP-ABE achieve access control and require a trusted party, called private key generator (PKG), to generate and issue private keys.

Access control is not the focus of this paper. In our threat model, the adversary is a malicious broker, and there are no requirements from the publisher regarding which subscribers can access the messages. However, achieving access control is an orthogonal aspect and current approaches may be combined with our solution. In addition, in our solution, subscribers and publishers can join and leave at any time without any authentication requirement. Similarly to access control, authentication [26], [27] is an orthogonal feature that can be integrated into our solution.

An interesting solution for achieving both access control and end-to-end confidentiality is [28]. However, this approach is based on the presence of trusted entities that delegate in a hierarchical way the access (through keys) to data. Moreover, [28] uses Identity-based Encryption (IBE) and then requires an entity (the highest in the hierarchy) to distribute keys. These assumptions do not hold in our proposal. Finally, [28] does not provide data flow integrity, which is the aim of this paper.

The above-presented works propose security solutions in which the communication is mediated by a potentially untrusted broker. A different scenario is considered in [29]. Specifically, the authors propose addressing the problem of malicious or compromised brokers by confining the broker software to a trusted execution environment (TEE). This measure would prevent malicious behavior by the broker. However, as highlighted by [10], deploying such a solution could be challenging, as TEEs are not available on every server. Furthermore, there are known attacks on TEEs [30]. In general, a realistic threat model cannot assume that brokers are fully trusted.

We observe that all the papers mentioned above rely on trusted parties. In this paper, we consider a more severe threat model since we cannot assume the presence of trusted third parties. Indeed, we consider that all the communications between publishers and subscribers should be mediated by an untrusted broker, and no other party is involved in our protocol. Moreover, the above proposals pursue different goals (i.e., confidentiality and access control).

In this paper, we are interested in a different problem: end-to-end data flow integrity. When we refer to data flow

integrity, we mean not only the integrity of individual messages but also the completeness and correctness of the entire client-to-client data flow.

The above problem presents some similarities with the problem of reaching data integrity in the context of outsourced data [13]. In the following section, we discuss the main solutions proposed in this context.

Outsourced data integrity. Typically, in the context of outsourced data, data are handed over to an untrusted storage entity (e.g., a cloud provider) that can misbehave, for instance, by deleting all or part of the handed data. The class of techniques applied in this context aims to preserve data integrity. Provable data possession (PDP) [31], [32] and Proof of Retrievability (POR) [33], [34] techniques fall into this context. These techniques can be applied when the legitimate owner of the data or a trusted party auditor (TPA) needs to verify the integrity of the outsourced data without retrieving them. However, PDP and POR are not suitable for our scenario because they do not ensure that the data received by the subscribers match the publisher's original data. A class of techniques closer to our context pertains to the query integrity domain [35]. These techniques employ Authenticated Data Structures (ADS) [36], [37] for data maintenance and retrieval. Specifically, through them, a client can query a storage entity to obtain the desired data along with proof of their authenticity (and thus integrity). Generally speaking, since the client is not the legitimate owner of the data, it needs some pieces of information from the owner to be able to verify the proof of integrity provided by the storage entity. Notice that, unlike PDP and POR, the approaches mentioned above enable a client to verify the integrity of the received data, rather than verifying whether the storage entity maintains data integrity. In the context of query integrity, there are some proposals oriented toward data streams, called Streaming Authenticated Data Structures (SADS) [38]. These approaches involve both the verifier and the prover observing the data stream and performing a real-time computation on the observed data. Subsequently, the verifier can obtain a portion of this data from the prover along with proof of its integrity. Then the verifier can check the integrity of the queried data by employing the result of the computation made while observing the stream. In conclusion, approaches related to SADS share a similar goal with ours. However, as demonstrated in Section 9, they are not applicable in the MQTT context.

To conclude this section, we highlight that to the best of our knowledge, our proposal is the first in the literature addressing the data flow integrity problem in the MQTT context. We further observe that our client-to-client solution is not based on the preliminary sharing of secrets between publishers and subscribers, which is, per se, little realistic in open/dynamic scenarios. Additional arguments on this aspect are provided in Section 5.1.

3 Background

In this section, we provide some background notions concerning the MQTT protocol and the Merkle Hash Tree data structure.

3.1 MQTT

MQTT [1] is a client-server publish/subscribe messaging transport protocol that involves two kinds of agents: MQTT clients and MQTT brokers. In turn, MQTT clients can be of two types: publisher (producer of information) and subscriber (consumer of the provided information).

In the MQTT architecture, publishers and subscribers do not communicate directly with each other but rather through an MQTT broker. When a publisher sends a message to the broker, it specifies the information and the topic under which that information should be published. The broker then forwards the published information to all the subscribers interested in that topic.

In MQTT, topics are UTF-8 strings that can include one or more levels separated by a forward slash ('/'). This way, topics can be organized in a hierarchical structure. When a client subscribes to a topic, it can subscribe to a single topic or use *wildcards* to subscribe to multiple topics simultaneously. MQTT allows two different kinds of wildcards: single-level ('+'), which can replace only a single topic level, and multi-level ('#'), which can replace an arbitrary number of topic levels. For instance, suppose that a client subscribes to the topic *temperature/#*, it will receive all the messages published on topics having 'temperature' as a prefix, such as *temperature/dining_room* or *temperature/dining_room/2*.

Concerning MQTT messages, clients can choose the desired Quality of Service (QoS) level. MQTT supports three different Quality of Service (QoS) levels. In detail, **level 0** implies that messages are delivered *at most* once. **level 1** implies that messages are delivered *at least* once. **level 2** implies that messages are delivered *exactly* once.

3.2 Merkle Hash Tree

A Merkle Hash Tree [14] is a data structure that allows for the verification of data integrity. The concept behind it is to hierarchically aggregate different data, potentially intended for different recipients, into a single digest that is signed just once by the owner. Each recipient can efficiently verify the integrity of individual data blocks without necessarily accessing all the data.

It works as follows. Suppose the data are organized into blocks, so that each block can be verified individually by a recipient. A Merkle Hash Tree is a binary tree in which the leaves represent the digests of these individual blocks. The rest of the nodes (internal nodes) are obtained by applying a cryptographic hash function H to the concatenation of their two children. For example, in Figure 1, we consider 8 blocks of data $B_1, \ldots, B_8$. The leaves of the tree are obtained as $H_x = H(B_x)$ for $x = 1, \ldots, 8$. Then, H_1 and H_2 are concatenated and hashed to obtain $H_{12} = H(H_1||H_2)$. Similarly, the other nodes of the tree are obtained. Then, only the root H_R is signed.

Suppose now a recipient wants to verify the integrity of the block B_3. It just needs three nodes of the tree, namely H_4, H_{12}, and H_{58}, to recompute the root. Indeed, the recipient computes in this order $H_3 = H(B_3)$, $H_{34} = H(H_3||H_4)$, $H_{14} = H(H_{12}||H_{34})$, and finally the root as $H_R = H(H_{14}||H_{58})$. Then, the signature of the root is verified. Observe that just a logarithmic number of hash computations (in the number of blocks) are required

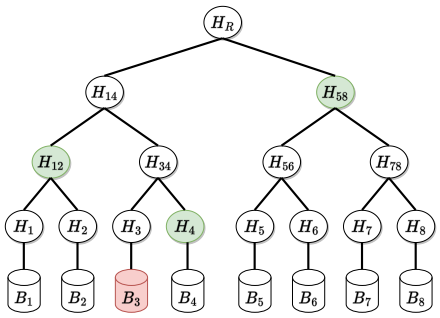


Fig. 1. Merkle Hash Tree with 8 leaves.

for the verification. For simplicity, the example reported in this section considers a Merkle Hash Tree with a number of leaves that is a power of 2. For a more general algorithm, with any number of leaves, see function `MerkleTree` in Algorithm 1.

4 Threat Model

Through this section, we formally describe the adversarial model and the security properties we aim to achieve via our solution. We consider a scenario in which a publisher p publishes messages on a given topic t and a subscriber s receives messages on the topic t. We do not make any assumption on t, then, all the security results obtained are still valid for each topic t through which p and s communicate.

First, we provide some notations.

Let U be the universe of all possible messages. We denote by $M_t^p \subset U$ the (finite) set of messages published by p on the topic t. We denote by $M_t^s \subset U$ the (finite) set of messages received by s on the topic t. We denote by $\overline{M_t^p} = U \setminus M_t^p$ the complement set of M_t^p, i.e., the set of messages not published by p on the topic t. Similarly, we denote by $\overline{M_t^s} = U \setminus M_t^s$ the complement set of M_t^s, i.e., the set of messages not received by s on the topic t. We denote by $\mathcal{T}$ the set of all possible timestamps. We introduce an injective function $f_t^p : M_t^p \rightarrow \mathcal{T}$ that associates each message of M_t^p with a timestamp (representing the sending time of the message). Similarly, we define an injective function $f_t^s : M_t^s \rightarrow \mathcal{T}$ that associates each message of M_t^s with a timestamp (representing the sending time of the message).

We formalize the notion of data flow integrity (on the topic t) by describing the security properties to be satisfied. These properties are borrowed from the domain of query integrity [35], [39].

Completeness CM: All the messages sent by p should be received by s. We formalize the completeness (on the topic t) with a boolean predicate CM_t. We define $CM_t \equiv (M_t^p \subseteq M_t^s)$.

Correctness CR: All the messages received by s should be in the set of messages sent by p. We formalize the correctness (on the topic t) with a boolean predicate CR_t. We define $CR_t \equiv (M_t^s \subseteq M_t^p)$.

Liveness L: The liveness property might be intended in a weak sense (we call it **weak liveness**) or in a strong sense (we call it **strong liveness**). The following definitions take inspiration from [12]. It is easy to realize that this notion of liveness extends the standard notion of **freshness** defined in the query integrity domain [35], [39].

We say that the **weak liveness WL** property is satisfied if the order of the messages published by p and received by s is preserved. We formalize the weak liveness (on the topic t) with a boolean predicate WL_t. We define $WL_t \equiv \forall\ m, m' \in (M_t^p \cap M_t^s)$, it holds $(f_t^p(m) < f_t^p(m')) \iff (f_t^s(m) < f_t^s(m'))$.

We say that the **strong liveness SL** property (over a parameter Δ) is satisfied if the following two conditions hold: (1) the weak liveness property is satisfied, and (2) s receives each message sent at timestamp τ within the deadline $\tau + \Delta$. We formalize the strong liveness (on the topic t) with a boolean predicate SL_t. We define $SL_t \equiv WL_t \wedge (\forall\ m \in (M_t^p \cap M_t^s)$ it holds $f_t^s(m) < f_t^p(m) + \Delta)$.

At this point, we define the **data flow integrity DFI** on the topic t as: $DFI_t \equiv CM_t \wedge CR_t \wedge SL_t$. Observe that DFI_t implies also WL_t (since it is implied by SL_t).

We now formalize the adversarial model. Specifically, we consider an adversary $\mathcal{A}$ that can actively interfere with the execution of the protocol by performing one or more of the following compromises.

C1: $\mathcal{A}$ can modify some messages sent by p. Formally, $C1 \equiv (M_t^p \cap \overline{M_t^s} \neq \emptyset) \wedge (\overline{M_t^p} \cap M_t^s \neq \emptyset)$.

C2: $\mathcal{A}$ can inject some messages in M_t^s. Formally, $C2 \equiv (M_t^p \subset M_t^s)$.

C3: $\mathcal{A}$ can delete some messages originally sent by p. Formally, $C3 \equiv (M_t^s \subset M_t^p)$.

C4: $\mathcal{A}$ can change the order of the messages sent by p. Formally, $C4 \equiv \exists\ m, m' \in (M_t^p \cap M_t^s)$, such that $f_t^p(m) < f_t^p(m')$ and $f_t^s(m) > f_t^s(m')$.

