Blockchain-based Privacy Enforcement in the IoT domain

Federico Daidone, Barbara Carminati *Senior member*, and Elena Ferrari *IEEE Fellow*

Abstract—The Internet of Things (IoT) pervades our lives every day and has given end users the opportunity of accessing personalized and advanced services based on the analysis of the sensed data. However, IoT services are also characterized by new challenges related to security and privacy because end users often share sensitive data with different consumers without precise knowledge of how they will be managed and used. To cope with these issues, we propose a blockchain-based privacy enforcement framework where users can define how their data can be used and check if their will is respected without relying on a centralized manager. The preliminary tests we performed, simulating different scenarios, show the feasibility of our approach.

Index Terms—Data privacy, IoT, privacy preference enforcement, blockchain, MIoT.

1 Introduction

The rise and widespread of IoT devices have made it possible to deliver personalized and advanced services to end users. Just think about systems like Amazon Alexa or Google Home, collecting a large amount of data and sending them to the cloud for processing, or Samsung SmartThings, where everyone can insert an application to interact with IoT devices. However, all these IoT systems, which often do not have a user interface, raise serious concerns about user privacy management [1]. Another very privacy sensitive IoT domain is the Medical Internet of Things (MIoT), which makes it possible to improve the quality of patients' care significantly, thanks to remote patient monitoring (RPM) [2]. MIoT has recently raised its importance due to its crucial role in managing the COVID-19 pandemic emergency. Indeed, what has been learned about the pandemic till now is that patients should be as much as possible treated at their home since hospitals are often close to their full capacity. However, safely keeping a COVID-19 patient at home requires a constant monitoring of vital signs (e.g., oxygenation level) and a constant adjustment of the treatment, and this can be done through MIoT applications (e.g., [3]). However, delivering such personalized healthcare services remotely implies addressing many privacy concerns in that highly sensitive data should be monitored and processed.

Typically, data consumers operate on the collected data following privacy regulations, thus they accordingly have to provide their privacy policies. Data owners can use these privacy policies to take a privacy-aware decision on the services' usage. However, today, the most common privacy strategy is the opt-out one, according to which data owners only option is either to accept the privacy policy as is, or not to use the service. This results in a lack of flexibility, as the data owner is forced to accept the consumers' policies to access the service, especially in critical services, like the ones related to MIoT. To give more control to the data owner, he/she should be able to state his/her preferences on how his/her data have to be managed on consumers side. For example, a data owner should state for which purpose his/her data can be collected (such as geolocalization and not analysis of habits) and/or for how long these data can remain in consumers' servers.

Therefore, it is essential to start developing devices following the privacy by design/default model and, in particular, being able to enforce user privacy preferences on the delivered data [4]. This could be practically infeasible in real-world scenarios because of the many devices already on the market. Another critical concern to inject privacy-aware capabilities in IoT devices is that they are often resource-constrained devices, managing only elementary sensing tasks (e.g., temperature sensing). Moreover, the possible usage of IoT resource-constrained devices for privacy compliance checks opens additional challenging issues. Above all, since the IoT sector is rapidly developing commercial solutions, subject to fewer tests and distributed on a large scale to many users, the possibility that smart devices are subject to security vulnerabilities is still considerable [5]. Thus, performing a compliance check on an IoT device, under these conditions, means not always getting a reliable result. On the other hand, in the context of IoT we have to consider that, in accordance with the concept of edge computing, data processing is increasingly moving towards end-users. Therefore, privacy compliance check needs to be moved to the same level as data processing [6]. We believe that the rise of blockchain [7] as a new trusted and distributed computation paradigm can help in addressing these issues.

In this paper, we aim to benefit from blockchain to perform privacy preferences' compliance checks in a decentralized fashion on IoT devices, ensuring the correctness of the process through smart contracts.[1] This has the advantage of not having to rely on a centralized trusted

- *F. Daidone (fdaidone@uninsubria.it), B. Carminati (barbara.carminati @uninsubria.it), and E. Ferrari (elena.ferrari@uninsubria.it) are with the Department of Theoretical and Applied Science (DiSTA), University of Insubria, Italy.*

[1] Smart contracts are arbitrary owner-defined programs encoding blockchain transactions validation processes.

monitor to perform the check. Essentially, the compliance check is executed via smart contracts and validated thanks to a distributed consensus among the parties. Then, the result is immutably stored on the blockchain. Data streams generated by IoT devices are complemented with proper metadata, storing information on device owner's privacy preferences, by exploiting the Manufacturer Usage Description[2] (MUD) standard, which allows device manufacturers to implement communication policies on end-devices. The proposed system has been entirely developed and tested with different scenarios to show its performance and feasibility.

The remainder of this paper is organized as follows. Section 2 provides some background information. In Section 3, we discuss the architecture of the proposed solution, and then we deepen the two layers that compose it in Sections 4 and 5. We discuss the security properties in Section 6 and the experimental evaluation in Section 7. Later, Section 8 presents the related work, whereas Section 9 sets out the conclusions. The paper also contains Appendix A and B, containing an in-depth study of Hyperledger Fabric's chaincodes and formal proof of theorems provided in Section 6, respectively.

2 Background

In what follows, we briefly introduce some key concepts related to blockchain and the MUD standard that are needed to understand our proposal, as well as the model we adopt to specify privacy preferences.

2.1 Manufacturer Usage Description

The Manufacturer Usage Description (MUD) is an Internet Engineering Task Force (IETF) standard, which allows end devices to indicate to the network their communication needs. Let us consider, for instance, a smart light bulb. In general, this device does not require to interact with other smart home devices (such as heaters or coffee machines). The only connection needed is to the specific vendor service for remote control. As such, by properly configuring MUD, the light bulb owner can block any other unexpected communication. This allows to reduce the attack surface and block some malicious attempts to exploit the device for other purposes. The general MUD schema is as follows. The manufacturer defines, through a configuration file, the access policies and the type of network functionalities required by the device. The configuration can express that the light bulb has to communicate only on port 80 with https protocol to a specific server. When a new device joins a network, it communicates the URL where its MUD configuration file can be retrieved. Then, the network access device (NAD) retrieves the URL and sends it to the MUD manager. This component, usually placed in the local network, retrieves the MUD configuration file from the MUD file manager (aka, the smart light bulb vendor) and adapts the network setting such as to adhere to the MUD configuration (e.g., smart light bulb can connect only with its remote server). Although the standard currently covers only aspects related to network access control, the goal is to extend it to other fields, such as quality of service and data privacy. Indeed, the standard includes a MUD Extension[3] field designed at this purpose.

2.2 Blockchain

Blockchain is a distributed ledger where data are stored in chained blocks publicly accessible to the network nodes [7]. Any activity or exchange of resources made by network participants is stored in the blockchain as a transaction. Transactions are grouped and inserted into a block. Each block contains the hash of the previous block that creates a link between the blocks, comparable to a chain, making the blocks immutable. Before transactions are entered into the blockchain, peers of the network must agree on their validity. In distributed computing, this problem is known as consensus[4]. According to the way the network nodes are selected, a blockchain can be classified as permissionless or permissioned. A permissionless blockchain permits to anonymous nodes to participate in consensus. In contrast, in a permissioned blockchain, only selected nodes are authorized to join the network and participate in distributed consensus. Regardless of the permission type, all the major blockchain frameworks support smart contracts, enabling the inclusion of other functionalities with respect to standard validation operations. A smart contract is a program autonomously executed on the blockchain as part of a transaction process validation. Our proposal is based on Hyperledger Fabric[5], a permissioned blockchain that can manage large amounts of data with high performance. We refer the interested readers to Appendix A for more details about Fabric smart contracts.

2.3 Privacy model

In general, a privacy policy is specified by a consumer to mainly state which personal data it collects from individuals, for which purpose, for how long, and whether the collected data will be released to third-parties. On the other hand, data owners can specify their privacy preferences as constraints on each single privacy policy component (e.g., purpose, retention time, third-party release). The literature presents several models to represent user privacy preferences (see e.g., [9]). Although the proposed framework can work with several privacy models, we choose to adopt the model presented in [10], since this has been designed for the IoT scenario. This is an expressive privacy model that, in addition to standard privacy-related elements (e.g., purpose), also supports a set of features tailored to the IoT domain to allow data owners to limit how and which data can be derived during IoT analytic processes. Moreover, it allows the automatic generation of privacy preferences for newly derived data (e.g., information resulting from data fusion). In this proposal, as a first step, we have considered a lighter version of [10], such to focus mainly on the traditional privacy preference components (e.g., purpose,

[2] IETF-RFC8520 (MUD): https://tools.ietf.org/html/rfc8520

[3] Reporting MUD behavior to vendors - Available at https://tools.ietf.org/html/draft-lear-opsawg-mud-reporter-00.

