Authorization transparency for accountable access to IoT services

Luca Ferretti*, Francesco Longo†, Michele Colajanni*, Giovanni Merlino†, Nachiket Tapas†

*University of Modena and Reggio Emilia, Italy
{luca.ferretti,michele.colajanni}@unimore.it
†University of Messina, Italy
{flongo,gmerlino,ntapas}@unime.it

***Abstract*—Highly distributed smart environments, such as Smart Cities, require scalable architectures to support a large number of stakeholders that share Internet of Things (IoT) resources and services. We focus on authorization solutions that regulate access of users to smart objects and consider scenarios where a large number of smart objects owners want to share the resources of their devices in a secure way. A popular solution is to delegate third parties, such as public Cloud services, to mediate authorization procedures among users and smart objects. This approach has the disadvantage of assuming third parties as trusted proxies that guarantee correctness of all authorization procedures. In this paper, we propose a system that allows to audit authorizations managed by third parties, to detect and expose their misbehaviors to users, smart objects owners and, possibly, to the public. The proposed system is inspired by the transparency projects used to monitor Web Certification Authorities, but improves over existing proposals through a twofold contribution. First, it is specifically designed for IoT devices, provided with little resources and distributed in constrained environments. Second, it complies to current standard authorization protocols and available open-source software, making it ready to be deployed.**

***Keywords*-IoT, access control, authorization, authentication, transparency**

I. Introduction

IoT networks are becoming more and more pervasive and populate most of our life-spaces, including houses, workplaces, cars, and entire cities. They are strictly tied to the physical world, either by monitoring our environments, in case of sensors and alarms, or by having direct control over our environments, such as smart locks connected to our houses and vehicles. In largely distributed IoT networks, where a large number of heterogeneous stakeholders share smart objects, a very important aspect is a secure access control management system. In this context, where smart objects are provided with low resources and are placed in constrained environments, access control is usually delegated to a third party, such as a public Cloud service. Examples include research literature proposals [1], [2], [3] and commercial products (e.g., AWS and Google IoT [4], [5]). However, all the existing proposals consider third parties as trusted actors, i.e., as someone that will grant access on IoT resources to all and only the users authorized by the smart object owners.

In this paper, we propose a system that increases the security of IoT environments by allowing to monitor the behavior of intermediate parties that manage authorization procedures. Our proposal relies on strong cryptographic primitives and assumes that third parties could be malicious. We say that the system enables *accountable access to IoT services* because it does not prevent the authorization service from behaving maliciously, but allows both owners and users to detect wrong or malicious behaviors by the authorization services, and to show proofs of that behaviors to the public. This is a strategy that allows to build a system that is practical (i.e., guarantee good performance for the considered scenario) and that offers strong security guarantee in presence of adversaries that have a reputation, such as the Cloud services that operate authorization services. Existing solutions based on audit logs allow to monitor the security of Cloud-managed systems against external attackers [6], but do not consider powerful attackers that operate within the service provider. Moreover, proposals that improve the security guarantees by using cryptographic protocols [7], [8], [9], [10] or public distributed ledgers [11], [12] cannot be deployed in IoT environments due to their high computational, storage and network requirements.

The contribution of the paper is twofold. First, we propose a system for auditable authorizations with strong (cryptographic) guarantees. Existing proposals with similar goals are typically designed to maintain public information (e.g., Web certificates [7]), and cannot be used in the context of secret authorization information. Second, contributing to ongoing efforts of the community in the context of IoT environments, we design the system in order to comply with Web standards for authorization protocols in constrained environments [13].

The manuscript is organized as following. Section II compares our proposal with related work. Section III analyzes the considered system and threat models with regard to existing authorization protocols. Section IV describes the details of the proposal, including its architecture and protocols, and compares it to existing proposals. Section V concludes the paper and discusses future work.

II. Related work

Security solutions in well-known IoT authorization services include adoption of consolidated authorization protocols and log systems (e.g., Google Access Transparency [6][1]) that improve security by allowing the service administrators and even the users to monitor all mechanisms of the system. However, they are not able to prevent attacks by malicious attackers within the authorization service infrastructure, such as admins that have complete access on these systems.

Approaches to guarantee strong security of outsourced services aim at allowing to detect violations to data integrity or even to the correctness of entire algorithms and protocols. Data integrity can be protected by using Digital Signatures, MACs and authenticated data structures [14], [15]). A few approaches exist for detecting algorithms correctness, including advanced cryptographic primitives (e.g., verifiable computation [16]), secure hardware enclaves (e.g., SGX [17]), and distributed protocols based on consensus assumptions (e.g., secure multi-party computation [18], blockchain protocols [11], [19]).