C5: $\mathcal{A}$ can delay the messages sent by p more than a time threshold Δ. Formally, $C5 \equiv \exists\ m \in (M_t^p \cap M_t^s)$ such that $f_t^s(m) > f_t^p(m) + \Delta$.

In Section 11, we show that these compromises are all and only the possible actions $\mathcal{A}$ can perform to break data flow integrity.

Realistically, the adversary $\mathcal{A}$ with the capabilities to perform the above compromises is a malicious (or compromised) broker in the middle of the communication between p and s.

5 Motivations

In this section, starting from the design challenges highlighted in the introduction, we show that the problem addressed in this paper is effective. In other words, a straightforward application of existing techniques is not possible and a new technique becomes necessary. Then, we discuss in an intuitive fashion how we meet the design challenges. Finally, we report a motivating example of application of our proposal.

To help the reading of the section, we report here the design challenges:

Challenge 1: We have to capture the dynamicity of the considered scenario, in which clients exchange a data stream between them and they can leave and join the system at any time.

Challenge 2: The decoupling principle between publishers and subscribers should be satisfied,

Challenge 3: We have to take into account the fact that data are multiplexed across multiple topics.

5.1 Existing approaches

In the following, we discuss why existing approaches cannot be applied to the MQTT domain to reach end-to-end data flow integrity.

A straightforward approach to achieve it is via encryption-based solutions. Observe that, independently of the type of encryption (between symmetric and public-key), a publisher should send a different encrypted flow of data per subscriber, as in our threat model encryption cannot be delegated to the potentially untrusted broker. This would lead to an intolerable overhead, aggravated by **Challenge 1**. Indeed, this approach would introduce an overhead in terms of message size (and thus bandwidth) and computational cost (due to multiple encryptions) for the publisher that grows linearly with the number of subscribers, which can be arbitrarily high. On the contrary, the overhead introduced by our solution is independent of the number of subscribers. Moreover, **Challenge 2** would not be fully addressed. Indeed, the publisher would become non-agnostic with respect to subscribers. Without public-key encryption, a preliminary agreement between publisher and subscriber would be necessary to pre-share keys. This strongly pulls us away from **Challenge 2**. Observe that, a shared group key (shared by the subscribers and publisher) may prevent the publisher from sending a different flow per subscriber. However, any subscriber colluding with the broker may impersonate the publisher and send altered messages to the other subscribers (thus violating the integrity).

Instead, with public-key encryption, the solution would violate the agnostic property of decoupling more than the necessary, thus partially compromising **Challenge 2**. Indeed, the problem itself (i.e., data flow integrity) implies data source authentication due to the assumption of untrustworthiness of the broker, so it is necessary that subscribers need the public key of the publishers, but not vice versa (i.e., it should not be necessary that publishers be aware of subscribers). Unfortunately, any public-key encryption-based solution would require this.

Other approaches in the literature share some similarities with the goals of this paper and belong to the domain of outsourced data integrity. Therefore, it is worth considering whether these approaches can be applied in the MQTT domain. We refer to the approaches identified in Section 2. Regarding PDP and POR techniques, they typically involve the owner or a trusted party (TPA) acting as a verifier, and the storage entity acting as a prover, providing proof of the integrity of the stored data to the verifier. In principle, subscribers could act as TPA, and the broker could act as the untrusted storage entity. Then, subscribers could challenge the broker in order to verify that it has the publisher's original data. However, this approach does not guarantee that what the subscriber receives from the broker actually corresponds to the publisher's original data, which is the primary objective of our proposal. In sum, PDP and POR have different objectives and are not applicable to solving our problem, since they do not take into account the stream of data exchanged between publishers and subscribers (**Challenge 1**).

On the other hand, the class of techniques that employ Streaming Authenticated Data Structures (SADS) to address the query integrity problem shares the most similar objective with ours. Unlike PDP and POR, these techniques aim at verifying the integrity of streaming data received by the verifier. In principle, they could be applied in our domain by letting the subscriber play the role of the verifier and the broker play the role of the prover. However, both **Challenge 2** and **Challenge 3** would be not addressed.

Concerning **Challenge 2**, it happens that all SADS techniques require the existence of a trusted direct channel between the data owner and verifier. In our context, the role of the data owner is played by the publisher, while the role of the verifier is played by the subscriber, Therefore, the existence of such a trusted channel is fully against **Challenge 2**.

Concerning **Challenge 3**, we succinctly state here that SADS are not designed for multi-topic streams and, then, to take into account them, we would introduce a computational disadvantage, with respect to our technique, that is measurable even in terms of asymptotic costs. We dedicate an entire subsection to explain this point.

5.2 Addressing design challenges

Through this section, we discuss how the design challenges introduced at the beginning of this section can be addressed.

To address **Challenge 1**, our approach proceeds by splitting the data streams into chunks exchanged in time rounds in such a way that the integrity of each chunk is verified at the end of the corresponding round. This leaves subscribers free to join the system at any time since they can verify the integrity of the flow of data starting from the next time round. Observe that finding the optimal round duration requires studying a trade-off. The shorter the round time, the higher the security level reached, but it comes at a performance cost. This aspect is analyzed in Section 10.

To address **Challenge 2**, we do not assume the presence of any direct channel between publishers and subscribers. Moreover, our protocol does not require any key exchange between MQTT clients.

Consider now **Challenge 3**, i.e., topic multiplicity. To explain this point, we recall that our approach takes inspiration from the context of outsourced data. In this context, data integrity (also called query integrity) is typically ensured through the construction of a Merkle Hash Tree [14].

The first problem is to understand whether this data structure might introduce unnecessary overhead in our case. Indeed, in the domain of data outsourcing, clients typically need to extract a portion of data (through queries) and verify their integrity (i.e., freshness, correctness, and completeness of the query result). This operation can be efficiently performed if the outsourced data are organized in a Merkle Hash Tree since this data structure allows the integrity verification of even a portion of data without the need to download all the outsourced data.

It is easy to realize that, in the MQTT context, if a client is subscribed to a single topic, then it will receive messages just on that topic from each publisher. Then, it is interested in verifying the completeness, correctness, and liveness of the whole sequence of messages received in a round from each publisher. Therefore, a Merkle-Hash-Tree data structure would introduce unnecessary overhead. Indeed, a hash

chain (per publisher) of the messages sent, equipped with the signature of the publisher of the last digest, would be sufficient to ensure integrity.

Otherwise, in the general case of subscription to multiple topics, two alternatives exist:
(1) The publisher follows the same approach as the case of single-topic subscribers, i.e., it builds a hash chain per topic. It is important to note that, in this case, subscribers do not need to verify portions of the stream of messages, and, consequently, there is no need to adopt a Merkle Hash Tree.
(2) The publisher builds a single hash chain by mixing all the topics to which it publishes. In this case, subscriber-side verification requires a similar approach as data outsourcing, as the subscriber needs to extract from the entire hash chain only the portions regarding the topics to which it is registered. Therefore, the use of a Merkle Hash Tree seems advantageous.

The advantage of (1) is that no extra hash computation arises from the Merkle Hash Tree construction. The advantage of (2) is that each publisher performs only one signature, and each subscriber only verifies one signature. In contrast, in (1), the publisher generates a signature per topic, and the subscriber verifies a signature per topic. Therefore, the choice between (1) and (2) is not straightforward and deserves further study. It should take into account that, in the context of IoT, the required computational power and energy consumption are critical aspects. This is the primary focus of our paper, in which we conclude that the non-trivial solution (2) is the better choice. In Section 6, we present option (1) in detail, which we refer to as the *baseline approach*. Then, in Section 7, we introduce the Merkle Hash Tree-based solution (i.e., option 2), called MQTT-I. In Sections 8 and 10, we compare the two solutions and conclude that MQTT-I is the best solution, thus supporting the main proposal of this paper. Additionally, in Section 10, we also explore the issue of round frequency mentioned earlier, which is an integral part of the complete definition of the MQTT-I protocol.

5.3 Motivating example

To conclude this section, we provide a real-life example to show the relevance of the problem faced in this paper. One of the main areas of applications of MQTT is smart cities [4], [40], [41]. IoT devices, communicating via MQTT, can be used in this context both for ubiquitous sensing and actuation. In a smart city, it is necessary to track various parameters of a city area (such as temperature, humidity, pollution, traffic conditions, and the state of the power grid) to remotely monitor the specific state of that area, and also integrate monitoring devices with other distributed applications that collect data [4] or control actuators [42]. Typically, an MQTT device is equipped with various sensors/actuators or acts as an aggregator for different sensors/actuators [8], [43]. The devices aggregating sensors act as publishers, while the devices aggregating actuators act as subscribers. MQTT topics enabled by a smart city environment can be various. Examples of topic names belonging to the domain of sensing are "environment/air/temperature" or "environment/air/carbonMonoxide" [44], just to give a few. Instead, actuators may operate in areas such as surveillance, smart electricity and water distribution, smart transportation, smart healthcare, and smart infrastructure [42], resulting in a large set of possible topics. In addition to actuators, other examples of subscribers in smart cities include citizens who monitor events using their smartphones [45] or facilities (e.g., emergency facilities) [46].

In this scenario, brokers (possibly placed in the cloud [47], [48]), could collude with parties that have an interest in manipulating, deleting, or shuffling data. Consider for example the case of data coming from sensors that monitor road or electrical infrastructures. Companies responsible for maintenance could have an interest in altering these data. Similar considerations can be made for sensors and actuators related to homeland security, a scenario in which criminal groups or terrorists could have an interest in compromising the integrity of related data flows. For these reasons, it is realistic to consider the threat model in which the broker is untrusted, and can pursue the objective to alter the integrity of data flows.

In the exemplified context, let us discuss now why **data flow integrity** should be intended as defined in Section 4, i.e., as the combination of four security requirements: **completeness**, **correcteness**, **weak liveness**, and **strong liveness**. The need for **completeness** and **correctness** can be trivially explained. Generally speaking, these properties are desired when a given parameter is monitored, to be able to correctly reconstruct the evolution of a process over time. Moreover, this becomes crucial when dealing with safety-critical scenarios. Indeed, suppose a device is monitoring a critical parameter (such as the level of carbon monoxide in a certain area), it is crucial that all the data are delivered intact to the interested subscribers. Indeed, the omission of data (i.e., a compromise of the **completeness** property) or tampered data (i.e., a compromise of the **correctness** property) can lead to safety incidents. A similar reasoning can be made concerning **weak liveness**. Indeed, changing the order of the MQTT packets (i.e., a compromise of the **weak liveness** property) can obviously affect the understanding of the evolution of a process over time. This may also lead to safety incidents. Consider, for instance, a device monitoring the temperature of a given area. If in that area something wrong is happening (e.g., a fire), observing an increasing temperature over time can help trigger the emergency facility on time. On the contrary, if an attacker can reverse the MQTT packet order, it may appear that an anomalous situation (e.g., a high-temperature value) is returning to normal. Finally, ensuring that MQTT packets are not delayed over a chosen time window (i.e., **strong liveness**) is a desirable feature. Indeed, while for periodic monitoring, the absence of a message at a given time can be detected as an anomaly, this is not true in the case of event-based monitoring. For example, consider a sensor detecting traffic incidents that notifies a central operative. In this case, no periodic traffic is expected from the monitoring device (but only in the case of an incident). **Strong liveness** ensures that the integrity of the notification is verified within a fixed time. In general, **strong liveness** gives strict guarantees about the time window in which a certain phenomenon is observed. Obviously, this is helpful in gaining a complete understanding of the observed phenomenon.

6 BASELINE APPROACH

In this section, we describe the baseline solution introduced Section 5.2, denoted as option (1), in which no Merkle Hash Tree is adopted and integrity is ensured through a simple hash chain.