[4] Proof-of-Work (PoW) or Proof-of-Stack (PoS) are example of BFT protocols [8]

[5] Available at https://www.hyperledger.org/

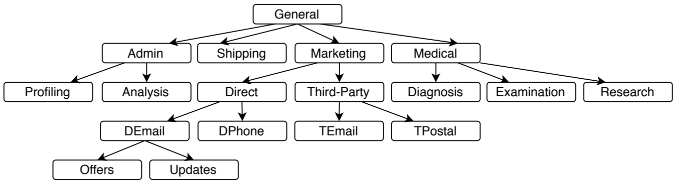

Fig. 1: An example of purpose tree

retention time). We postpone as future work to extend the framework to have full compliance with the model in [10].

According to [10], purposes are hierarchically organized into a tree structure PT, as shown by the example in Figure 1. Such tree structure is used to limit the number of purposes that need to be specified into a privacy preference. Indeed, a preference with the purpose $purp$ also implies the authorization for all the purposes rooted at $purp$ in PT. However, the model also allows the specification of exceptions to this propagation, as formalized by the following definition.

Definition 1 (Intended purpose [10]). An intended purpose ip is a pair (Aip, Exc), where Aip (allowed intended purposes) is a set of purposes belonging to a purpose tree PT and Exc (exceptions) is a set of purposes that descend from elements in Aip. ip authorizes the access for all purposes that descend from[6] the elements in Aip, except for those that descend from any element in Exc.

Now, it is useful to define the set of purposes implied by ip, denoted as $\overrightarrow{ip}$, to check the privacy compliance. This is the set of $purp$ in Aip including all their child nodes in PT. From this set, we remove all $purp$ in Exc, plus their child and parent nodes in PT.

Formally, let $\overrightarrow{Aip} = \bigcup_{purp \in ip.Aip} purp^{\downarrow}$[7], where $purp^{\downarrow}$ is the set composed of $purp$ and all purposes descending from $purp$ in PT, and $\overrightarrow{Exc} = \bigcup_{purp \in ip.Exc} purp^{\updownarrow}$, where $purp^{\updownarrow}$, the set composed of $purp$ and all purposes descending and ascending from $purp$ in PT, $\overrightarrow{ip} = \overrightarrow{Aip} - \overrightarrow{Exc}$.

A privacy preference is in turn defined as follows.

Definition 2 (Privacy preference [10]). A privacy preference is a tuple $pp = \langle \alpha, consumer, ip, rt, tpu \rangle$, where α is an attribute of a data stream generated by a smart object to which the policy refers to, $consumer$ specifies the set of consumer's identities to which pp applies, ip specifies the intended purposes for which α can be collected and used by any entity in $consumer$, rt specifies the retention time, and tpu the third party usage.[8]

Example 1. Let us consider the scenario of a smart home, equipped with MIoT devices for the remote monitoring. The basic symptoms monitoring system is composed by sensors for acquiring the respiration rate, heart rate, temperature, and oxygen saturation [11]. The sensed data must reach the hospital, namely the consumer, who takes care of the treatments. A patient can define a privacy preference pp for each data stream generated by the monitoring system. Consider, for instance, the heart rate monitoring. Suppose that a user wants to share the sensed data only with his/her hospital and only for administration and medical purposes, except for research. Moreover, he/she allows a retention time of 90 days and prohibits the dissemination of such information to third parties. To model these requirements, the user can specify pp as follows: $pp = \langle heart\text{-}rate, \{hospital\text{-}company\}, \langle \{admin, medical\}, \{research\} \rangle, 90d, unshareable \rangle$. By considering the purpose tree in Figure 1, the set of purposes authorized by pp is $\overrightarrow{ip} = \{admin, profiling, analysis, diagnosis, examination\}$.

A privacy policy can be modelled as a tuple $\langle \alpha, up, dataRet, dataRel \rangle$, where α denotes the attribute of a stream sensed by an IoT device to which the policy applies, up is the data usage purpose, whereas $dataRet$ is the data retention time, and $dataRel$ is the third party usage. The privacy compliance check compares a privacy preference with a privacy policy, in order to verify whether the consumer's data usage complies with the data owner's will.

3 Architecture

In this section, we introduce the overall architecture of the proposed blockchain-based privacy preferences enforcement framework, shown in Figure 2. The key idea is that data owners can leverage on blockchain for privacy compliance before their data are sent to consumers. We assume that both devices of data owner and consumers are registered on the blockchain. More precisely, we model a smart environment at the data owner side a set of connected IoT devices, hereafter IoT network, owned by a given user. The IoT network can sense data from the user environment, eventually locally elaborate them, and then send them to the consumer servers. We further assume that the IoT network is connected to consumer servers via a limited set of special IoT devices, called gateways. In our proposed solution, the gateways become the point of contact of an IoT environment with the blockchain. This implies that they also act as blockchain nodes, called IoT blockchain nodes, in addition to performing their gateway's functions. Data collected by the gateways are complemented with metadata encoding the owner's privacy preferences. Consumers register their privacy policies into the blockchain.

IoT manufacturers play a key role in the privacy enforcement process. They know their products in detail, for example, which network ports they use, which endpoints they connect to, which information they process and which instead they send to consumers, etc. Manufacturers can leverage the MUD standard to define the behavior of their IoT devices at internetworking level (e.g., MAC address, IP address, network port). With respect to data privacy, in our framework, the manufacturer can insert a by default privacy preference, called system-defined privacy preference pps, using the custom field provided by MUD (see, Section 2.1). This represents a privacy preference defined with the aim of providing the first level of privacy, by design and by default, to unaware users and therefore being compliant

[6] Hereafter, we assume this relationship as reflexive, meaning that every element descends from itself.

[7] In what follows, we use the dot notation $dataStructure.element$ to indicate an element inside a tuple or a data structure.

[8] For simplicity, in what follows, we assume that the retention time is expressed in days and the third party usage tpu assumes one of two values, namely $shareable$ or $unshareable$.

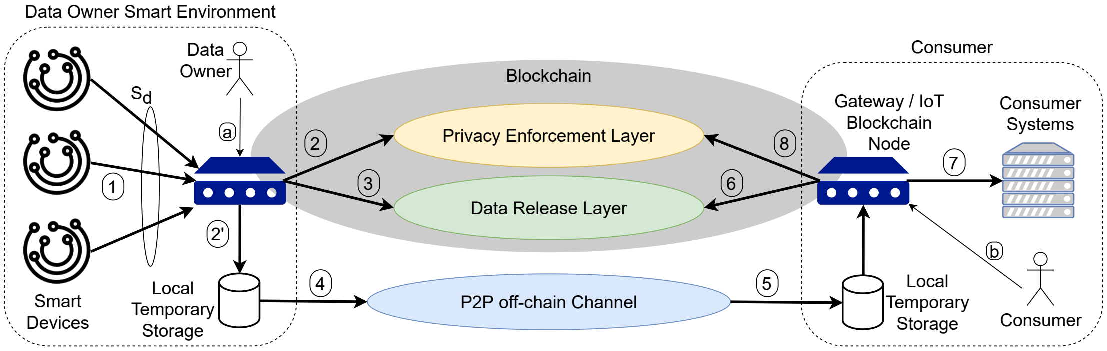

Fig. 2: Blockchain-based privacy enforcement workflow

with regulations, such as the GDPR[9]. In any case, the data owner is free to add further restrictions to system-defined privacy preferences, by specifying an owner-defined privacy preference *ppo*. In particular, the framework provides the functionalities to merge the owner and system-defined privacy preferences before performing a compliance check. As it will be described in Section 4, this process is carried out via a dedicated smart contract deployed in the blockchain. Hereafter, when we talk about privacy preference, we will refer to the one generated by the combination of the system-defined and the owner-defined privacy preferences.