In this paper, we aim at designing a practical and secure solution that can be deployed in IoT environments. We avoid approaches based on non-practical cryptographic primitives, or those based on hardware technologies that seem still vulnerable to attacks [20] that prevent their immediate adoption in critical environments. Our proposal is more related to solutions based on *transparency logs* [21], [7], that allow to outsource a log service to semi-trusted parties without affecting the security of the system, and on (permissionless) distributed ledger technologies [22], [11], [23], [21], [24] and smart-contracts, that allow to maintain a public, distributed and modifiable log [11], [19]. Our proposal is based on standard cryptographic primitives that guarantee the security of the system without affecting its efficiency. To the best of our knowledge, no existing proposals based on transparency logs consider the challenges related to auditing authorizations of IoT devices, as proposed in this paper.

The most popular *transparency log* solution is Certificate Transparency (CT) [7], that is a system for monitoring Certification Authorities by requiring browsers to verify that all certificates fetched from Websites have been stored in approved authenticated logs. Literature also proposed a more general approach called *key transparency* [21], that allows to prevent the so-called *equivocation*, that is, attacks based on binding different cryptographic keys to the same identity. Neither solution can be trivially applied to the considered scenario due to the different characteristics of the delegated authorization protocols, including management of secret information and updates to the authorization policies.

[1]Despite the General Transparency project and Transparency logs are both maintained by Google and share similar names, they are completely independent projects, with different scopes, aims, and security assumptions.

Existing proposals for distributed ledgers often provide advanced functionalities but incur in security or performance disadvantages. The protocol proposed in [23] penalizes misbehaving entities by using time-locked deposits in the form of Bitcoins and accountable assertions, offering an additional guarantee with regard to the proposed approach at the expense of a less efficient protocol with regard to similar solutions [11]. We note that having penalties automatically applied to misbehaving entities is not a critical requirement in reputation-based systems, where parties are discouraged due to fear of public scrutiny (such as the one considered in this paper, where most authorization services are deployed by well-known companies on Cloud services). Authors of [24] propose a public auditing system based on distributed ledgers to store interactions between IoT entities as evidence to support investigation during attacks. However, to preserve the confidentiality of data that cannot be exposed publicly, the system involves a trusted third-party to ensure the confidentiality of the data. Finally, a hybrid protocol that combines transparency logs and permissionless distributed ledger is represented by [21], that extends the key transparency approach [25] by integrating Ethereum blockchain to eliminate the need of auditors. The approach cannot be applied to the considered IoT scenario due to the limitations of the underlying log and blockchain systems, but we consider integration of our proposal to distributed ledgers as a future work.

The state-of-the-art regarding public logs maintained on the Bitcoin network is represented by the Catena protocol [11], where the authors model the equivocation guarantee first analyzed by [21] as a double-spending problem in the Bitcoin network. Moreover, the systems proposed in [19], [26] detect malicious authorization services by using smart-contracts on the public Ethereum network. Any proposal based on distributed ledger technologies have disadvantages that prevent their adoption in many resource-constrained, such as high storage requirements on things, the need to rely on trusted third parties, or to connect to the public network of the ledger. We compare our system with proposals based on permissionless distributed ledgers [11], [19] in Section IV-E.

III. System and threat models

We outline the scope and guarantees of our proposal. First, we describe a reference IoT scenario that motivates the design of the system (Section III-A). Second, we model the scenario with regard to established authorization frameworks (Section III-B). Finally, we discuss the security threats and guarantees of the proposed system (Section III-C).

A. Reference scenario

We consider a real-world example represented by an online rental service that acts as a broker between room owners and potential guests. To improve the flexibility of the service, each room is equipped with a smart lock that

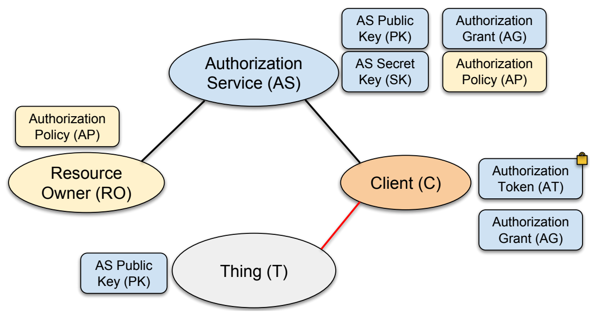

Figure 1: Reference architecture for delegated authorizations

is compliant with standard WiFi technologies available on any smartphone. Room owners subscribe to the service and delegate it with the capability of allowing guests to open the room door by interacting with the associated smart lock through their smartphones. Once an agreement is in place (e.g., upon successful payment), the guest should be authorized to open the smart lock in an unsupervised way during the agreed time frame.