We introduce the notation that will be used throughout the remainder of the paper.

Each publisher p owns a pair of public and private keys (PB^p, PR^p) and is identified by the hash of its public key, i.e., $ID^p = H(PB^p)$. We denote by T the set of all the topics of the system. We denote by $T^p \subseteq T$ the set of topics on which a publisher p publishes data. Finally, we denote by $T^s \subseteq T$ the set of topics to which a subscriber s subscribes. We assume that T^p and T^s are sorted according to the lexicographic order of the topics. Then, t^p_x (t^s_y,resp.) denotes the x-th (y-th, resp.) topic of the set T^p (T^s, resp.).

Our solution performs in rounds of duration RT configurable according to the security and performance requirements. The underlying idea of our approach is that during each round, a flow of data is sent by publishers and, at the end of the round, the integrity of this flow can be verified by each subscriber. We want to stress that each publisher can choose a different round time RT.

Publisher operations. For each topic, p maintains a sequence number and a digest.

Each time p wants to send a message m on the topic $t^p_x \in T^p$, it performs as follows.

First, p publishes the message m on the topic $t^p_x/ID^p/SN^p[x]$, where $SN^p[x]$ is the current sequence number associated with the topic t^p_x. The value $SN^p[x]$ is increased by 1.

Then, p updates the current digest $D^p[x]$ associated with the topic t^p_x with the value $H(m||D^p[x])$. In other words, p builds a hash chain for each topic so that the integrity of the entire message flow in each round can be verified by checking the integrity of the last digest.

At the end of the round, p performs the following operations.

For each topic $t^p_x \in T^p$, p retrieves the associated digest $D^p[x]$ and binds it with the current timestamp τ^p, by computing a signature σ^p_x.

Then, p publishes $D^p[x]||\tau^p||\sigma^p_x$ on a dedicated topic named $t^p_x/ID^p/signature$, where $signature$ is just a fixed string used to notify the subscriber that the received message on the topic t^p_x is a signature for verifying data integrity.

Subscriber operations. We assume s has previously subscribed to each topic in T^s using the wildcard mechanism, i.e., s subscribes to $t^s_y/\#$ for each $t^s_y \in T^s$. Each subscriber s maintains, for each topic $t^s_y \in T^s$, a list of digests and a list of sequence numbers. Each list associated with a topic contains an entry for each publisher on that topic.

Each time s receives a message m on the topic $t^p_x/ID^p/SN^p[x]$, it proceeds as follows.

First, s retrieves the index y of the topic $t^s_y \in T_s$ such that $t^s_y = t^p_x$.

Then, s checks if the sequence number $SN^p[x]$, reported in the topic, is equal to $SN^s[y][ID^p]$, which is the sequence number associated with the topic t^s_y and the publisher p. If this is not the case, s should temporarily store m while waiting for a message with the expected sequence number (further details are provided in Section 7). Then, $SN^s[y][ID^p]$ is increased by one.

Finally, s updates the value $D^s[y][ID_p]$ in the list of digests (associated with the topic t^s_y and the publisher p) with $H(m||D^s[y][ID_p])$. In other words, it recomputes the same chain of digests as the publisher p.

At the end of the round for the publisher p, s will receive the message $D^p[x]||\tau^p||\sigma^p_x$ labeled with the topic $t^p_x/ID^p/signature$.

s will perform the following operations.

First, s checks if τ^p is equal to its local timestamp modulo a given threshold δ, to take into account different time basis between p and s and any possible network delay.

Then, s verifies the signature σ^p_x of $D^p[x]||\tau^p$. This ensures the integrity of the data flow received from p on the topic t^p_x up to this point.

We point out that s may not verify the signature at each round, thus waiting for subsequent rounds for signature verification. This leaves the subscriber free to choose the time interval between two successive verifications. Indeed, since the hash chain built at each round is linked to the previous round, the last signature ensures the integrity of all the previous rounds.

To conclude this section we highlight that new subscribers may freely enter the system at any time with the only price that they can verify the integrity of the data flow starting from the next round of each publisher. Indeed, they have to wait for the next $D^p[x]$ provided by the publisher to compute the correct hash chain.

7 MQTT-I: THE PROPOSED APPROACH.

The main drawback of the baseline approach is that both publishers and subscribers have to compute/verify a signature for each topic (per round) they are interested in. In this section, we introduce a more sophisticated technique, called *MQTT-I*, that improves upon the baseline approach.

The idea behind MQTT-I is to aggregate all the digests associated with the topics (publisher-side) and perform an aggregate signature. This approach reduces the number of signature computations to just one per round for each publisher. Likewise, on the subscriber side, there is only one signature verification per round for each publisher.

The most effective way to aggregate different topics is by constructing a Merkle Hash Tree (see Section 3.2) on the publisher side. Indeed, this requires a linear number of hash computations (as opposed to a linear number of signatures) in the number of topics.

Similarly, concerning subscribers, the computational effort moves from a linear number of signature verifications in the number of topics (for each publisher) to a logarithmic number of hash computations in the number of topics of the publishers. A detailed discussion of the computational complexity of MQTT-I and the baseline approach is provided in Section 8.

As a preliminary observation, we highlight that, to guarantee the integrity of the data flow, at least QoS 1 must be adopted in MQTT. Indeed, by definition, QoS 0 does not ensure message delivery.

We enter into the details of MQTT-I.

7.1 Publisher operations

Each publisher p maintains two arrays D^p and SN^p, both of length $|T^p|$, which contain a digest and sequence number, respectively, for each topic in T^p. D^p and SN^p are initialized with default values (e.g., all values set to 0). We assume that each topic is associated with a position of the array according to its order in the set T^p, so that $D^p[x]$ (or $SN^p[x]$) represents the x-th element of the array and is associated with the topic t_x^p.

Along with the arrays SN^p and D^p, each publisher p maintains a variable pr to store the root of the Merkle Hash Tree computed in the previous round. This root will be used to link the current round with the previous round.

Each time p wants to send a message m on the topic $t_x^p \in T^p$, it performs as follows.

- p publishes the message m on the topic $t_x^p/ID^p/SN^p[x]$ and increments $SN^p[x]$ by 1. The introduction of the sequence number is necessary to guarantee message ordering and prevent duplicate messages. Indeed, as observed in [49], even though QoS 2 guarantees single message delivery with no duplicates, it does not guarantee message ordering.
- p updates the current value of $D^p[x]$ with $H(m||D^p[x])$.

At the end of the round, p performs as follows.

- p builds a Merkle Hash Tree by using the digests contained in the array D^p as leaves. We consider an *in-place* algorithm to reduce the amount of memory required by our solution. This algorithm is described in the function `MerkleTree` reported in Algorithm 1. This function returns the current root of the tree, say cr.
- p signs $H_r = H(pr||cr||\tau^p)$ with its private key PR^p. We denote this signature by σ^p.
- $H_r||\tau^p||\sigma^p$ is published on the topic $ID^p/signature$, where $signature$ is just a fixed string included in the topic name used to notify the subscriber that the received message is a signature to verify data integrity.
- The previous root pr is updated with H_r, and D^p and SN^p are restored to the default value.

Observe that the broker forwards the root of the tree to the subscriber (according to the decoupling principle of MQTT). However, the signature of the publisher ensures that no entity can tamper with such a root.

Algorithm 1: Publisher p (MQTT-I)

```
for t_x^p ∈ T^p do
    SN^p[x] ⟵ 0 ;
    D^p[x] ⟵ 0 ;
pr ⟵ 0 ;
while Data to send are available do
    m, t_x^p ⟵ ObtainData();
    ProcessAndSendData(m, t_x^p);
    if Round Time is expired then
        EndRound();
Function ProcessAndSendData(m,t_x^p):
    MQTTPublish(m, t_x^p/ID^p/SN^p[x]);
    SN^p[x] ⟵ SN^p[x] + 1;
    D^p[x] ⟵ H(m||D^p[x]);
Function EndRound():
    τ^p ⟵ getCurrentTime();
    cr ⟵ MerkleTree();
    H_r ⟵ H(pr||cr||τ^p);
    σ^p ⟵ Sign(H_r, PR^p);
    MQTTPublish(H_r||τ^p||σ^p, ID^p/signature);
    pr ⟵ H_r;
    for t_x^p ∈ T^p do
        SN^p[x] ⟵ 0 ;
        D^p[x] ⟵ 0 ;
Function MerkleTree():
    size ⟵ |T^p| − 1 ;
    do
        z ⟵ 0;
        for (i ⟵ 0; i <= size; i ⟵ i + 2) do
            if i = size then
                D^p[z] ⟵ D^p[i];
                break;
            D^p[z] ⟵ H(D^p[i]||D^p[i + 1]);
            z ⟵ z + 1;
        if size is even then
            size ⟵ size/2 ;
        else
            size ⟵ (size − 1)/2 ;
    while size >= 1;
```

7.2 Broker changes

Each subscriber, to verify the integrity of the data published by p, needs to recompute the root of the tree.

Observe that signature verification is not a straightforward operation for subscribers since they are not typically subscribed to all the topics of a publisher. Then, a proper mechanism should be implemented.

To recompute the root of the Merkle Hash Tree, subscribers need the digests forming the paths from the leaves of the tree to the root (see Section 3.2). These paths may be computed by the publisher itself and provided to the subscribers. However, it is more efficient to delegate this task to the broker since it is less resource-constrained and has higher computational power (compared to the publishers).

Then, we require some changes in the broker. In detail, for each publisher p, the broker maintains an array D^p (the same as maintained by p) to store the hash chains associated with the topics in T^p. When the broker receives $H_r||\tau^p||\sigma^p$ on the topic $ID^p/signature$, it does not immediately publish this message, but it first performs the following operations.

It computes the Merkle Hash Tree to obtain the root cr. However, differently from the publisher, when the broker builds the tree, it stores, for each leaf, a sequence of digests that allows a subscriber to recompute the root starting from such a leaf. We denote by L_x^p, the list of digests enabling the computation of the root starting from the leaf associated with the topic t_x^p (i.e., $D^p[x]$). Furthermore, along with L_x^p, the broker stores a sequence of bits B_x^p, which is of the same length as L_x^p, that indicates whether the elements of L_x^p should be concatenated to the left or right to recompute the root. We assume that bit "0" means "left-concatenation" and bit "1" means "right-concatenation".

For example, consider Figure 1 in Section 3.2. Suppose H_3 is the leaf of the tree associated with the topic t_x^p. In this case, $L_x^p = (H_4, H_{12}, H_{58})$ and $B_x^p = (0, 1, 0)$. Finally, for each topic $t_x^p \in T^p$, the broker publishes $L_x^p||B_x^p$ on the topic $t_x^p/ID^p/path$, where $path$ is a fixed string. Then, it publishes the message of p, $H_r||\tau^p||\sigma^p$, on the topic $ID^p/signature$.

7.3 Subscriber operations

We discuss the procedure followed by subscribers.

Each subscriber s maintains two arrays D^s and SN^s, each of length $|T^s|$. D^s and SN^s will contain a list of digests and sequence numbers, respectively, for each topic in T^s. The length of each list (for each topic) may be different. Specifically, the list of digests (or sequence numbers) associated with the topic $t_y^s \in T^s$ will contain an entry for each publisher on that topic. Formally, $\forall\, t_y^s \in T^s$, the length of the associated list will be $|\{p : t_y^s \in T^p\}|$. Each entry of the list is indexed by the identifier ID^p of a publisher p. We denote by $D^s[y][ID^p]$ ($SN^s[y][ID^p]$, resp.) the element of D^s (SN^s, resp.) associated with the topic t_y^s and the identifier ID^p of p.

s also locally stores an additional array, say R^s of length equal to the number of publishers whose topics are of interest for s. Formally, the length of R^s is $|\{p : T^s \cap T^p \neq \emptyset\}|$. R^s is indexed by the identifiers of the publishers and contains the previous roots provided by publishers.