We also rely on smart contracts for the enforcement of privacy preferences. In particular, the gateways analyze the collected data, and when they find a new privacy preference (e.g., a privacy preference that has not yet been evaluated), they trigger the execution of a smart contract implementing the privacy compliance check. This is designed to verify whether the consumer policy satisfies the constraints specified by the data owner in his/her privacy preference. Finally, we save on-chain the proof of the results returned by the compliance check for auditing purposes.

In devising our framework, we have to consider that, by design, information stored in the chain is distributed among all blockchain nodes. Each node can see what is saved on-chain as well as the smart contract contents. The privacy enforcement does not treat sensitive information, and therefore any node can execute the smart contract without affecting the data owner's privacy. Moreover, the proposed architecture leverages on a permissioned blockchain that can be configured such that only given stakeholders are authorized to join the privacy compliance check.[10] Once privacy enforcement has been executed (i.e., its smart contract has been validated), we exploit the blockchain to enforce the data release process. This implies to release owner's data only to consumers that satisfy his/her privacy preferences. Since data could be sensitive, we cannot store them directly in the blockchain. For this reason, we assume that once the data have reached the gateway, they are kept in local temporary storage, at the gateway side, waiting to be released to the authorized consumers. To coordinate the data release, we need a communication channel able to directly connect the gateway holding the data to the blockchain node corresponding to the authorized consumer. This channel must be private, avoiding thus any other nodes in the blockchain to access exchanged data. At this purpose, we exploit the *private data* mechanism, natively supported by Hyperledger Fabric. Thanks to this, Hyperledger Fabric can create peer-to-peer links between two or more nodes to exchange off-chain information, keeping on-chain an evidence of the data exchange. In particular, all communications take place via an encryption layer, established through the Transport Layer Security (TLS). Therefore, when the data owner's privacy preferences have been verified, another smart contract takes care of data release. It is executed directly by the gateway hosting the data to be released. Its execution aims at moving the data from the temporary local storage to the authorized consumer via the P2P channel supported by Hyperledger private data. The sensitive data are processed only within the data owner's IoT blockchain node (i.e., the gateway acting as blockchain node). The remaining nodes can only see the hash of this data, stored on the blockchain as a result of the smart contract execution. To enforce the above-mentioned steps (i.e., privacy enforcement and data release), we leverage on a permissioned blockchain to create two groups of blockchain nodes, namely the privacy enforcement and the data release layer (see Figure 2) to which we assign different privileges on transactions/smart contracts. Nodes on the privacy enforcement layer implement the privacy compliance check, whose outcome is used by nodes on the data release layer. Both layers are discussed in detail in Sections 4 and 5, respectively.

4 Privacy Enforcement Layer

In this section, we illustrate the three phases carried on by the privacy enforcement layer: privacy preference enforcement, tuple grouping, and enforcement audit.

4.1 Privacy preference enforcement

As depicted in Figure 2, data sensed by IoT devices are sent to gateways, that act as blockchain nodes (see step ① in Figure 2). IoT devices send their data inside a data tuple t_d, formally defined as follows.

Definition 3 (Data tuple). Let S_d be a stream containing data sensed by an IoT device. A data tuple t_d in S_d has the following structure: $\langle idS, sn, d, hash(d)\rangle$, where d is the

[9] General Data Protection Regulation - EUR-Lex 32016R0679: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[10] Permissioned blockchains allow us to have an integrated identity management system relying on public key infrastructure to identify authorized stakeholders.

sensed data, $hash(d)$ is the hash value of d, idS is the id of S_d, whereas sn represents the sequence number of t_d in S_d.

We use hashes to let consumers check that the received data are exactly those sent by IoT devices and have not been manipulated or corrupted (see Section 4.3). To avoid further overload of the blockchain, the hash values are directly computed by IoT devices. For instance, the tuple $t_d = \langle stream01, 32, \text{“}heart\text{-}rate : 60\text{”}, \text{“}de6960c7\text{”}\rangle$ can be used to encode heart rate information of a patient equipped with a wearable device to sense heartbeats. Before sending their data, data owners must register their data streams on gateways. Consumers can then subscribe to registered data streams. This subscription is subject to data owner acceptance. Once the data owner accepts a new subscribed consumer, he/she communicates a vector of subscribed consumers' **consumerVector**[11] to the gateway (see step ⓐ in Figure 2). The vector is formed by consumer identifiers idC and wrapped into a tuple $t_{cv} : \langle idS, \mathbf{consumerVector}\rangle$, where idS is the target stream identifier. On the other hand, consumers register their privacy policies in the blockchain (see step ⓑ in Figure 2). A consumer's gateway sends to the blockchain the privacy policy p related to a stream idS through a privacy policy tuple t_p with the following structure: $\langle idS, idC, p\rangle$, where idC is the consumer identifier, idS is the data stream identifier, whereas p denotes a privacy policy specified as explained in Section 2.3. When the blockchain receives t_p, it executes a tailored smart contract to store the tuple with the identifier $idTp$ and to create an association between the consumer idC and the stream idS.

We recall that, to make the task of privacy preferences specification easier, we assume that IoT manufacturers create system-defined privacy preferences, with the aim to also protect those data owners with little awareness of the privacy risks related to data disclosure. Such preference is stored into the MUD extension field. Moreover, the preference is also stored in the blockchain for future audits (e.g., in case the MUD file is no longer available on the manufacturer server). A skilled data owner can specify his/her own privacy preferences, called owner-defined privacy preferences. An owner-defined privacy preference complements the system-defined privacy preference, making it more restrictive. The gateway, upon receiving the data owner's privacy preference (see step ⓐ in Figure 2) and catching the MUD URL coming from the corresponding IoT device, sends this information to blockchain injecting into the corresponding stream a privacy preference tuple t_{pp}, formally defined as follows.

Definition 4 (Privacy preference tuple). Let S_d be a stream containing data sensed by a smart device. A privacy preference tuple t_{pp} in S_d has the following structure: $\langle idS, sn, mudUrl, ppo\rangle$, where idS is the id of S_d, sn represents the sequence number of t_{pp} in S_d, $mudUrl$ is the URL pointing to the manufacturer's MUD file of the corresponding IoT device, and *ppo* is the owner-defined privacy preference, specified according to the model in Section 2.3.

In case of legacy devices, which are not compatible with the MUD standard, the $mudUrl$ field will be left blank. An empty *ppo* field means that the data owner has not specified his/her own privacy preference. When both the *ppo* and $mudUrl$ fields are empty, it is equivalent to not execute any privacy enforcement.

The privacy enforcement smart contract (see Pseudocode 1[12]) includes the *submitPrivacyPreference()* function that generates a unique *pp* starting from the input privacy preferences provided in t_{pp}. It is started when the blockchain receives a new t_{pp}. As first step, it checks that the sequence number is as expected, to avoid inconsistencies and conflicts of privacy preference (see, Line 6). When the tuple t_{pp} contains only the data owner-defined privacy preference, that will become the actual *pp*. The same holds if only a system-defined preference is contained into the tuple. When the privacy preference tuple contains both a system and an owner-defined preference, the resulting *pp* is obtained by combining these two in such a way that they must be both satisfied in order to deliver the protected data to the requesting consumers. Preferences combination is implemented by the $ppCombine()$ function (Pseudocode 1). The generic $''join*** (pps.*, ppo.*)''$ function combines each element of the two privacy preferences. At the end, the function returns the resulting *pp*, which is saved in the blockchain as a parameter of t_{pp}. In addition, we also append the system-defined privacy preference *pps* retrieved from the manufacturer server. The execution of the privacy enforcement smart contract is then validated by the network via distributed consensus, ensuring thus the correct compliance check.

Example 2. Returning to Example 1, a privacy preference tuple for the heart rate can be: $t_{pp} = \langle stream01, 24, \text{“}manufacturerUrl\text{”}, dataOwnerPP\rangle$, where "stream01" is the stream id, "24" is the tuple id, "manufacturerUrl" is the url of the MUD file containing *pps*, "dataOwnerPP" is the owner-defined privacy preference, which we assume being the *pp* of Example 1, that is: $\langle heart\text{-}rate, \{hospital\text{-}company\}, \langle\{admin, diagnosis\}, \emptyset\rangle, 90d, unshareable\rangle$. In this example the data owner grants access to his/her sensitive data. The system-defined privacy preference acts by excluding any purpose which is not GDPR compliant (e.g., to ensure that consumers cannot speculate on medical data). This can be encoded by the following system-defined privacy preference: $pps = \langle\emptyset, \emptyset, \langle\emptyset, \{marketing\}\rangle, \emptyset, \emptyset\rangle$. The combination of the two privacy preferences that will be saved on blockchain is: $pp = \langle heart\text{-}rate, \{hospital\text{-}company\}, \langle\{admin, diagnosis\}, \{marketing\}\rangle, 90d, unshareable\rangle$.