Let us assume that the guest tries to open the smart lock during her renting period. The interaction between the guest, the service, and the lock is composed by three phases. First, the guest interacts with the service, issuing a request to open the lock. Second, the service validates the request and decides whether the guest should be allowed to open the lock. The third operation depends on the outcome of the second one: if the service decides that the guest is authorized, it grants her access to the room by releasing the due authorization material that allows her to open the smart lock; if the guest is not authorized, the service denies the guest access by not releasing the authorization material.

The aim of the proposed system is to allow both the room owners and the guests to detect whether the rental service is behaving correctly, allowing (1) the owner to detect illegitimate authorizations, and (2) both the owner and the guests to detect illegitimate denied authorizations.

B. *Reference architecture and operations framework*

We describe the architecture for delegated authorizations by referring to Figure 1. We consider four roles: *Things* (*T*), *Resource Owners* (*RO*), *Clients* (*C*) and *Authorization Services* (*AS*). *Things* represent cyber-physical IoT devices provided with sensors and/or actuators that can communicate locally (e.g., via wireless local and/or personal area networks), but might not have reliable Internet connectivity (e.g., very slow, discontinuous, expensive, or no connectivity at all). *Resource owners* represent entities (e.g., a person, a company) that regulate accesses to things. They have access to devices with moderate capabilities (e.g., modern smart-phones, personal computers) and to reliable Internet connectivity, but do not own large infrastructures to deploy highly available and scalable services. *Authorization services* represent highly available and scalable services (e.g., Web services deployed on a public Cloud infrastructure) that act on behalf of resource owners to control access to things. *Clients* represent entities that need to get access to things (e.g., information provided by thing-hosted sensors, access to a facility controlled by thing-hosted actuators). They are authorized by resource owners to access things and must interact with authorization services to obtain the authorization material required by things to validate their requests.

The architecture includes five types of data: *Authorization Policies* (*AP*), *Authorization Grants* (*AG*), *Authorization Tokens* (*AT*), and *Authorization Service Secret* and *Public Keys* (*SK*, *PK*). *Authorization policies* are rules that regulate access to things by clients. The semantic of the acceptable rules depend on the adopted access control model and are orthogonal to the authorization protocol. *Authorization grants* are authentication tokens used by authorization services to validate clients requests. They are reference tokens, that is, they are opaque values with no implicit information that refers to authoritative information stored within the database of the authorization service (e.g., they could be implemented as long unique identifiers chosen at random). Authorization grants usually have long expiry times and can be easily revoked before-hand. *Access tokens* are authentication tokens used by things to validate clients requests. They are self-contained cryptographic tokens that include all the due information to validate a request without accessing any database (e.g., a JSON Web Token). As they cannot be easily revoked before-hand, access tokens usually have short expiry times. *Authorization service secret and public keys* are cryptographic keys used to sign and verify access tokens. While the secret key is only known by the authorization service, the public key is public information that is known by all parties.

We consider that the parties deploy network services as described in Table I, where: the first and second columns (*server* and *client*) show the parties that offer and that use the service, respectively; the third column (*operation interface*) shows the interface of the service, including input and output data; the fourth column (*description*) describes what is the purpose of the operation.

The workflow of the operations that allow a client to access a thing is as following. *(1)* The resource owner inserts a new authorization policy AP_{RO} at the authorization service by using the *UpdatePolicy* service, that answers with the due return value (*‘accept’*) to confirm the acceptance of the policy. *(2)* A client requests an authorization grant to access a thing by sending an authorization policy AP_C to the authorization service by using the *Authorization* routine. If the requested authorization AP_C complies to the authorization policies AP_{RO} released by the resource owner, the authorization service answers with an authorization grant *AG*. *(3)* The client uses the authorization grant *AG* to request an access token *AT* to the authorization service by using

Server	Client	Operation interface	Description
AS	*RO*	$\{\textit{‘accept’}, \textit{‘reject’}\} \leftarrow \textit{UpdatePolicy}\,(AP)$	Update authorization policy
	C	$\{AG, \textit{‘reject’}\} \leftarrow \textit{Authorization}\,(AP)$	Request an authorization grant *AG* for the authorization policy *AP*
		$\{AT, \textit{‘reject’}\} \leftarrow \textit{Token}\,(AG)$	Request an access token *AT* with regard to the authorization grant *AG*
T	*C*	$\textit{result} \leftarrow \textit{Request}\,(\textit{data}, AT)$	Access the thing by using the access token *AT* and send payload *data*.