We assume s has previously subscribed to each topic in T^s using the wildcard mechanism, i.e., s subscribes to $t_y^s/\#$ for each $t_y^s \in T^s$. Furthermore, s subscribes to the topic $ID^p/signature$ for each of p such that $T^s \cap T^p \neq \emptyset$. This topic prevents s from receiving the signature of the publishers multiple times.

Three types of MQTT messages can be received by s.

First, s can receive a message m on the topic $t_x^p/ID^p/SN^p[x]$. In this case, it performs as follows:

- s retrieves the index y of topic $t_y^s \in T_s$ such that $t_y^s = t_x^p$.
- s checks if the sequence number $SN^p[x]$, reported in the topic, is equal to $SN^s[y][ID^p]$. If this is not the case, s should temporarily store m while waiting for a message with the expected sequence number (more details are provided in Section 7). We use another list to store possible out-of-order messages. Nevertheless, we want to point out that this case is very uncommon, hence the size of this list is expected to be very small. This list can be efficiently implemented as a hash map using the sequence number received in the topic (i.e., $SN^p[x]$) as the key and the message and the topic itself (i.e., $m, t_x^p/ID^p/SN^p[x]$) as the value. This operation is performed in Function `ProcessReceivedData` in Algorithm 2.
- s increments $SN^s[y][ID^p]$ by one.
- s updates the value $D^s[y][ID_p]$ with $H(m||D^s[y][ID_p])$.

In the second case, s receives a message in the form $L_x^p||B_x^p$ on the topic $t_x^p/ID^p/path$.

When s receives this message:

- s retrieves the index y of the topic $t_y^s \in T_s$ such that $t_y^s = t_x^p$.
- Starting from $D^s[y][ID^p]$ (corresponding to a leaf of the Merkle Hash Tree of p), s computes the root of the tree by leveraging L_x^p and B_x^p, as described in the function `ComputeRoot` in Algorithm 2. This root will be stored in $D^s[y][ID^p]$.

In the third case, s receives the message $H_r||\tau^p||\sigma^p$ on the topic $ID^p/signature$. It performs as follows:

- s checks if the τ^p is equal to its local timestamp modulo a given threshold δ, to take into account different time basis between p and s and any possible network delay.
- s retrieves the previous root associated with p, i.e. $R^s[ID^p]$.
- s needs to retrieve the current root cr associated with p. This root is stored in D^s. Observe that, since this root is unique for ID^p, s checks that $D^s[y][ID^p]$ contains the same value (i.e., cr) for each possible y associated with the same ID^p.
- s computes $H_s = H(R^s[ID^p]||cr||\tau^p)$ and verifies the signature σ^p of H_s
- Finally s updates $R^s[ID^p]$ with H_s.

Observe that, if multiple data are provided to s by the same publisher on different topics, the signature can be verified just once while the paths are verified for each topic. However, the computational effort required to recompute the root is negligible compared to the effort required for signature verification.

Finally, we would like to highlight the following concerning the chunk-based mechanism implemented by MQTT-I. We recall that MQTT-I works by slicing the data flow into chunks by setting a time period (called round time). For each chunk, MQTT-I requires the construction of a Merkle Hash Tree, whose number of leaves is equal to the number publisher's topic. Indeed, each leaf (associated then with a given topic) consists of a hash chain including all the packets belonging to that chunk tagged with that topic. As a result, the size of the Merkle Hash Tree is not dependent on the size of the chunk (i.e., the round-time duration and the number of messages exchanged within that interval), because the number of leaves coincides with the number of topics. Additional details on the cost derived from the Merkle Hash Tree construction are reported in Section 8. The entire procedure followed by the subscriber is summarized in Algorithm 2.

8 Computational Overhead

Through this section, we analyze the computational overhead introduced by the baseline approach and MQTT-I compared to the standard MQTT protocol (which does not offer end-to-end integrity).

We introduce some notations.

We recall that the most common hash functions follow the Merkle–Damgård construction [50] so that the input message is split into blocks of fixed size B (e.g., for SHA256, $B = 64$ bytes). The total time required to compute a digest is linear in the number of blocks forming the message to hash. We denote by C_H the time required to perform a hash computation on a block of size B. Then, to compute a hash on a message of size M, the required time is $\frac{M}{B} \cdot C_H$.

We denote by C_S the time to compute a digital signature and by C_V the time to verify the digital signature. Observe that the input of the signature computation and verification procedure is equal to the dimension of a digest and does not depend on other factors. Once performing the computation, in both the baseline approaches and MQTT-I, the signature (along with a digest and a timestamp) has to be published through MQTT. We denote by C_P this publication time.

The other operations (assignments, concatenations, comparisons, and so on) are negligible compared to the previous ones. This is also confirmed by the experiments performed in Section 10. Finally, we denote by M the size of the MQTT messages transferred by the publishers to the subscribers.

We start by analyzing the computational overhead of each publisher.

Algorithm 2: Subscriber s (MQTT-I)

```
for t_y^s ∈ T^s do
    for p ∈ {p : t_y^s ∈ T^p} do
        SN^s[y][ID^p] ⟵ 0 ;
        D^s[y][ID^p] ⟵ 0 ;
tempList ⟵ ∅; R^s ⟵ ∅;
while Data to read are available do
    m*, t* ⟵ ReceiveMQTTData();
    if t* = t_x^p/ID^p/SN^p[x] then
        ProcessReceivedData(m*,t*);
    if t* = ID^p/signature then
        EndRound(m*, t*);
    if t* = t_x^p/ID^p/path then
        ComputeRoot(m*, t*);
Function ProcessReceivedData(m, t*):
    t_x^p, ID^p, SN^p[x] ⟵ splitTopic(t*) ;
    y ⟵ findIndex(t_x^p);
    if SN^p[x] = SN^s[y][ID^p] then
        SN^s[y][ID^p] ⟵ SN^s[y][ID^p] + 1;
        D^s[y][ID^p] ⟵ H(m||D^s[y][ID^p]);
        while tempList.contains(SN^s[y][ID^p]) do
            m, t* ⟵ tempList.get(SN^s[y][ID^p]) ;
            t_x^p, ID^p, SN^p[x] ⟵ splitTopic(t*) ;
            y ⟵ findIndex(t_x^p);
            SN^s[y][ID^p] ⟵ SN^s[y][ID^p] + 1;
            D^s[y][ID^p] ⟵ H(m||D^s[y][ID^p]);
    else
        tempList.add(SN^p[x], m, t*);
Function EndRound(m*, t*):
    ID^p ⟵ splitTopic(t*); τ^s ⟵ getCurrentTime() ;
    τ^p, σ^p ⟵ splitMessage(m*)
    if |τ^s − τ^p| > δ then
        return ERROR;
    cr ⟵ D^s[0][ID^p];
    for t_y^s ∈ T^s do
        if D^s[y][ID^p] ≠ cr then
            return ERROR;
    H_s ⟵ H(R^s[ID^p]||cr||τ^p);
    if Verify(σ^p, H_s, PB^p) then
        R^s[ID^p] ⟵ H_r;
        for t_y^s ∈ T^s do
            SN^s[y][ID^p] ⟵ 0 ;
            D^s[y][ID^p] ⟵ 0 ;
        return SUCCESS;
    else
        return ERROR;
Function ComputeRoot(m*, t*):
    t_x^p, ID^p ⟵ splitTopic(t*) ;
    y ⟵ findIndex(t_x^p);
    L_x^p, B_x^p ⟵ splitMessage(m*)
    for (i ⟵ 0, i < |B_x^p|, i ⟵ i + 1) do
        if B_x^p[i] == 0 then
            D^s[y][ID^p] ⟵ H(D^s[y][ID^p]||L_x^p[i]);
        else
            D^s[y][ID^p] ⟵ H(L_x^p[i]||D^s[y][ID^p]);
```

In both the baseline approach and MQTT-I, this overhead is introduced during two phases: (i) the message transmission and (ii) the end of the round.

Concerning (i), for each message to send, the publisher computes a digest and the input of the hash function (i.e., $m||D^p[x]$) has size M plus the size of a digest. In the most common hash functions (e.g., SHA256), the size of the digest is half the size of the block B. However, since the messages are padded until they have a size multiple of B, we can assume that the time to compute a hash function with input less than B is still C_H. Then, the overhead introduced in this phase is $\left(\frac{M}{B}+1\right)\cdot C_H$ in both the baseline approach and MQTT-I.

We want to highlight that in practical terms this overhead is negligible compared to the time needed to complete the transmission of the MQTT message (see Table 3 in Section 10). Indeed, since obtaining integrity requires at least QoS level 1, the transmission time includes at least the time to transmit the message to the broker and the time to receive the MQTT acknowledgment from the broker.

Consider now the overhead introduced by the baseline approach at the end of the round. Three operations have to be considered, i.e., a hash computation (requiring a time C_H), the computation of the signature (requiring a time C_S), and its publication (along with a digest and a timestamp) through MQTT (requiring a time C_P).

Since these operations are performed for each topic in T^p of the publisher, in the baseline approach, we have a computational overhead per round of $|T^p|\cdot(C_H+C_S+C_P)$.

On the other hand, in the MQTT-I protocol, three operations are performed just once. They are the signature computation (with time C_S), the MQTT publication (with time CP), and the Merkle Hash Tree construction. The latter operation requires a linear number of hash computations in the number of topics. It is easy to realize that this number of hash computations is strictly less than $|T^p|$ (corresponding to the number of leaves of the tree). The input of each hash computation is composed of two concatenated digests with a total size of B. Then each hash computation takes C_H. By including the last hash computation (assuming the size of a timestamp less than B), we have that, at most, the time required for the Merkle Hash Tree construction is $(|T^p|+1)\cdot C_H$.

We conclude that the overhead per round of MQTT-I is $C_S+C_P+(|T^p|+1)\cdot C_H$.

By comparing the baseline approach and MQTT-I, we observe that the baseline approach performs better only for $|T^p|=1$. For $|T^p|>1$, it is easy to realize that MQTT-I overcomes the baseline approach. Indeed, in any realistic device and application $C_H << C_S$ and $C_H << C_P$ (see Table 3 in Section 10).

At this point, we analyze the overhead subscriber-side introduced per each publisher p sending data on the topics T^s in which s is interested. We assume $T^s \subseteq T^p$, representing the worst case for the subscriber (otherwise fewer signatures can be verified).

Again this overhead is introduced during two phases: (i) the message reception and (ii) the end of the round of the publisher p.

For each received message, both in the baseline approach and MQTT-I, the subscriber computes a digest requiring a time $\left(\frac{M}{B}+1\right)\cdot C_H$.

However, we make the following consideration. In real applications, the time between two consecutive messages received by the broker is much greater than $\left(\frac{M}{B}+1\right)\cdot C_H$, so this computation is negligible and can be performed between the reception of two consecutive messages.

Concerning the overhead introduced during the end of the round of p, we distinguish the baseline approach from

TABLE 1
Computational overhead.

	Publisher		Subscriber	
/	**Baseline**	**MQTT-I**	**Baseline**	**MQTT-I**
Transmission of messages	$\left(\frac{M}{B}+1\right)\cdot C_H$	$\left(\frac{M}{B}+1\right)\cdot C_H$	$\left(\frac{M}{B}+1\right)\cdot C_H$	$\left(\frac{M}{B}+1\right)\cdot C_H$
End Round	$\lvert T^p\rvert\cdot(C_S+C_P+C_H)$	$C_S+C_P+(\lvert T^p\rvert+1)\cdot C_H$	$\lvert T^s\rvert\cdot(C_V+C_H)$	$C_V+C_H+\lvert T^s\rvert\cdot C_H\cdot log_2\lvert T^p\rvert$

MQTT-I.