When the blockchain receives a new t_{pp}, the contained privacy preference *pp* takes effect substituting the previous one. This implies to run the privacy compliance check of the new preference against privacy policies of consumers subscribed to the related stream (same idS). This check is performed by the *privacyComplianceChecker()* function (Pseudocode 1). The function is triggered by the *submitPrivacyPreference()*

[11] All vectors are denoted in lowercase bold.

[12] Functions $''getBc***(k)''/''putBc***(k,v)''$ read/write the data with key k and value v on the blockchain, respectively.

function each time a new t_{pp} is received. After obtaining the privacy preference tuple and the consumer vector, *privacyComplianceChecker()* verifies which of the consumers have the authorization to receive the data in the stream to which t_{pp} refers to. This is done by leveraging on the $verifyAuth()$ function (Pseudocode 1). The purpose check (see, Line 8) is satisfied if up specifies an allowed purpose, that is, a purpose contained in $\overrightarrow{ip}$ (see, Section 2.3). Also, $dataRet$ must be less than or equal to rt (see, Line 11), whereas $dataRel$ must match tpu (see, Line 14). If all checks are satisfied, then the compliance check successes. The result is then stored in the blockchain. More precisely, since the function is executed for each subscribed consumer, all returned results are collected into a unique vector:

$$\begin{aligned} \textbf{check} &= (check_1, check_2, ..., check_n) \\ s.t. \quad check_j &= (idC, idTp, grant) \end{aligned} \tag{1}$$

where n is the number of subscribed consumers, $grant$ is the value returned by $verifyAuth()$ for consumer whose id is idC, and $idTp$ is the privacy policy identifier. As introduced in Section 4.1, when t_p is submitted to the blockchain, an identifier $idTp$ is generated, referenced on-chain with the keys idC and idS. $idTp$ is retrieved from blockchain via the $getBcIdTpByIdSIdC()$ function (see, Line 19). Finally, the **check** vector is saved in the blockchain with $idCheck$ as key.

4.2 Tuple grouping

In a realistic IoT scenario often many devices in the IoT network push their tuples to the gateway simultaneously. However, blockchain might not be able to manage a high incoming rate of tuples, such as gateways might produce. Therefore, the blockchain can become a bottleneck. To avoid this, we group data tuples to which the same privacy preference applies. This selection has the purpose to lighten the workload on the smart contract and therefore on the blockchain. By grouping tuples with similar privacy protection requirements, the blockchain can process them in one round. Indeed, when the smart contract receives a group of tuples that share the same idS and pp, it can perform the compliance check only once and apply it to the whole group. To correctly create the tuple selection σ, we must take into account the validity range of pp for a given data stream S_d, that is, the set of tuples to which it applies. The range of tuples to which pp applies begins with the privacy preference tuple referring to pp and ends with the subsequent privacy preference tuple. We call this data stream subset *privacy preference scope*, denoted as S_{pp}. Another important dimension is the time interval on which the selection acts. Considering different streams as input to the gateway, the selection must wait for the collection of a certain number of data tuples. We can limit the maximum waiting time and the maximum number of tuples per selection to be sent to the blockchain. By tuning these two parameters, we create queues within the gateway with different priorities and capacities. For example, in the case of streams that need low latency and small data size, such as near real-time, we can select a low accumulation time and a high number of tuples in a single selection. Otherwise, if the flow contains big size data that can be delayed, such as batch processing, the queue may have a higher grouping time than in the previous case. We have considered these two extreme scenarios, but the platform is also capable of handling mixed scenarios, by using the same approach. Obviously, different strategies must be adopted to tune the window and balance the latency according to the considered application scenario. In any case, σ contains a certain number of data tuples collected in a time interval, that respects the constraint of having the same stream identifier and the same privacy preference applied to all its tuples. Each tuple in the selection has its own sequence number, so we can define an interval as two integers $[n, m]$, where n and m are the sequence numbers of the first and the last tuple, respectively. $\sigma_{[n,m]}(S_d)$ represents a selection of tuples $t_d \in S_d$ such that t_d belongs to S_{pp} and $n \leq t_d.sn \leq m$.

For the obvious limitations of the blockchain, we must include further optimizations with respect to the use of data hashes contained in each data tuple (see, Def. 3). Therefore, we exploit a hash function to compute and store only a digest, representative of a whole tuple selection. Let us consider a hash function $hash()$ and n messages $m_1, m_2, ..., m_n$, the digest is calculated as $d = hash(hash(m_1)||hash(m_2)||...||hash(m_n))$, where $||$ denotes the concatenation operation. Keeping only one digest in the blockchain allows us to combine in a single hash all the tuples over which the control has been made, with a considerable saving of space. More precisely, from a selection $\sigma_{[n,m]}(S_d)$ we obtain the set of tuples t_d composing it (see, Figure 3). For each t_d, we select only the field $hash(d)$. Finally, we compute the cascading hash on the resulting set of hash values.

After that, for each t_d in $\sigma_{[n,m]}(S_d)$, we collect d into a data vector **d**. The data vector is saved on temporary local storage, ② in Figure 2, and it is not shared with anyone, but it is only accessible locally by the data owner.

Leaving aside **d** (it will be discussed in Sec. 5), we represent the tuple selection $\sigma_{[n,m]}(S_d)$ on the blockchain with a small size tuple suitable for on-chain storage, called chunk tuple t_c, whose structure is:

$$\begin{gathered} \langle idS, idT_{pp}, [n, m], idCheck, digest(\mathbf{h}) \rangle \\ s.t. \quad \mathbf{h} = \textstyle\prod_{hash(d)}(\sigma_{[n,m]}) \end{gathered} \tag{2}$$

where **h** is the projection on component $hash(d)$ of selection $\sigma_{[n,m]}$, idS is the identifier of the stream to which the selection refers to, idT_{pp} is the selection's privacy preference identifier, $[n, m]$ is the interval of data tuples' sequence numbers belonging to the selection, $idCheck$ is the privacy compliance check identifier, whereas $digest()$ is the digest built on **h**. t_c is created by function $dataTupleChunk()$ of the privacy enforcement smart contract (see, Pseudocode 1 and step ② in Figure 2). It receives σ as input, from which it derives idS, **h**, and $[n, m]$. The two identifiers $idTpp$ and $idCheck$ are obtained from blockchain with key idS. Next, we calculate the digest (see, Line 33) and then store t_c, with its identifier $idTc$, on the blockchain. Figure 3 shows a graphic representation of the tuple selection process. At the top, each square represents a tuple, those with the bold border are privacy preference tuples, while the others are data tuples. The colors green, red, and yellow represent tuples from three different data streams. Focusing on the green stream, "idS: 1", we can see two privacy preference

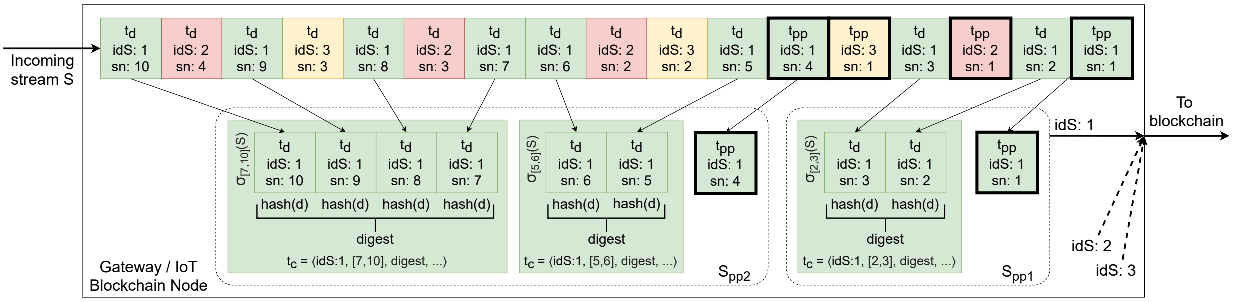

Fig. 3: Tuple selection process inside the gateway

scopes S_{pp}, the first with a selection, and the second with two selections. For instance, the selection $\sigma_{[7,10]}(S_d)$ consists of the tuples from sequence number 7 to 10 in the stream $idS : 1$ belonging to the privacy preference scope S_{pp2}.