Table I: Plaintext operations framework of the reference architecture

the *Token* routine. The authorization service will accept the request only if the authorization grant has not expired and has not been revoked. *(4)* The client issues a request to the thing by sending the (request) *data* and the access token *AT* with the *Request* operation. The thing approves the request only if the access token is valid.

We note that the proposed model represents an instance of the OAuth2 standard for delegated authorizations [27] and related extensions for constrained environments [13]. As a minor variant, we denote as *things* the *resource servers* of the OAuth2 framework to highlight that they are devices characterized by limited resources and connectivity. As a major difference, while in the standard frameworks authorization and resource servers are controlled by the same authority, in the proposed model things are controlled by resource owners and are independent of authorization services. As a result, our proposal shares many design choices with the existing frameworks, but also provides additional guarantees that are driven by the additional system requirements and security threats.

C. Threat model and security guarantees

We assume that resource owners, authorization services and clients have known identities, and can establish secure and mutually authenticated connections by using any standard Web protocols for secure communications and credentials systems. Our proposal does not limit the adoption of any of these solutions. We do not assume that things and clients are able to establish secure channels in local networks. In case of non-secure channels, they must use the due cryptographic protocols to validate access tokens and, optionally, to protect the security of the application data. These are design choices that are orthogonal to our proposal. Finally, we assume that things know the public keys of the authorization services. As an example, public keys could be flashed by an OEM at the factory or configured by an admin in the local network.

The reference architecture guarantees security of the system against all known attacks operated by clients (e.g., a client that might try to access a thing illegitimately) and by external attackers (e.g., impersonation attacks) in the considered security model. For a comprehensive discussion about these attacks and about additional technicalities for their correct implementation the reader can refer to [13] and [28].

We distinguish from any standard authorization framework by considering weaker security assumptions, where parties may behave dishonestly by not operating all protocols correctly. First, we assume that authorization services could issue authorizations that do not comply with the policies defined by the resource owners, that is, they may illegitimately deny or release an authorization to a client that do not comply with the authorization policies defined by the resource owners. The proposed system guarantees that a client (or a resource owner) is able to detect these misbehavior and publicly accuse the authorization service by showing the due cryptographic proofs. A second security threat rises as a consequence of this security guarantee, that is, the possibility for the authorization service of being accused illegitimately by other parties [8], [9]. Our proposal considers this threat and propose mechanisms that allow authorization services to defend themselves from false accusations.

IV. Proposed system for authorizations transparency in IoT environments

We describe the proposed system for auditable authorizations in IoT environments. First, we describe the adopted access control model (Section IV-A). Second, we outline the architecture and operations framework of the system (Section IV-B). Third, we give details on the main auditable authorization protocol (Section IV-C) and for verifying the correct behavior of the authorization service (Section IV-D). We conclude by comparing the proposal to other approaches (Section IV-E).

A. Access control model and notation

We describe the details of the types of data adopted by the reference architecture proposed in Section III-B for a discretionary access control model. For ease of presentation, we consider a simple model that does not distinguish access privileges (e.g., read, write) and multiple resources available on a thing. That is, a client authorized to access a thing can operate any type of operation on all its resources. This model does not pose limitations to many IoT scenarios, such as the smart-lock motivating scenario described in Section III-A. Note that more complex access control models, such as fine-grained attribute-based models, can be used as-well without limitations.

We represent an access policy as a tuple $AP = \langle AR, tn, V \rangle$, where *AR* denotes an access rule, *tn* denotes the issue time of the authorization policy and *V* denotes the validity period of the rule. Each access rule identifies the clients that can

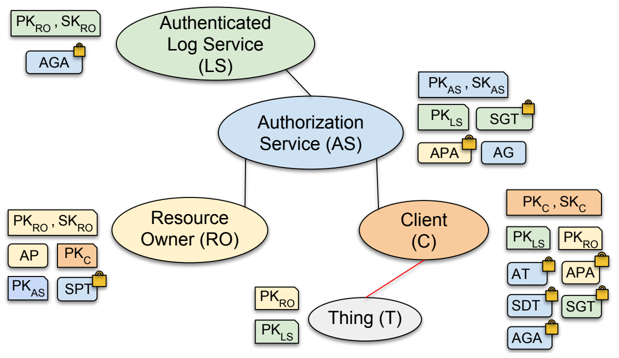

Figure 2: IoT transparency architecture

access to a thing as the tuple $AR = \langle C, T \rangle$, where client C can access thing T. The validity period defines when the access rule is to be considered valid and is represented as $V = \langle nb, na \rangle$, where nb and na denote the *not before* and *not after* time instants that define the time range, similarly to x509 Web certificates.