In the baseline approach, for each topic in T^s, a signature verification and a hash computation are performed. This requires a cost of $|T^s|\cdot(C_H+C_V)$.

In MQTT-I, the computational overhead is $C_V+C_H+|T^s|\cdot C_H\cdot log_2|T^p|$. The first two terms represent the cost of a signature verification and a hash computation. The third term counts, for each topic in T^s, the cost of the number of hash computations needed to recompute the root of the Merkle Hash Tree built by p. This number is logarithmic in the number of topics $|T^p|$ of the publisher.

By comparing MQTT-I and the baseline approach, the latter may realistically outperform the former only in the case $|T^s|=1$. On the other hand, if $|T^s|>1$, the baseline approach may outperform MQTT-I only if $|T^p|>2^{\frac{C_V}{C_H}+1}$. By taking as an example the values reported in Table 3, $|T^p|$ should be greater than 2^{2658}, which is definitively an unrealistic number of topics. Then the benefits of MQTT-I compared to the baseline approach are confirmed also for the subscribers. The results of this analytical evaluation are summarized in Table 1. We experimentally confirm these results in Section 10, in which we also show that the overhead introduced by MQTT-I compared to the standard MQTT protocol is acceptable.

9 Comparison with existing approaches

As discussed in Sections 2 and 5, the only approaches in the literature that share some similarities with our proposal are the SADS, used in the domain of streaming data outsourcing. In SADS, a data owner generates a stream of data and sends it to a prover (typically, a cloud provider) and a verifier. Subsequently, the verifier submits some queries to the prover to obtain part of these data along with a proof of their integrity. The verifier can then check the integrity of the retrieved data using the result of the computation made while observing the stream.

By comparing this scenario with ours, we can identify the following similarities: *(i)* the roles of data owner, prover, and verifier are played by the publisher, broker, and subscriber, respectively; *(ii)* the goal is the same: the subscriber verifies the integrity of the data originating from the publisher and provided by the broker. However, we highlight two (main) differences: *(iii)* SADS are designed to support query integrity for specific queries; *(iv)* SADS rely on the presence of a trusted direct communication channel between the data owner and the verifier.

Regarding the first difference, it holds significant importance in the case of data outsourcing because, often, the entity hosting data is the cloud, which also serves as query provider. This is not the case of MQTT, where the broker's sole role is to suitably forward data by intermediation (in other words, the broker remains fully agnostic regarding how data is used by the subscriber). Therefore, the only objective we can pursue is to support the integrity of the entire data flow. Based on this consideration, MQTT-I can be compared with SADS by considering the case of the simplest query i.e., "select all", once a topic is fixed. However, the second difference makes SADS not applicable to MQTT. Indeed, it is not realistic to assume the existence of a trusted direct channel in the context of MQTT, due to the decoupling principle of MQTT between publisher and subscriber [51].

TABLE 2
Comparison between SADS and MQTT-I

Technique	Publisher/Broker (Data Structure Construction)	Subscriber (Verification)
[38]	$O(\lvert M^p\rvert\cdot log^2(\lvert M^p\rvert))$	$O(\lvert M^s\rvert\cdot log\lvert M^p\rvert)$
[52]	$O(\lvert M^p\rvert\cdot log(\lvert M^p\rvert))$	$O(\lvert M^s\rvert\cdot log\lvert M^p\rvert)$
[53]	$O(\lvert M^p\rvert\cdot log(\lvert M^p\rvert))$	$O(\lvert M^s\rvert\cdot log\lvert M^p\rvert)$
[54]	$O(\lvert M^p\rvert\cdot log(\lvert M^p\rvert))$	$O(\lvert M^s\rvert\cdot log\lvert M^p\rvert)$
MQTT-I	$O(\lvert M^p\rvert+\lvert T^p\rvert)$	$O(\lvert T^s\rvert\cdot log\lvert T^p\rvert)$

Nevertheless, in the following, we perform an analytical comparison between SADS and MQTT-I to demonstrate that even in terms of asymptotic costs our approach offers clear advantages over SADS. This is just a further argumentation to highlight the intrinsic efficiency of our approach, but, we remark that it remains the non-applicability of SADS to the context of MQTT explained above. We consider the following SADS in our comparison: [52], [53], [38], [54].

Table 2 reports the result of our comparison in terms of asymptotic costs per round for constructing the data structure (performed by publisher and broker) and for verifying the integrity of the data received by the subscriber. We denote by $|M^p|$ and $|M^s|$ the total number of messages sent by the publisher and received by the subscriber, respectively. Observe that they may not coincide since they can publish/subscribe on/to different topics. We denote by $|T^p|$ and $|T^s|$ the number of topics in which the publisher sends messages and the subscriber receives messages, respectively. Since, it is realistic to assume $|T^p|<<|M^p|$ and $|T^s|<<|M^s|$, our approach proves to be more advantageous for both publisher/broker and subscriber.

Intuitively, the advantage of our approach derives from the fact that all the SADS are based on hash trees, as is MQTT-I. However, unlike MQTT-I, SADS do not consider the concept of topic (**Challenge 3**), and the hash trees are built using the messages as leaves instead of the topics as in MQTT-I. This leads to higher costs. This outcome is expected, as they support different features, including specific queries that are not meaningful in the MQTT context.

10 EXPERIMENTS

In this section, we experimentally validate our proposal.

As previously highlighted in Section 5.1, there is no existing solution in the literature applicable to our scenario, so an experimental comparison with other proposals is not possible. However, to give a measure of the advantages introduced by our solution we compare our technique with two "baselines". The first is the baseline approach presented in Section 6, which supports the same security goals as MQTT-I. The second is standard MQTT, which represents the ideal solution from the performance point of view, since no extra computation is required to guarantee integrity. The aim of the last comparison is to understand if the overhead introduced by MQTT-I is acceptable.

10.1 Experimental Environment

Our experimental campaign was carried out by implementing the standard MQTT protocol, the baseline approach, and MQTT-I on physical devices to measure their performance.

Specifically, we used a Raspberry Pi Pico W device to implement publishers and subscribers. It is a very recent (2022) and cheap (6$) device with low computational capabilities (dual-core ARM Cortex-M0+ with frequency 133Mhz), low RAM storage (256 Kb), and low flash memory (2 Mb). This choice is made to show that our protocol is fully supported even in domains where high-performance hardware resources are not available. Concerning the broker, it was deployed on a standard laptop equipped with a 1.8 GHz Intel i7-8850 CPU and 16 GB of RAM.

Publishers, subscribers, and the broker were connected to the network via Wi-Fi.

As software tools, we used the `ArduinoMQTT` library to implement the publisher and subscriber components of MQTT in the three protocols. As for the cryptographic functions (used only in the baseline approach and MQTT-I) we relied on the `Arduino Cryptography Library`. Specifically, we used SHA256 as the hashing algorithm and Ed25519 as the public-key digital signature algorithm.

The broker component was implemented by customizing the Java library `HiveMQT` to include the MQTT-I functionalities, such as the Merkle Hash Tree construction.

In Table 3 we report the time required to perform some basic operations, which are obtained in the considered network environment, using the aforementioned hardware and software. The notation used is the same as Section 8. Observe that C_P is the time required to publish an MQTT message of 100 bytes with QoS level 1 and receive the corresponding ACK from the broker. Since the employed digital signature and hashing algorithms produce signatures of 64 bytes and digests of 32 bytes, C_P is also the time required to publish the digital signature at the end of each round. In the table, we also report the time C_M needed for the publisher to build a Merkle Hash Tree.

TABLE 3
Basic Operation times.

Operation	Time (ms)
Hashing time of a message (C_H)	0.147
Signature computation (C_S)	244.993
Signature Verification (C_V)	390.775
Publish time of a message (Qos 1) (C_P)	50.551
Merkle Hash Tree construction with 100 topics (C_M)	14.684

10.2 Publishers Performance

We start by analyzing the performance of the publishers. We choose as a measure of performance the *throughput*, defined as the number of messages (of 100 bytes) that the publisher can transmit in unity of time. For unity of time we use seconds (denoted by s).

In the standard MQTT protocol, since no overhead is present, the publisher can reach a throughput equal to $\frac{1}{C_P}$, corresponding to about 20 messages per second.

On the other hand, in both the baseline approach and MQTT-I, as discussed in Section 8, an overhead is introduced during the transmission of the messages and at the end of each round. Consequently, a lower throughput is expected. However, the first contribution is definitely negligible, given that the hashing time is minimal in comparison to the publish time. On the other hand, the second contribution may have an impact. Two main parameters may influence the performance of the baseline approach and MQTT-I: the number of topics $|T^p|$ in which the publisher publishes and the round time RT.

We evaluated how the throughput varies as $|T^p|$ and RT vary. The results are depicted in Figures 2 (with $RT = 15s$) and 3 (with $|T^p| = 20$).

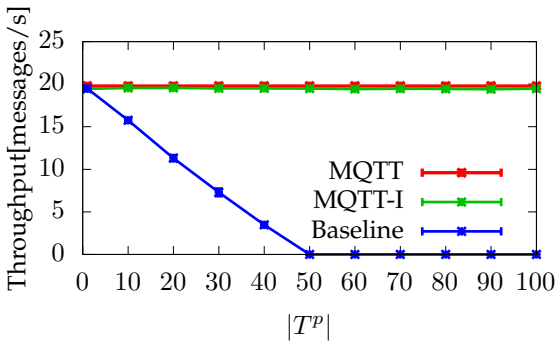


Fig. 2. Throughput of publishers as a function of the number of topics.

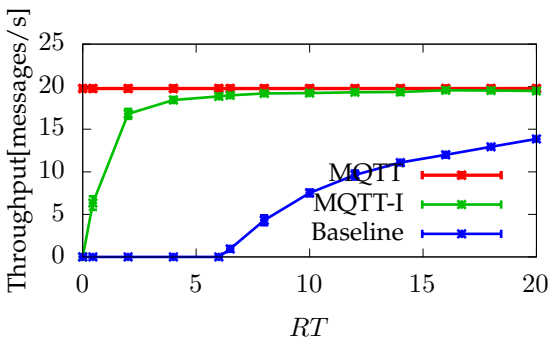


Fig. 3. Throughput of publishers as a function of the round time.

Concerning Figure 2, clearly, in MQTT the throughput does not depend on the number of topics. As for MQTT-I, theoretically, the throughput decreases linearly with the

number of topics since a Merkle Hash Tree (requiring a linear number of hash computations) has to be built at the end of each round. However, this decrease is very slow since the signature time is prevalent (about 250 ms) compared to the Merkle Hash Tree construction (14 ms with 100 topics). Then, no appreciable difference was experimentally observed as $|T^p|$ increases. A slight difference (less than $1,7\%$) is observed between the throughput of MQTT-I and the standard MQTT. On the other hand, concerning the baseline approach, since a linear number of signatures (instead of hash computations) has to be performed at each round, the effect on the throughput is relevant. Indeed, when $|T^p| = 50$, the publisher computes and publishes 50 signatures, which takes more time than the round time (15 seconds), so that no more messages can be sent and the throughput is 0.

At this point, we investigate the effect of the round time on the throughput. Obviously, again the throughput of MQTT does not depend on RT. In both the baseline approach and MQTT-I, as expected, the throughput increases as the round time increases. Indeed, the time to perform the operations at each round is fixed (by fixing the number of topics), then an improvement is expected when these operations are performed less frequently. However, it is clear that RT is a security parameter that defines when the integrity can be verified (see Section 11), then it should not be too high. In Figure 3, we observe that MQTT-I reaches the throughput of the standard MQTT very quickly (it reaches 6 messages per second with RT =0.45s and 16 messages per second with RT =2s). On the other hand, the increase in throughput for the baseline approach is much slower (it requires RT =6.5s to obtain a throughput of 1 message per second).