4.3 Privacy preference enforcement audit

Our framework allows data owners to check whether the compliance of their privacy preferences with the consumers' privacy policies has been correctly carried on for a certain data tuple. Regarding consumers, they can check the outcome of their privacy policy enforcement and whether the received data set is complete.

Mainly, the data owner has the right to verify the enforcement of his/her privacy preferences over his/her data and to which consumers they have been released. To proceed with the audit verification of a specific data tuple t_d, the data owner needs to determine the chunk tuple t_c where the outcome of the compliance check for t_d is kept. Knowing idS, he/she can go back to t_c thanks to the sn component of t_d. Once the data owner gets t_c from the blockchain, he/she can carry out the checks itself. First, the data owner can verify that the tuples contained in the range $[n, m]$ have been enforced with the privacy preference t_{pp}, retrieved from $idTpp$. Another check is on the vector **check**, kept with $idCheck$ on blockchain. Specifically, each consumer has his/her own $check$ inside **check** vector. By doing so, the data owner gets the list of consumers enabled to receive the data. A third more in-depth check can be carried out on each individual consumer, to re-evaluate privacy enforcement on t_{pp} and t_p. In this case, the data owner must carry out the check on his/her own and verify that the outcome matches the one stored in the $grant$ component of the element referring to that consumer in the **check** vector.

On the other hand, the consumer receives t_c and **d**. From t_c he/she can see the outcome of the privacy enforcement carried on his/her privacy policy and, if he/she wants, recalculate it with t_{pp} and t_p for double-check, like the data owner. Instead, to verify the completeness of the received data set, he/she can calculate the entire digest of **d** and compare it with the one contained t_c. If they match, then the consumer has received all the data.

5 Data Release Layer

The data release layer transfers an authorized data item d from the data owner IoT blockchain node to a consumer IoT blockchain node. Since d may contain sensitive information, they cannot be exchanged on-chain. For this reason, we exploit a peer-to-peer (P2P) off-chain private channel to exchange data between the involved parties. For this purpose, in this paper, we leverage on private data provided by Hyperledger Fabric. Private data allows the establishment of a P2P link between two or more nodes for data sharing, saving on the blockchain a trace of the exchange.

More precisely, the data release layer operates on the output of the privacy enforcement layer (see, Section 4), that is: (1) the data d to be released to the intended consumers, which are saved on local temporary storage as data vector

Pseudocode 1: Privacy enforcement smart contract

```
Let:
$tpp$ be the privacy preference tuple;
$idTpp$ be the privacy preference tuple identifier;
$\sigma$ be the tuple selection ;
Function submitPrivacyPreference (tpp)
    checkSequenceNumber($tpp$);
    $mudFile$ = downloadMudFile($tpp.mudUrl$);
    $tpp.pps$ = extractPps($mudFile$);
    $tpp.pp$ = ppCombine($tpp.pps$, $tpp.ppo$);
    new $idTpp$;
    putBcTpp($idTpp$, $tpp$);
    privacyComplianceChecker($idTpp$);
end
Function privacyComplianceChecker ($idTpp$)
    $tpp$ = getBcTpp($idTpp$);
    $consumerVector$ = getBcConsumerVector($tpp.idS$);
    check = new vector;
    forall $idC$ in $consumerVector$ do
        $idTp$ = getBcIdTpByIdSIdC($tpp.idS$,$idC$);
        $tp$ = getBcTp($idTp$);
        $grant$ = verifyAuth($tpp.pp$, $tp.p$);
        check.add($idC$,$idTp$,$grant$);
    end
    new $idCheck$;
    putBcCheck($idCheck$,check);
end
Function dataTupleChunk ($\sigma$)
    $idS$ = getIdS($\sigma$);
    $idTpp$ = getBcIdTppByIdS($idS$);
    h = getH($\sigma$);
    $[n, m]$ = getInterval($\sigma$);
    $idCheck$ = getBcIdCheckByIdS($idS$);
    $digest$ = calculateDigest(h);
    $tc$ = ($idS$, $idTpp$, $[n, m]$, $idCheck$, $digest$);
    new $idTc$;
    putBcTc($idTc$, $tc$);
end
Function ppCombine (pps,ppo)
    new $pp$;
    $pp.ip$ = joinIp($pps.ip$,$ppo.ip$);
    $pp.rt$ = joinRt($pps.rt$,$ppo.rt$);
    $pp.tpu$ = joinTpu($pps.tpu$,$ppo.tpu$);
    return $pp$;
end
```

Pseudocode 2: Verify authorization function

```
Let:
$pp$ be the data owner's privacy preference;
$p$ be the consumer's privacy policy ;
Function verifyAuth ($pp$,$p$)
    Let $ipFlag$ be a boolean variable, initialized as False;
    Let $rtFlag$ be a boolean variable, initialized as False;
    Let $tpuFlag$ be a boolean variable, initialized as False;
    if ($p.up \in pp.\overrightarrow{ip}$) then
        $ipFlag$=True;
    end
    if ($p.dataRet \leq pp.rt$) then
        $rtFlag$=True;
    end
    if ($p.dataRel = pp.tpu$) then
        $tpuFlag$=True;
    end
    if ($ipFlag$=True & $rtFlag$=True & $tpuFlag$=True) then
        return True;
    else
        return False;
end
```

d, and (2) the chunk tuple t_c, stored on the blockchain. The purpose is to send **d** to the allowed consumers. The data release is managed by a smart contract, described in Pseudocode 3, and executed by the data owner IoT blockchain node. The data release smart contract leverages the $dataRelease()$ function, which sends data to consumers according to the compliance check results. The function receives as input parameters (1) and (2) and verifies that each consumer has received a positive result from the compliance check performed by *privacyComplianceChecker()* (see, Pseudocode 1), otherwise data are not released.

First, let us get t_c by $idTc$ (see, Line 5, Pseudocode 3), where is contained $idCheck$ (it represents a pointer within the blockchain to the privacy enforcement check). We use $idCheck$ to derive the vector **check** from blockchain (see, Line 6, Pseudocode 3). Inside this vector, we have the outcome of privacy enforcement per each consumer. When the smart contract finds a consumer, who is entitled to receive the data, it gets the private channel identifier and sends **d** on the P2P off-chain channel through a tuple $(idTc, \mathbf{d})$. The function $putPrivateData()$ (see, Line 10 of Pseudocode 3) takes care of sending $(idTc, \mathbf{d})$ to the consumer on an Hyperledger Fabric private channel. Since in a blockchain network any node can be vulnerable and attacked, data contained in **d** are encrypted and signed by the IoT devices that generate them. Hyperledger Fabric provides a Public Key Infrastructure (PKI) in which each node has its own identity with a public and a private key (wallet). Taking advantage of this feature, an IoT device can send through the P2P off-chain channel a symmetric key to a consumer, encrypting it with the consumer's public key. The consumer uses the Hyperledger Fabric $getPrivateData()$ function in order to obtain his/her data. This function requires the channel identifier $idChannel$ and the chunk tuple identifier $idTc$ as input parameters. The consumer knows the channel identifier $idChannel$ as this is released at subscription time, the chunk tuple identifier $idTc$ because the P2P off-chain channel has a built-in push notification mechanism that alerts the consumer when a new chunk tuple is available. After that, **d** is sent to the consumer (⑦ in Figure 2).

Pseudocode 3: Data release smart contract

```
1  Let:
2  $idTc$ be the chuck tuple identifier;
3  d be the data vector;
4  Function dataRelease ($idTc$,d)
5      $tc$ = getBcTc($idTc$);
6      check = getBcCheckVector($tc.idCheck$);
7      forall check in check do
8          if $check.grant$ is True then
9              $idChannel$ = getChannel($check.idC$);
10             putPrivateData($idChannel$, $idTc$, d);
11         end
12     end
13 end
```

6 Security discussion

In this section, we discuss the security guarantees provided by each layer of our framework, as well as of the underlying infrastructure. In general, we assume that each component of our infrastructure (e.g., blockchain, gateways, and IoT devices) is untrusted and controlled by different entities that might have conflicting interests.

Infrastructure: To ensure secure communication among IoT devices, gateways, and blockchain peers, we leverage the authentication and encryption mechanism features provided by Hyperledger Fabric via Membership Service Provider (MSP)[13], which is based on TLS protocol and X.509 certificates.

A further key component of the proposed infrastructure is the blockchain, which might be subject to vulnerabilities and threats, e.g., double spending, smart contract coding flaws, Sybil attack, etc. In our framework, to cope with these threats we adopted the countermeasures designed for Hyperledger described in [12]. Moreover, we rely on a permissioned blockchain and, as such, blockchain's clients and peers are known, having they own identities. Therefore, they are accountable for their behaviors, and any perpetrator of abuse or malicious behavior is easily identifiable and can be banned from the network. We also use the raft consensus algorithm which is considered safe and preventing double spending by design [12]. Although the blockchain allows us to run a reliable privacy enforcement even in an untrusted environment, we have to consider smart contracts' security. Smart contracts are prone to various programming errors that can lead to bugs and/or vulnerabilities. They can be exploited to manipulate the workflow of the smart contract and obtain different results from those expected. To cope with this issue, we have implemented a set of strategies, such as imposing maximum time for smart contract execution, setting root privileges only where necessary, enabling access control lists (ACLs) for channel access, setting policies for chaincode lifecycle and endorsement, testing input parameters against parameter tampering, testing the chaincodes with static analysis tools (revive^CC[14] and gosec[15]).

[13] Membership Service Provider - Available at https://hyperledger-fabric.readthedocs.io/en/latest/msp.html

[14] Available at https://github.com/sivachokkapu/revive-cc

[15] Available at https://github.com/securego/gosec

Privacy enforcement layer: Attacks on the correct functioning of the privacy enforcement layer can be done by malicious IoT devices or gateways that try modifying the preference/policy tuples. Our scenario foresees different IoT devices each with its own tailored privacy preference, specifically the system-defined privacy preference pps (cfr. Section 3). For the sake of scalability, in the default setting of our framework IoT devices generate and sign privacy preference tuples. So it is possible that a compromised IoT device, generates a fake privacy preferences, in conflicts with the real owner preferences. However, the data owner is able to verify on the blockchain the stored privacy preference, sent by the IoT device, and detect the attack. This solution could fit many IoT deployment and achieve a good compromise between security and scalability. However, to cope with more risky scenarios (e.g., characterized by resource-constrained IoT devices that can be easily compromised), the proposed infrastructure supports the data owner to directly submit in the blockchain his/her privacy preferences. This would exclude any tampering by compromised IoT devices.

Another security issue is due to possible DoS attack, where IoT devices try to send an excessive number of privacy preference tuples to continuously trigger the privacy enforcement. We consider that the gateway is equipped with standard countermeasures against DoS (queuing systems, detection, and monitoring system, IoT devices' ban, log and notification, etc.), so it can detect and mitigate by avoiding sending the tuples to the blockchain. In the case that also the gateway participates in the attack, the blockchain administrators can monitor the number of transactions submitted, set rate limiter, identify, and isolate malicious IoT devices or gateway.

Besides participating in DoS attacks, a malicious gateway could modify, duplicate, or omit the privacy preference tuples. We recall that IoT devices sign these, thus any tamper by the gateway is easily identifiable by the blockchain due to signature invalidation. The privacy preference tuples duplications can be identified thanks to the sequence number that cannot be altered, since it's included in the signature. Also, the privacy preference tuples omission can be detected by proposed smart contracts (see, Line 6, Pseudocode 1), which know the expected tuple sequence number and raise an alert in case of wrong sequence number. Finally, the correctness of the privacy enforcement process relies on the correctness of the proposed smart contracts. This has been proven by Theorem 1 available in Appendix B.

Data release layer: A possible security issue in this layer is represented by data tuples sent to unauthorized entity (e.g., consumers, peers). This could happen due to (1) untrusted gateway that sends the collected data to unauthorized entity or (2) an attacker who directly eavesdrops the communication. We have to recall that data sensed by IoT devices are encrypted with their private keys before their release (cfr. Section 5). Thus, even if data are shared with unauthorized entities by an untrusted gateway or eavesdropped by attacker, these are not accessible. Furthermore, the risk of eavesdropping is limited by the TLS protocol adopted by Hyperledger Fabric. A data leakage could also occur at the IoT device, as a malicious IoT device could send the sensed data to unauthorized entity. To cope with this possibility, we leverage on MUD (cfr. Section 2.1), by which we can limit the IoT device communication only to the gateway and make the IoT device t not directly reachable remotely. Further security problems are given by the omission and modification of data tuples by the gateway. To cope with this threat, we adopt the same countermeasures seen in the previous section, e.g., exploiting auditing, signature, and sequence numbers. Finally, the logical correctness of the data release smart contract has been proven by Theorem 2, available in Appendix B.

7 Evaluation

In this section, we present the evaluation of our solution with a realistic load. At this purpose, we run a set of experiments aiming at measuring: (1) the data throughput, that is, the amount of data that the blockchain can manage in a time unit; (2) the space overhead implied by the additional information (metadata) that our solution requires to insert in the original IoT device streams; and (3) the time spent in privacy preference enforcement and data release. In running the experiments, we considered two main dimensions that impact performance. The first is the number of owner's IoT devices registered in the blockchain, that is, the number of data streams to be processed. Indeed, each distinct inbound data stream requires a distinct aggregation process in order to create t_c. Moreover, since the compliance checks have to be performed against each registered privacy policy, the second dimension is the number of registered consumers. In our state of the art analysis (cfr. Section 8) we did not find any work directly comparable to our proposal, as they differ in type of infrastructure, scenario, blockchain, and adopted privacy model.

7.1 Test environment

We simulated a smart home scenario exploiting both Raspberry Pi 3[16] devices and virtual machines[17]. The adopted smart home scenario consists of four different entities: (1) a gateway collecting data owner streams; (2) blockchain peers supporting the dialogue with consumers (i.e., policy registration, stream subscription); (3) an IoT device manufacturer, and (4) a third-party entity. This latter represents those entities that are not directly involved in the data generation/release but whose peers participate in the blockchain consensus. Moreover, we assume that each entity joins the blockchain with two peers and a certificate authority[18]. For data owners, consumers, and IoT manufacturers, we have implemented a peer on a Raspberry Pi 3 device, whereas the other peers and the certificate authority run virtualized on the server. Instead, peers and certificate authorities of the third party entities run only on the server.

In the experiments, we varied the number of data owners and consumers by changing the load on the corresponding blockchain peers. For the implementation, we used the latest stable Hyperledger Fabric release, that is, 1.4 version. We

[16] Model B+: Cortex-A53 (ARMv8) @ 1.4GHz, 1GB LPDDR2.

[17] Running on server Intel Core i7-6700 @ 3.4Ghz, 16GB DDR4

[18] In Hyperledger Fabric, each entity must have its own certificate authority for the generation of the cryptographic information (PKI keys, certificates, etc.).

used the Raft consensus with 5 orderers, running on the server. To implement a blockchain peer on Raspberry Pi 3, we installed Ubuntu 18.04.2 4.15.0-1033-raspi2 AArch64 OS and compiled Hyperledger Fabric for arm64 platform. For the virtual peers, we used Ubuntu server 18.04 with peers virtualized on a switched Gigabit LAN. The chaincode was developed in Go[19], whereas we used the Hyperledger Fabric SDK for Node.js[20] to interface peers with clients.

The experiments were conducted by exploiting the CASAS project dataset[21], which is widely used by the scientific community. In particular, we used the "Two-resident apartment" dataset consisting of sensors' data collected from a house lived by two people. The dataset contains data gathered from 108 different devices for several days. For simplicity, we consider only 24 hours of a unique data stream. This interval is representative as the other days' pattern is similar due to daily activities' cyclical nature. To implement a more complex scenario, we duplicated this stream to simulate the flow generated by different houses. Each of them has a different data owner, with a different privacy preference. We assumed a minimum of 10 streams. At the beginning, for each stream, we assumed only one consumer with a single privacy preference. Then, we increased these numbers up to 100 subscribed consumers and 100 privacy preferences associated with each stream.