In conclusion, from the perspective of publishers, beyond its relevant benefits in terms of security, the overhead of MQTT-I is negligible compared to the standard MQTT. On the other hand, the baseline approach does not present advantages (compared to MQTT-I) and may be adopted only in restricted cases (a single-topic scenario).

10.3 Subscribers Performance

Now, we study the performance of the subscribers when implementing the three protocols. Again, we consider the maximum throughput each subscriber may support, which is defined as the maximum number of messages that can be received (and processed) within a unit of time. However, unlike publishers, the throughput of the subscribers does not depend only on the device hardware and network limitations, but also on the number of publishers and their sending rate.

Now, observe that, in general, the throughput of a subscriber cannot exceed the sum of the sending rates of the publishers. On the other hand, if the throughput were significantly less than this sum, it would imply that the messages are being enqueued at some point (in the broker queue, or in the subscriber queue). However, in the long run, this situation cannot be sustained, as the size of the queues is limited. Then, the maximum throughput supported by the subscriber is very close to the sum of the sending rates of the publishers.

By trial and error, we found that a subscriber (on a Raspberry Pi Pico W device) implementing the standard MQTT protocol may support 12 simultaneous publishers (also implemented on the same typology of devices) each transmitting at the rate of 5 messages per second (for such a subscriber). Then, in MQTT, the maximum supported throughput for subscribers is about 60 messages per second.

In the baseline approach and MQTT-I, the signature verification at the end of the publishers' rounds leads to lower throughput. Indeed, during the round, the messages are processed at the same throughput of MQTT (the hash computation does not affect the throughput). However, when a signature is being verified, other messages coming from the same or other publishers are enqueued at the subscriber but cannot be immediately processed. They will be processed at the end of the signature verification. If this processing occurs before the end of the next round, the corresponding throughput can be measured; otherwise, we need to reduce the number of publishers, thus resulting in a lower throughput.

Again, the performance of MQTT-I depends on the number of topics of the subscribers and the round time of the publishers. As worst-case conditions for MQTT-I and the baseline approach, we consider 12 (or fewer) synchronized publishers in which $T^p = T^s$, i.e., they send messages simultaneously (as much as possible) on all the topics of the subscriber.

The results of our analysis are reported in Figures 4 (with $RT = 15$ seconds) and 5 (with $|T^s| = 20$).

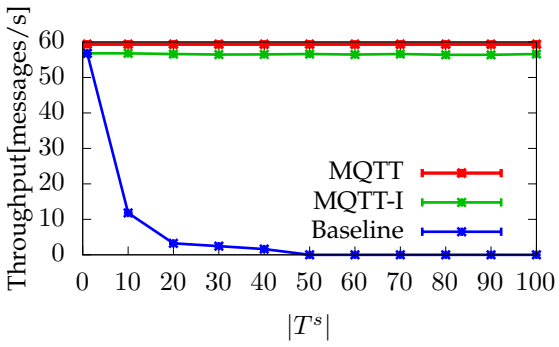


Fig. 4. Throughput of subscribers as a function of the number of topics.

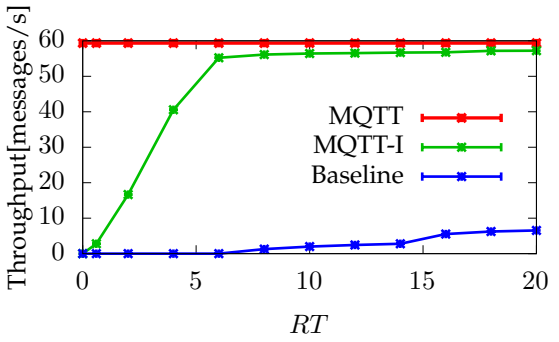


Fig. 5. Throughput of subscriber as a function of the round time.

These results are aligned with those in Section 10.2. MQTT-I requires a logarithm number (in $|T^s|$) of hash

computations at the end of each round, but this cannot be appreciated compared to the single signature verification. Then the throughput remains nearly constant in $|T^s|$ and very close to that of MQTT (with a percentage difference of approximately 5%). On the other hand, the performance of the baseline approach degrades rapidly as $|T^s|$ increases.

Concerning the plot of Figure 5, the baseline approach exhibits slow growth with an increase in round time. On the other hand, MQTT-I approaches MQTT when $RT > 6s$.

10.4 Power Consumption

We analyzed the overhead in terms of power consumption introduced by our proposal compared to the standard MQTT protocol. In particular, we measured the energy required to transmit $n = 2000$ packets in the three approaches. To do this, we leveraged a high-precision USB multimeter, with a sampling rate equal to 1000sps. From some preliminary measurements, we drew some considerations.

First, to send data and receive acknowledgements, the Wi-Fi module has to be enabled. This absorbs a current of about 60 mA excluding the instant in which data are sent and received. Concerning the cryptographic operations (hashing and signature computations), we observe a very slight increment of the required current (about 1 mA). Finally, we observe some peaks of currents (80-140 mA) when the packets are sent and their acknowledgements are received. Since the number of transmitted packets is the same for the three protocols, the impact of these peaks in terms of energy will be the same. Furthermore, the overhead to perform the hash computations in terms of current is negligible. However, since the throughput achieved in MQTT-I and the baseline approach is smaller than the throughput of MQTT, the devices need more time to transmit the same number of packets. This leads to higher power consumption.

Given that the core proposal of this work is MQTT-I (and not the baseline approach), we focus on a scenario in which some differences between MQTT-I and MQTT can be appreciated. Specifically, we considered the second scenario outlined in Section 10.2 where we fixed 20 topics and analyzed the energy consumption to transmit 2000 packets as the round time varies.

As previously discussed, the number of topics, in practice, does not influence the performance of MQTT-I. The results of these measurements are reported in Figure 6.

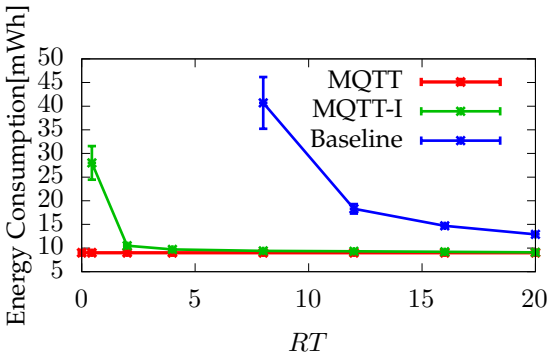


Fig. 6. Energy consumption for the transmission of 2000 packets.

As expected, this plot is mirrored with respect to the plot of Figure 3. Specifically, MQTT always requires less energy consumption. However, MQTT-I rapidly approaches MQTT, so MQTT-I does not require high values of RT. In contrast, the baseline approach performs well only with high round times. We conclude that, even from an energy consumption perspective, MQTT-I is applicable.

10.5 Discussion

In this section, we summarize the results of the previous sections and discuss the trade-offs in terms of security, throughput, and energy consumption of MQTT, MQTT-I, and the baseline approach. Furthermore, we provide some considerations related to the configuration of MQTT-I parameters. Finally, we discuss some limitations of our approach.

We briefly compare MQTT-I and the baseline approach. Essentially, no trade-off exists. MQTT-I offers the same security guarantees as the baseline approach (for the same round time) but outperforms it in terms of throughput and energy consumption for any round time and any non-degenerate case. Indeed, the baseline approach slightly outperforms MQTT-I only in the case of a single topic shared by the publisher and subscriber, which represents a degenerate case of use of MQTT. Concerning the comparison between MQTT and MQTT-I, we observe that a trade-off exists in terms of security and performance. Although MQTT offers better throughput and energy consumption, it provides no guarantee of data integrity.

In our experimental campaign, we investigated the main parameters potentially affecting the performance of MQTT-I, i.e., the number of topics and the round time. It arises that the number of topics does not appreciably affect the performance of MQTT-I, making it suitable for use in a scenario with high number of topics [55], [56].

Then, the only parameter to set properly is the round time RT. The performance of MQTT-I improves with higher values of RT. However, the round time also defines a sort of *vulnerability window* during which the integrity of received messages is not yet verified. In other words, the subscriber has to wait the completion of the round to check the integrity of the data received in this round. Then, from a security standpoint, RT should be maintained as short as possible, as it represents a delay imposed on the subscriber that intends to process data only after their verification.

The experimental results show that acceptable performance (comparable to standard MQTT) can be achieved for both publishers and subscribers in terms of throughput and energy consumption with $RT > 6s$. The presence of a vulnerability window is a limitation of MQTT-I, which may make it unsuitable for those hard real-time applications in which data integrity should be guaranteed within milliseconds. However, these applications [57] rely on hardware with no limited capability as that typically considered in MQTT. However, this aspect would merit further investigation (which we plan as future work). Indeed, in the current configuration of our method, the reduction of the round time is hindered by the cost of digital signatures, whose frequency increases inversely with respect to the round time (actually, we require one digital signature per round time). Therefore, in a high-speed application, we could think of some highly efficient signature schemes (such as [58]) to

make this source of inefficiency acceptable. Obviously, this aspect does not impact the outcomes of this paper, in which we aim to demonstrate the advantages of our methods and its general applicability to the MQTT context. Therefore, we used standard Elliptic curves for digital signatures whose advantage is the availability of secure cryptographic libraries and hardware implementations.

11 Security Analysis

Through this section, we provide a security analysis of the proposed approach according to the threat model presented in Section 4. Since the baseline approach does not present any advantage from the performance point of view, we perform our analysis only on MQTT-I, which is the actual proposal of this work. On the other hand, it is easy to see that they are equivalent from the security point of view.

Security Properties. In Section 4, we formalized the notion of data flow integrity (on the topic t) as $DFI_t \equiv CM_t \wedge CR_t \wedge SL_t$, where:

- $CM_t \equiv (M_t^p \subseteq M_t^s)$
- $CR_t \equiv (M_t^s \subseteq M_t^p)$
- $SL_t \equiv WL_t \wedge (\forall\ m \in (M_t^p \cap M_t^s)$ it holds $f_t^s(m) < f_t^p(m) + \Delta)$, where $WL_t \equiv \forall\ m, m' \in (M_t^p \cap M_t^s)$, it holds $(f_t^p(m) < f_t^p(m')) \iff (f_t^s(m) < f_t^s(m'))$

Adversarial Model. We consider the adversary $\mathcal{A}$ presented in Section 4 which performs the following compromises:

- $C1 \equiv (M_t^p \cap \overline{M_t^s} \neq \emptyset) \wedge (\overline{M_t^p} \cap M_t^s \neq \emptyset)$.
- $C2 \equiv (M_t^p \subset M_t^s)$.
- $C3 \equiv (M_t^s \subset M_t^p)$.
- $C4 \equiv \exists\ m, m' \in (M_t^p \cap M_t^s)$, such that it holds $f_t^p(m) < f_t^p(m')$ and $f_t^s(m) > f_t^s(m')$.
- $C5 \equiv \exists\ m \in (M_t^p \cap M_t^s)$ such that $f_t^s(m) > f_t^p(m) + \Delta$.

The first part of this analysis is devoted to proving that these compromises are all and only the possible actions leading to a breaking of data flow integrity.

In the following lemma, we show for each of the four identified security properties which are the compromises that can break them.