7.2 Performance results

In this section, we present the test results for throughput, space, and time overhead. We assess the performance by testing the scenario with different settings, by varying the number of data owner streams and consumers from 10 to 100 (i.e., 10, 50, 100). We changed privacy preferences and policies every 10 minutes to trigger the new compliance check computation, stressing the system.

Throughput: Figure 4 shows the 24h throughput for four settings, where s and c indicate the number of streams and consumers, respectively (e.g., 10s10c stands for 10 streams and 10 consumers setting). We omitted the 50 streams and 50 consumers configuration for lack of space, but they are taken into account in the following graphs. The $byteIn$ input throughput is the number of bytes in a second, that the system through the IoT blockchain nodes can accept. The $byteOut$ output throughput is the number of bytes in a second confirmed on the blockchain and that have reached the consumer. $byteOut$ and $byteIn$ have timestamps associated with them, the difference between these two timestamps represents the time interval during which the system performs all needed operations: selection, aggregation, privacy enforcement, data release, and blockchain consensus on each data. This time is the platform transaction time. Focusing on the outgoing stream ($byteOut$), we notice that the trend is jagged for a few consumers; instead, it becomes flat and constant for a higher number of consumers. This is because the system is not fully loaded initially and thus follows the input trend and has high overhead. When the incoming stream saturates the blockchain's speed of accepting transactions, the gateway's queuing and selection mechanism temporarily store the packets, reducing the outgoing overhead. Although there are higher peaks with few consumers, the average throughput trend is higher in the second case because the sending is continuous. The maximum throughput, on configuration *100s100c*, is around 9000 B/s. Certainly, this throughput is not comparable with the classic IoT systems that achieve higher performance but it is a first milestone in the use of the blockchain in this sector. In fact, being able to successfully manage 100 smart environments, creating about 10,000 data flows (assuming that each IoT device had its own consumer and therefore with a ratio of 1:1) is a good result that paves the way for subsequent experiments.

Overhead: In this experiment, we evaluated the impact that the metadata required by our solution implies. We measure overhead, after tuple grouping, as the ratio of metadata (t_c) to the overall value of metadata and data ($t_c, \mathbf{d}$), in percentage. Figure 5 shows the 24h average overhead for nine settings. In general, from our experiments we have seen that a data tuple t_d has a dimension of 272 bytes, of which 136 bytes of data (d) and 136 bytes of privacy metadata (idS, sn, $hash(d)$). Looking at the 100s100c throughput configuration with the peak value of 9000 B/s and knowing that we were able to output a $\mathbf{d}$ data vector of 8432 bytes (62 d data values of 136 bytes), the overhead is about 568 bytes, that is 6.3% of the total size. In this case, we reach the minimum overhead of the output stream. The overhead has the most significant impact when there are few transactions at the entrance and, therefore, little possibility of aggregating them. In fact, we achieved the minimum average overhead of 21% with 100 streams and 100 consumers. The overhead chart highlights the efficiency of the system on the aggregation of tuples and privacy compliance checks. The negative trend confirms the scalability of the platform as streams and consumers increase.

Transaction time: Figure 6 shows the 24 hours average transaction time for different settings. Our framework is able to reach around 6 transactions per second (0.17 s/tx), with an average of 3 transactions per second (0.33 s/tx), as shown in Figure 5. The average transaction time is almost constant as the number of streams changes because the data owner gateway aggregates the stream and lightens the blockchain. We note a time increase when the number of consumers increases because the compliance checks on privacy preferences and policies increase. Despite the increase in incoming streams, the graph shows that we are able to maintain an almost constant processing time and that it mainly depends on the number of consumers. This means that the system is able to scale well with respect to the number of input streams, thanks to the aggregation techniques used. Based on the obtained results, we estimate that our proposal can handle about 10 buildings[22] concurrently with 10 dwellings each and latency under 400 milliseconds. This confirms the feasibility of our approach, also considering the large margins for improvement in the use of more performing hardware, software, and networks [13].

[19] Available at https://golang.org/

[20] Available at https://hyperledger.github.io/fabric-sdk-node/

[21] Available at http://casas.wsu.edu/datasets/

[22] Calculation based on the Italian average number of dwellings per buildings https://entranze.enerdata.net/average-number-of-dwellings-per-building.html

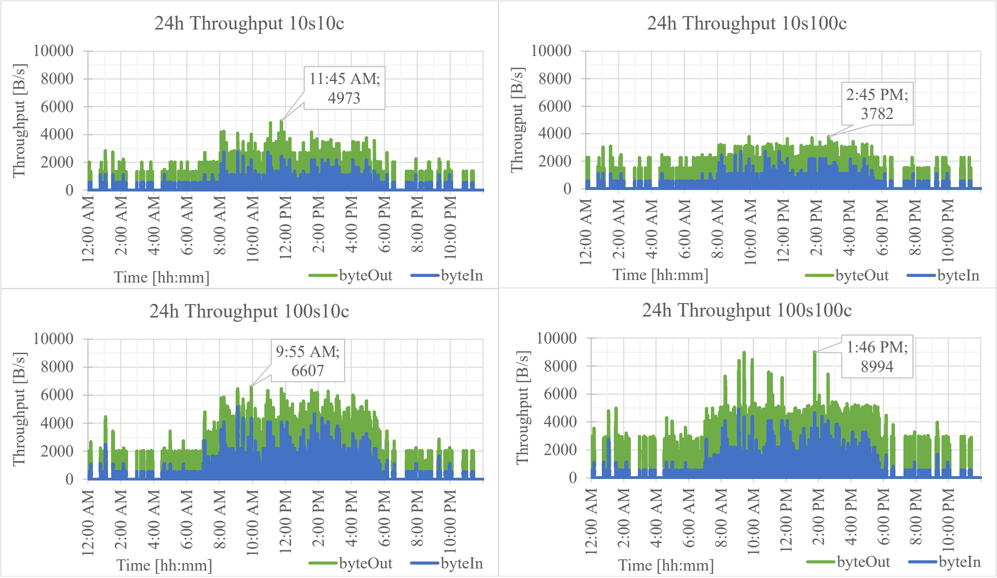

Fig. 4: 24h Throughput

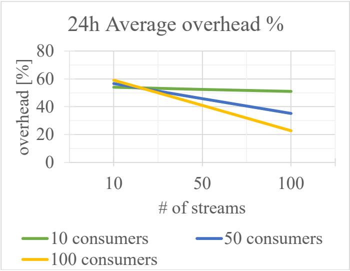

Fig. 5: 24h average overhead

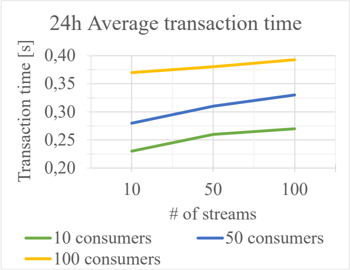

Fig. 6: 24h average transaction time

8 Related work

The IoT field has experienced considerable development and, as a result, security and privacy have been studied deeply. In what follows, we briefly review those proposals closer to our proposed framework. Many papers have addressed the issue of privacy preferences enforcement in the IoT domain. Over the years we have seen a gradual migration of the policy enforcement process from the cloud to the fog. The authors in [14] present the Policy Enforcement Fog Module (PEFM), achieving better privacy enforcement results with less overhead than a cloud-based solution. The work presented in [10], from which we borrow the privacy model adopted in our framework, introduces a decentralized privacy enforcement framework for the IoT scenario. Users can specify privacy preferences on local gateways which control the release of the data to the consumers. [15] presents the Privacy Preference for IoT (PPIoT) Ontology and provides methods and examples to apply it in the IoT context in light of the GDPR. The use of machine learning is also increasingly present in the privacy enforcement process, for instance [16] demonstrates this trend. The authors proposed how machine learning can use data sources for more robust privacy management. An overall view of the different technologies and methodologies for IoT privacy enforcement is given by a recent comparative study [17]. It provides an analysis of the state-of-the-art of the main IoT frameworks and focuses on GDPR support. The results show that many frameworks are still not GDPR-compliant, and there are still open challenges, such as usable user interfaces, lack of privacy risk analysis and so on. Our framework has a number of innovative features compared to the abovementioned proposals, the main is that it is a fully decentralized solution for privacy preferences' enforcement, leveraging on blockchain, that also guarantees transparency of the enforcement process for all the involved parties without the need of a trusted entity.