Lemma 11.1. *The following claims hold.*
Claim 1: $\neg CM_t \iff C1 \vee C3$
Claim 2: $\neg CR_t \iff C1 \vee C2$
Claim 3: $\neg WL_t \iff C4$
Claim 4: $\neg SL_t \iff C4 \vee C5$

Proof. We prove the four claims.
Claim 1: By the double implication, proving $\neg CM_t \iff C1 \vee C3$ is equivalent to proving that $\neg(C1 \vee C3) \iff CM_t$. By definition $C1 \equiv (M_t^p \cap \overline{M_t^s} \neq \emptyset) \wedge (\overline{M_t^p} \cap M_t^s \neq \emptyset)$ and $C3 \equiv M_t^s \subset M_t^p$. Since $M_t^s \subset M_t^p \iff (M_t^p \cap \overline{M_t^s} \neq \emptyset) \wedge (\overline{M_t^p} \cap M_t^s = \emptyset)$, we have that $C3 \iff (M_t^p \cap \overline{M_t^s} \neq \emptyset) \wedge (\overline{M_t^p} \cap M_t^s = \emptyset)$. We say $A \equiv (M_t^p \cap \overline{M_t^s} = \emptyset)$ and $B \equiv (\overline{M_t^p} \cap M_t^s = \emptyset)$. Then, $C1 \iff \neg A \wedge \neg B$ and $C3 \iff \neg A \wedge B$. By applying De Morgan's laws $\neg(C1 \vee C3) \iff \neg(C1) \wedge \neg(C3)$ and $\neg(C1) \iff A \vee B$ and $\neg(C3) \iff A \vee \neg B$.

Then, $\neg(C1 \vee C3) \iff (A \vee B) \wedge (A \vee \neg(B)) \iff (A \wedge A) \vee (A \wedge \neg(B)) \vee (B \wedge A) \vee (B \wedge \neg(B)) \iff A \vee (A \wedge \neg(B)) \vee (B \wedge A) \iff A \iff (M_t^p \cap \overline{M_t^s} = \emptyset) \iff M_t^p \subseteq M_t^s$. This ends the proof.

Claim 2: By replacing M_t^s with M_t^p, it follows immediately by **Claim 1**. This ends the proof.
Claim 3: It immediately follows from the definitions of WL_t and $C4$. This ends the proof.
Claim 4: We say $A \equiv (\forall\ m \in (M_t^p \cap M_t^s)$ it holds $f_t^s(m) < f_t^p(m) + \Delta)$. By the double implication, proving $\neg SL_t \iff C4 \vee C5$ is equivalent to proving that $\neg(C4 \vee C5) \iff SL_t$. By De Morgan's laws, $\neg(C4 \vee C5) \iff \neg(C4) \wedge \neg(C5)$. By **Claim 3**, $\neg(C4) \iff WL_t$. By definition of $C5$, it holds that $\neg(C5) \iff A$. Then, $\neg(C4 \vee C5) \iff \neg(C4) \wedge \neg(C5) \iff WL_t \wedge A \iff SL_t$. This ends the proof. □

At this point, we can prove the data flow integrity can be broken if and only if at least one of the compromises occurs.

Theorem 11.1. $\neg DFI_t \iff C1 \vee C2 \vee C3 \vee C4 \vee C5$

Proof. By definition of DFI_t, $\neg DFI_t \iff \neg(CM_t \wedge CR_t \wedge SL_t)$. By De Morgan's laws, $\neg DFI_t \iff \neg CM_t \vee \neg CR_t \vee \neg SL_t$. By Lemma 11.1, it follows $\neg DFI_t \iff C1 \vee C2 \vee C3 \vee C4 \vee C5$. This ends the proof. □

At this point, the rest of the analysis is devoted to showing that MQTT-I ensures data flow integrity.

We start with some assumptions.

Assumptions. In this analysis we will assume the following.
A1: H is a secure cryptographic hash function (i.e., it offers preimage resistance, second preimage resistance, and collision resistance).
A2: the private key (i.e., PR^p) is in the exclusive possession of p.
A3: $\Delta \geq RT + \delta$.

The first two assumptions are basic assumptions adopted for any integrity protocol. The third assumption, as discussed in Lemma 11.6, refers to the fact that a temporal shifting of a message can be detected when it is greater than a round time plus a network delay.

In MQTT-I, all the security properties are verified by the subscriber when invoking the `EndRound` function of Algorithm 2. We call *current round*, the round in which this function is invoked by the subscriber.

We recall that the `EndRound` takes as input $m^* = H_r||\tau^p||\sigma^p$ and $t^* = ID^p/signature$.

The following Lemmas ensures that if `EndRound` returns $SUCCESS$, then the compromises $C1$, $C2$, $C3$, $C4$, and $C5$ do not occur.

Lemma 11.2. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C_1$

Proof. By contradiction, suppose $C1$ occurs. This implies that there exists a message received by s but not sent by p. Say m' such a message. m' is received either in the current round or in the previous rounds.

The function `EndRound`(m^*, t^*) returns $SUCCESS$ if and only if the signature σ^p of $H_s = H(R^s[ID^p]||cr||\tau^p)$ is verified.

By Assumption **A2**, σ^p cannot be forged by $\mathcal{A}$, then `EndRound`(m^*, t^*) returns $SUCCESS$ if and only if H_s is equal to $H(pr^*||cr^*||\widehat{\tau^p})$ where pr^*,cr^*, and $\widehat{\tau^p}$ are values computed by p.

By Assumption **A1**, this happens if and only if $pr^* = R^s[ID^p]$, $cr^* = cr$, and $\widehat{\tau^p} = \tau^p$.

Considering $cr^* = cr$, it means that the root of the Merkle Hash Tree obtained in the current round by p (i.e., cr^*) is the same as the current root obtained by s (i.e., cr). We recall that cr^* is obtained by performing a given number of hash computations including all the messages sent by p in the current round. Similarly, cr is obtained by performing the same number of hash computations including all the messages received by s in the current round. Then, by Assumption **A2**, if $cr^* = cr$, then m' cannot be included in the current round.

Then, consider m' belonging to a previous round, say R such a round. By applying the above reasoning, the root computed by p in the round R will be different from the root computed by s in the round R. We recall that pr^* is obtained by applying several hash computations including all the roots obtained in the previous rounds (among which R) by p. Similarly, $R^s[ID^p]$ is obtained by applying several hash computations including all the roots obtained in the previous rounds (among which R) by s. Then, if in the round R, the roots computed by p and s are different, then $R^s[ID^p] \neq pr^*$. Therefore, $R^s[ID^p] = pr^*$ and $cr^* = cr$ only if m' belongs to neither the current round nor the previous rounds. This contradicts the hypothesis and concludes the proof. □

Lemma 11.3. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C_2$

Proof. This proof is the same as the proof of Lemma 11.2. Indeed, if $C2$ occurs, then there exists a message m' received by s but not sent by p. This concludes the proof. □

Observe that $C1$ and $C2$ are indistinguishable from the point of view of the subscriber since they result in the same effect (i.e., a message not sent by p is received by s).

Lemma 11.4. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C_3$

Proof. By contradiction, suppose $C3$ occurs. This implies that there exists a message m' sent by p but not received by s. m' is sent either in the current round or in the previous rounds.

The reasoning is similar to the proof of Lemma 11.2. Indeed, if m' is sent during the current round, by Assumption **A1**, the root computed in the current round by p (including the hash computation of m') is different from the root computed by s (in which the hash computation of m' is not performed). On the other hand, if m' was sent in a previous round, by Assumption **A1**, the roots computed by p and s in such a round are different. Then, the values stored in pr^* and $R^s[ID^p]$ are different.

By Assumption **A2**, this proves that if m' exists, the signature cannot be verified and the `EndRound` function does not return $SUCCESS$. This concludes the proof. □

Lemma 11.5. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C_4$

Proof. By contradiction, suppose $C4$ occurs. This implies that there exist two messages m' and m'' such that m' is sent by p before m'' and m' is received by s after m''.

Two cases may occur: (i) m and m' belong to different rounds and (ii) m and m' belong to the same round, say R.

Considering (i), it means that there exists a round R^* in which a message not sent by p is received by s. Then, by applying the same reasoning of Lemma 11.2, by Assumption **A1**, the roots computed by p and s in the round R^* are different. Then, by Assumption **A2**, the signature cannot be verified (in the current round) and the `EndRound` function does not return $SUCCESS$.

Consider now case(ii). We recall that the hash computations including m and m' are performed by p in the same order in which the messages are sent and by s in the same order in which the messages are received.

Since this order is different, by Assumption **A1**, the roots computed by s and p in the round R are different. As before (with R in place of R^*), by Assumption **A2**, the signature cannot be verified (in the current round) and the `EndRound` function does not return $SUCCESS$.

This contradicts the hypothesis and concludes the proof. □

Lemma 11.6. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C_5$

Proof. By contradiction, suppose $C5$ occurs. This implies that there exists a message m' sent by p at time $\widehat{\tau^p}$ and received by s at time $\widehat{\tau^s} > \Delta + \widehat{\tau^p}$.

Suppose m' is sent by p at the round R. We denote by τ^p, the timestamp in which p sends the signature at the end of the round R.

Two cases may occur: (i) the adversary $\mathcal{A}$ does not delay the transmission of the signature, and (ii) the adversary delays the transmission of the signature.

Consider case (i). By Assumption **A3**, since $\Delta > RT + \delta$, the message m' is received by s after receiving the signature message of the round R.

This means that m' is not included in the round R and then, by Lemma 11.4, the signature cannot be verified and the `EndRound` function does not return $SUCCESS$.

In case (ii), the signature message is delayed so that the message m' is included in the round R from the point of view of the subscriber (otherwise we fall in case (i)).

This means that the signature message is received at a time $\tau^s > \widehat{\tau^s} > \Delta + \widehat{\tau^p}$.

The signature message also includes the timestamp τ^p generated by p. Since τ^p is included in the hash computation performed in the round R by p, by Assumptions **A1** and **A2**, the signature can be verified only if the value τ^p received by is the original one sent by p.

Since $\tau^p - \widehat{\tau^p} < RT$ (the signature is sent within the round in which m' is sent), we have that $\tau^s > \Delta + \widehat{\tau^p} > \Delta + \tau^p - RT$.

By Assumption **A3**, $\tau^s > RT + \delta + \tau^p - RT = \delta + \tau^p$. In this case, the `EndRound` function returns $ERROR$.

This concludes the proof. □

We conclude our security analysis with a theorem stating that MQTT-I guarantees data flow integrity.

Theorem 11.2. $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow DFI_t$

Proof. By Lemmas 11.2, 11.3, 11.4, 11.5, and 11.6, we have that $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow \neg C1 \wedge \neg C2 \wedge \neg C3 \wedge \neg C4 \wedge \neg C5$.

By De Morgan's laws, $\neg C1 \wedge \neg C2 \wedge \neg C3 \wedge \neg C4 \wedge \neg C5 \iff \neg(C1 \vee C2 \vee C3 \vee C4 \vee C5)$. By Theorem 11.1 $\neg(C1 \vee C2 \vee C3 \vee C4 \vee C5) \iff DFI_t$. Then, $(\texttt{EndRound}(m^*, t^*) = SUCCESS) \Rightarrow DFI_t$. This concludes the proof. □

12 Conclusions

In this paper, we propose MQTT-I, a solution to provide MQTT with end-to-end data flow integrity. The proposed solution consists of the non-trivial application of the notion of Merkle Hash Tree (used in the field of data outsourcing) to the context of MQTT. This was accomplished by addressing two specific problems: the dynamic nature of the scenario and the presence of multiple topics. Additionally, we rigorously considered the critical aspects of computational overhead and energy consumption to guide us toward the chosen solution. The security of MQTT-I is proven through a formal framework in which we formalize the concept of data flow integrity. In addition to experiments, which aim to carefully analyze the protocol's performance in terms of throughput and power consumption, we also provide a computational complexity analysis of the overhead introduced by MQTT-I. The outcomes validate the proposal, showing that, under a wide range of conditions, MQTT-I includes the desired security features with a very small impact on performance compared to standard MQTT.