Other related work are those adopting blockchain for solving different security and privacy issues in the IoT domain. [18] proposes a ”miner”, that is, a high resource device, installed in every smart home for communication between IoT devices and blockchain. Here, the blockchain is used to store policies and for access to resources, adding and removing the devices authorized for communication. In the medical field, healthcare organizations need data storage systems that protect patient privacy but, at the same time, enable data sharing for medical research. Therefore, blockchain has been investigated for its characteristics of transparency, trust, and immutability of data, as in [19] and [20], where the authors propose that sensitive patient information be encrypted and stored in the blockchain. Unlike our system, data are not shared as private data, but stored permanently in the blockchain. The general-purpose architecture proposed in [21] uses Hyperledger Fabric to protect communication between IoT devices, using asymmetric encryption. The blockchain includes malicious actor detection and guarantees non-repudiation and privacy. The work presented in [22] implements a reliable ownership management system for medical IoT devices that uses Ethereum and smart contracts instead of Trusted Third Parties. In the review study presented in [23], the authors analyzed the vulnerabilities of medical IoT devices and then proposed a solution for the exchange of patients personally identifiable information, leveraging on blockchain. [24] presents a system for the exchange of patient data via blockchain, where data are encrypted, signed with the digital ring model, and then stored on the cloud. [25] focuses on wearable medical devices only, where a smart device, such as a smartphone, collects raw data and communicates with the blockchain. At the end, it sends an alert to the hospital to report the presence of data on the blockchain. Another work closely related to ours is [26], a platform for monitoring patient vital signs using smart contracts on Hyperledger Fabric. PATRIoT [27] is an Ethereum IoT data sharing platform. They use an ontology-based privacy model called LIoPY. Unlike our platform, their privacy enforcement process is done off-chain, using the blockchain only to organize the data exchange between data producer and consumer.

Our proposed framework greatly differs from the abovementioned proposals in that none of them: (1) is highly user-centric with the possibility of specifying fine-grained data owner privacy preferences (2) is designed for the massive audit of the privacy preference enforcement process.

9 Conclusion

In this paper, we presented a blockchain-based privacy enforcement framework for IoT smart environments. Blockchain provides a distributed and immutable system for carrying out privacy enforcement and preserving its outcome, so that it can be used for future audits, guaranteeing non-repudiation. We plan to extend this work along many directions. First of all, we want to enable the blockchain to perform privacy-preserving complex event processing, so as to move the processing of the tuples from the consumer to the blockchain. We will also expand the privacy model in order to manage extended privacy preferences on derived data. We also plan to define optimization strategies for the execution of privacy preferences' enforcement on the blockchain and design an off-chain self-designed data release layer.

Acknowledgments

This work has received funding from CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 Research and Innovation program under grant agreement No 830927, and from RAIS (Real-time analytics for the Internet of Sports), Marie Skłodowska-Curie Innovative Training Networks (ITN), under grant agreement No 813162.

References

[1] V. Sivaraman, H. H. Gharakheili, C. Fernandes, N. Clark, and T. Karliychuk, "Smart iot devices in the home: Security and privacy implications," *IEEE Technology and Society Magazine*, 2018.

[2] J. a. George W. Jackson and S. Rahman, "Exploring challenges and opportunities in cybersecurity risk and threat communications related to the medical internet of things (miot)," 2019.

[3] X. R. Ding, D. Clifton *et al.*, "Wearable sensing and telehealth technology with potential applications in the coronavirus pandemic," *IEEE Reviews in Biomedical Engineering*, 2020.

[4] P. Schaar, "Privacy by design," *Identity in the Information Society*, 2010.

[5] O. Alrawi *et al.*, "Sok: Security evaluation of home-based iot deployments," in *IEEE Symposium on SP*, 2019.

[6] S. Parikh, D. Dave *et al.*, "Security and privacy issues in cloud, fog and edge computing," *Procedia Computer Science*, 2019.

[7] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, 2017.

[8] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International workshop on open problems in network security*, 2015.

[9] K. Adam, B. Price, M. Richards, and B. Nuseibeh, "A privacy preference model for pervasive computing," in *Proceedings of the First European Conference on Mobile Government*, 2005.

[10] G. Sagirlar, B. Carminati, and E. Ferrari, "Decentralizing privacy enforcement for internet of things smart objects," *Computer Networks*, 2018.

[11] M. Islam, S. Mahmud, M. Jibril, M. Islam, S. Nooruddin, and S. Ayon, "Wearable technology to assist the patients infected with novel coronavirus (covid-19)," *SN Computer Science*, 2020.

[12] S. Brotsis, N. Kolokotronis *et al.*, "On the security and privacy of hyperledger fabric: Challenges and open issues," in *2020 IEEE World Congress on Services (SERVICES)*, 2020.

[13] C. Gorenflo, S. Lee *et al.*, "Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second," in *IEEE ICBC*, 2019.

[14] A. Al-Hasnawi, S. M. Carr, and A. Gupta, "Fog-based local and remote policy enforcement for preserving data privacy in the internet of things," *Internet of Things*, 2019.

[15] O. R. Sanchez, I. Torre, and B. P. Knijnenburg, "Semantic-based privacy settings negotiation and management," *Future Generation Computer Systems*, 2020.

[16] M. Amiri-Zarandi, R. A. Dara, and E. Fraser, "A survey of machine learning-based solutions to protect privacy in the internet of things," *Computers & Security*, 2020.

[17] A. D. Kounoudes and G. M. Kapitsaki, "A mapping of iot user-centric privacy preserving approaches to the gdpr," *Internet of Things*, 2020.

[18] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *IEEE Int. Conf. PerCom Workshops*, 2017.

[19] G. Magyar, "Blockchain: Solving the privacy and research availability tradeoff for ehr data: A new disruptive technology in health data management," *IEEE 30th NC*, 2017.

[20] O. Choudhury, N. Fairoza, I. Sylla, and A. Das, "A Blockchain Framework for Managing and Monitoring Data in Multi-Site Clinical Trials," *arXiv e-prints*, 2019.

[21] U. Satapathy, B. K. Mohanta *et al.*, "A secure framework for communication in internet of things application using hyperledger based blockchain," in *Int. Conf. ICCCNT*, 2019.

[22] M. Alblooshi, K. Salah *et al.*, "Blockchain-based ownership management for medical iot (miot) devices," in *Int. Conf. IIT*, 2018.

[23] R. Soni and G. Kumar, "A review on blockchain urgency in the internet of things in healthcare," in *Int. Conf. ICISS*, 2019.

[24] A. Dwivedi, G. Srivastava, S. Dhar, and R. Singh, "A decentralized privacy-preserving healthcare blockchain for iot," *Sensors*, 2019.

[25] K. Griggs, O. Ossipova *et al.*, "Healthcare blockchain system using smart contracts for secure automated remote patient monitoring," *Journal of Medical Systems*, 2018.

[26] F. Jamil, S. Ahmad, N. Iqbal, and D.-H. Kim, "Towards a remote monitoring of patient vital signs based on iot-based blockchain integrity management platforms in smart hospitals," *Sensors*, 2020.

[27] F. Loukil *et al.*, "PATRIoT: A data sharing platform for IoT using a service-oriented approach based on Blockchain," in *ICSOC*, 2020.

Federico Daidone is a Ph.D. student in Computer Science at the University of Insubria, Italy. He received his master degree in Telecommunication Engineering with high honors from the University of Palermo, Italy, where he mainly focused on Smart Grids. The main research topics in his Ph.D. are related to decentralizing and enhancing data security and privacy in IoT, and improving performance and applicability of blockchain.

Barbara Carminati is a professor of Computer Science at the University of Insubria, Italy. She holds a Ph.D. in Computer Science from the University of Milano, Italy. Her main research interests are related to security and privacy for innovative applications. Barbara Carminati has been involved in the organization of several international conferences as program committee member as well as program and general chair.

Elena Ferrari is a professor of Computer science at the University of Insubria, Italy, where she leads the STRICT Socialab. Her research activities are mainly related to data security, privacy and trust. Ferrari has a Ph.D. in Computer Science from the University of Milano, Italy. She received several awards, including the IEEE Computer Society's 2009 Technical achievement award for 'outstanding and innovative contributions to secure data management.' She is an IEEE and ACM fellow.