As a future work, we plan to better investigate the limitations of our approach, discussed in Section 10.5. As explained there, we can argue that MQTT-I (in its current configuration) seems not suitable for high-rate hard real-time applications. It could be interesting to better study this limitation by defining more exactly the boundary of application of our technique in real-life settings and evaluate the application of highly efficient digital signature schemes to our technique.

Acknowledgments

This work was partially supported by the project STRIDE included in the Spoke 5 (Cryptography and Distributed Systems Security) of the Research and Innovation Program PE00000014, "SEcurity and RIghts in the CyberSpace (SERICS)", under the National Recovery and Resilience Plan, funded by the European Union, NextGenerationEU.

References

[1] OASIS, "MQTT version 3.1.1," vol. 1, 2014. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3

[2] G. C. Hillar, *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.

[3] R. Gandhewar, A. Gaurav, K. Kokate, H. Khetan, and H. Kamat, "Cloud based framework for IIoT application with asset management," in *2019 3rd International conf. on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 2019, pp. 920–925.

[4] A. Lachtar, T. Val, and A. Kachouri, "Elderly monitoring system in a smart city environment using LoRa and MQTT," *IET Wireless Sensor Systems*, vol. 10, no. 2, pp. 70–77, 2020.

[5] R. K. Kodali and S. Soratkal, "MQTT based home automation system using ESP8266," in *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*. IEEE, 2016, pp. 1–5.

[6] P. Colombo, E. Ferrari, and E. D. Tümer, "Efficient ABAC based information sharing within MQTT environments under emergencies," *Computers & Security*, vol. 120, p. 102829, 2022.

[7] D.-K. Choi, J.-H. Jung, J.-I. Kim, M. Gohar, and S.-J. Koh, "IoT-based resource control for in-vehicle infotainment services: Design and experimentation," *Sensors*, vol. 19, no. 3, p. 620, 2019.

[8] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S—a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conf. on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[10] M. Dahlmanns, J. Pennekamp, I. B. Fink, B. Schoolmann, K. Wehrle, and M. Henze, "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*, 2021, pp. 78–87.

[11] H. AP and K. Kanagasabai, "Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–15, 2019.

[12] A. A. Cardenas, T. Roosta, and S. Sastry, "Rethinking security properties, threat models, and the design space in sensor networks: A case study in scada systems," *Ad Hoc Networks*, vol. 7, no. 8, pp. 1434–1447, 2009.

[13] F. Zafar, A. Khan, S. U. R. Malik, M. Ahmed, A. Anjum, M. I. Khan, N. Javed, M. Alam, and F. Jamil, "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends," *Computers & Security*, vol. 65, pp. 29–49, 2017.

[14] R. C. Merkle, "A certified digital signature," in *Advances in cryptology—CRYPTO'89 proceedings*. Springer, 2001, pp. 218–238.

[15] B. Zhang, B. Dong, and W. H. Wang, "Integrity authentication for sql query evaluation on outsourced databases: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 4, pp. 1601–1618, 2019.

[16] O. Sadio, I. Ngom, and C. Lishou, "Lightweight security scheme for MQTT/MQTT-SN protocol," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 119–123.

[17] M. Hamad, A. Finkenzeller, H. Liu, J. Lauinger, V. Prevelakis, and S. Steinhorst, "SEEMQTT: secure end-to-end MQTT-based communication for mobile IoT systems using secret sharing and trust delegation," *IEEE Internet of Things Journal*, 2022.

[18] C.-S. Park and H.-M. Nam, "Security architecture and protocols for secure MQTT-SN," *IEEE Access*, vol. 8, pp. 226 422–226 436, 2020.

[19] H. Lee, J. Lim *et al.*, "MQTLS: toward secure MQTT communication with an untrusted broker," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2019, pp. 53–58.

[20] S. Pallickara, M. Pierce, H. Gadgil, G. Fox, Y. Yan, and Y. Huang, "A framework for secure end-to-end delivery of messages in publish/subscribe systems," in *2006 7th IEEE/ACM International Conference on Grid Computing*. IEEE, 2006, pp. 215–222.

[21] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar, "Secure MQTT for internet of things (IoT)," in *2015 fifth international conference on communication systems and network technologies*. IEEE, 2015, pp. 746–751.

[22] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, "AUPS: An open source authenticated publish/subscribe system for the internet of things," *Information Systems*, vol. 62, pp. 29–41, 2016.

[23] S. Berlato, U. Morelli, R. Carbone, and S. Ranise, "End-to-end protection of IoT communications through cryptographic enforcement of access control policies," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2022, pp. 236–255.

[24] C. Ge, W. Susilo, J. Baek, Z. Liu, J. Xia, and L. Fang, "Revocable attribute-based encryption with data integrity in clouds," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 2864–2872, 2022.

[25] J. Wei, X. Chen, X. Huang, X. Hu, and W. Susilo, "RS-HABE: revocable-storage and hierarchical attribute-based access scheme for secure sharing of e-health records in public cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2301–2315, 2019.

[26] Q. Wang, D. Wang, C. Cheng, and D. He, "Quantum2FA: efficient quantum-resistant two-factor authentication scheme for mobile devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 193–208, 2023.

[27] D. Wang and P. Wang, "Two birds with one stone: Two-factor authentication with security beyond conventional bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 708–722, 2018.

[28] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-Many End-to-End encryption and key delegation for IoT," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1519–1536.

[29] C. Segarra, R. Delgado-Gonzalo, and V. Schiavoni, "MQT-TZ: secure MQTT broker for biomedical signal processing on the edge," in *Health Technology and Informatics*. Proceedings of 2020 Medical Informatics Europe on Digital Personalized Health and Medicine (MIE '20), 2020, pp. 332—-336.

[30] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[31] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 598–609.

[32] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 1–34, 2011.

[33] A. Juels and B. S. Kaliski Jr, "PORs: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 584–597.

[34] H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of cryptology*, vol. 26, no. 3, pp. 442–483, 2013.

[35] P. Samarati, "Data security and privacy in the cloud," in *International Conference on Information Security Practice and Experience*. Springer, 2014, pp. 28–41.

[36] R. Tamassia, "Authenticated data structures," in *Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11*. Springer, 2003, pp. 2–5.

[37] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, pp. 21–41, 2004.

[38] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 353–370.

[39] F. Buccafurri, V. De Angelis, and G. Lax, "An integrity-preserving technique for range queries over data streams in two-tier sensor networks," *Computer Networks*, vol. 217, p. 109316, 2022.

[40] A. Zabasta, N. Kunicina, K. Kondratjevs, A. Patlins, L. Ribickis, and J. Delsing, "Mqtt service broker for enabling the interoperability of smart city systems," in *2018 Energy and Sustainability for Small Developing Economies (ES2DE)*. IEEE, 2018, pp. 1–6.

[41] L. G. A. Rodriguez and D. M. Batista, "Resource-intensive fuzzing for MQTT brokers: State of the art, performance evaluation, and open issues," *IEEE Networking Letters*, 2023.

[42] S. Rani and R. Kumar, "Bibliometric review of actuators: Key automation technology in a smart city framework," *Materials Today: Proceedings*, vol. 60, pp. 1800–1807, 2022.

[43] Y. Hashmy, Z. U. Khan, F. Ilyas, R. Hafiz, U. Younis, and T. Tauqeer, "Modular air quality calibration and forecasting method for low-cost sensor nodes," *IEEE Sensors Journal*, vol. 23, no. 4, pp. 4193–4203, 2023.

[44] N. Tantitharanukul, K. Osathanunkul, K. Hantrakul, P. Pramokchon, and P. Khoenkaw, "MQTT-topic naming criteria of open data for smart cities," in *2016 International Computer Science and Engineering Conference (ICSEC)*. IEEE, 2016, pp. 1–6.

[45] V. M. Larios, L. Gomez, O. B. Mora, R. Maciel, and N. Villanueva-Rosales, "Living labs for smart cities: A use case in Guadalajara city to foster innovation and develop citizen-centered solutions," in *2016 IEEE International Smart Cities Conference*, 2016, pp. 1–6.

[46] L. P. Maguluri, T. Srinivasarao, M. Syamala, R. Ragupathy, and N. Nalini, "Efficient smart emergency response system for fire hazards using IoT," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 1, 2018.

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

[48] F. Buccafurri, V. de Angelis, and S. Lazzaro, "MQTT-A: a broker-bridging p2p architecture to achieve anonymity in MQTT," *IEEE Internet of Things Journal*, vol. 10, no. 17, pp. 15 443–15 463, 2023.

[49] H. C. Hwang, J. Park, and J. G. Shon, "Design and implementation of a reliable message transmission system based on MQTT protocol in IoT," *Wireless Personal Communications*, vol. 91, no. 4, pp. 1765–1777, 2016.

[50] I. Mironov, "Hash functions: From merkle-damgård to shoup," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 166–181.

[51] B. Dickens III, H. S. Gunawi, A. J. Feldman, and H. Hoffmann, "Strongbox: confidentiality, integrity, and performance using stream ciphers for full drive encryption," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 708–721.

[52] J. Xu, L. Wei, Y. Zhang, A. Wang, F. Zhou, and C. zhi Gao, "Dynamic fully homomorphic encryption-based merkle tree for lightweight streaming authenticated data structures," *Journal of Network and Computer Applications*, vol. 107, pp. 113–124, 2018.

[53] C.-M. Yu, "Poster: Lightweight streaming authenticated data structures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1693–1695.

[54] Y. Sun, Q. Liu, X. Chen, and X. Du, "An adaptive authenticated data structure with privacy-preserving for big data stream in cloud," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3295–3310, 2020.

[55] M. R. Palattella, R. Soua, A. Stemper, and T. Engel, "Aggregation of MQTT topics over integrated satellite-terrestrial networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, p. 96–97, jan 2019.

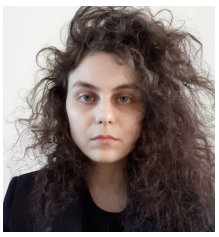
[56] Y. Hashmy, Z. U. Khan, F. Ilyas, R. Hafiz, U. Younis, and T. Tauqeer, "Modular air quality calibration and forecasting method for low-cost sensor nodes," *IEEE Sensors Journal*, vol. 23, no. 4, pp. 4193–4203, 2023.

[57] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols," in *2019 IEEE International Conference on Industrial Technology (ICIT)*, 2019, pp. 955–962.

[58] M. O. Ozmen, A. A. Yavuz, and R. Behnia, "Energy-aware digital signatures for embedded medical devices," in *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2019, pp. 55–63.

Francesco Buccafurri. Full professor of computer science at the University Mediterranea of Reggio Calabria, Italy. In 1995 he took the PhD degree in computer science at the University of Calabria (Italy). In 1996, he was visiting researcher at the database and knowledge representation group of Vienna University of Technology. His research interests include information security and privacy, social network analysis, deductive-databases, knowledge-representation and non-monotonic reasoning, model checking, data compression, data streams, agents, P2P systems. He has published several papers in top-level international journals and conference proceedings. He serves as a referee for international journals and he is a member of a number of conference PCs. He is Associate Editor of Information Sciences (Elsevier), he is also included in the editorial board of a number of international journals and played the role of PC chair in a number of international conferences.

Vincenzo De Angelis. Assistant professor at the University of Calabria, Italy. In 2023 he took the PhD degree at the University of Reggio Calabria. His research interests include information IoT security, blockchain, cloud, and applied cryptography. He is author of more than 20 papers published in international journals and conference proceedings. He is PC member of a number of conferences and Guest Editor of a special issue in an international Journal.



Sara Lazzaro. PhD student in information engineering at the University Mediterranea of Reggio Calabria, Italy. She received the Master's degree in telecommunication engineering in 2021. Her research interests include information security and privacy. She is author of a number of papers published in international journals and international conference proceedings. She is PC member of a number of conferences and Guest Editor of a special issue in an international Journal.