An authorization grant, that is a reference token, is implemented as a unique string identifier $AG = id$. An access token is a cryptographic bearer token defined as the tuple $AT = \langle AP, V, \sigma_{AT} \rangle$, where AP is a due authorization policy that authorizes the client to access a thing, V is the token validity period (with the same semantics described in the case of an access rule), and $\sigma_{AT} = sign_{SK_{AS}}(AP, V)$ is a proper signature of the authorization policy with the secret key of the authorization service.

B. Architecture overview

We describe the proposed architecture by referring to Figure 2. It extends the reference architecture for delegated authorizations (see Section III, Figure 1) by introducing an additional role, the *Authenticated Log Service* (*LS*), that represents a service that stores information about authorizations granted by the authorization service. This service is hosted by an additional authority, external to resource owners, authorization services and clients, at an highly available Web service. The service allows the authorization service to insert information about authorizations, while other parties (and possibly the public) can query it. All information stored within the service is authenticated via *authenticated data structures*, allowing to detect any modifications on the data by a malicious service.

The architecture includes all the types of data managed in the reference architecture: *authorization policies*, *authorization grants* and *access tokens*. Moreover, resource owners, authorization services, clients and authenticated log services maintain public and secret keys, that we denote as $\langle PK_{RO}, SK_{RO} \rangle$, $\langle PK_{AS}, SK_{AS} \rangle$, $\langle PK_C, SK_C \rangle$ and $\langle PK_{LS}, SK_{LS} \rangle$, respectively. We assume that each secret key is only known by its owner, while public keys are known by all parties. Finally, the architecture includes *cryptographic attestations* that authenticate the information exchanged by the parties.

The security of the proposed approach is based on three main design choices. *(a)* The system requires each party to operate requests by using *request attestations*, which are cryptographic data structures that allow to prove their (mis)behavior to third parties. *(b)* The authorization service, that is the root of trust for all issued authorizations, is forced to log all its operations to the external authenticated log service, thus allowing other parties and potentially public third parties to monitor its behavior. *(c)* By using authenticated data structures, the only security assumption on the authenticated log service is that it is always available, i.e., that it answers to all operations requests. If the service tries to misbehave by returning incorrect answers, the other parties are able to detect them.

For ease of presentation, we distinguish attestations in two categories: *request attestations* and *signed response timestamps*[2]. The *request attestations* authenticate the data sent by a party in a request. In this category we identify *authorization policy attestations* (*APA*) and *authorization grant attestations* (*AGA*). For ease of presentation, we pospone the details of their design to Section IV-C, that discusses the protocols in which they are used. The *Signed Response Timestamps* assess the acceptance of a request that applies modifications on the state of the service (e.g., insert or update data). In this category we identify the *signed policy timestamp* (*SPT*), *signed grant timestamp* (*SGT*) and *signed denial timestamp* (*SDT*). All signed response timestamps are computed with regard to a request as the following function $\mathcal{SRT}(\cdot)$:

$$\mathcal{SRT}(req) := \langle \mathcal{H}(req), nb, \sigma \rangle, \qquad (1)$$

where $\mathcal{H}(\cdot)$ is a deterministic collision-resistant hash function (e.g., SHA256), nb is the *"not before"* timestamp that defines at which time-instant the modifications required by the request are to be considered applied at the service, and σ is the cryptographic signature (e.g., ECDSA) of the service applied on the tuple $\langle \mathcal{H}(request), nb \rangle$.

In Table II we show the operations framework of the architecture. The operations *UpdatePolicy*, *Authorization*, *Token* and *Request* are variants of those proposed in the reference architecture (see Section III-B, Table I). *AccuseDenial* and *VerifyDenial* are additional operations offered by the authorization service and by the resource owner that allow parties to operate audit operations. Finally, operations *InsertGrant*, *QueryGrant* are services offered by the authenticated log service to store and query information by other parties. To focus on a solution that is ready to be applied in real-world applications, we consider operations that can be implemented by using the Trillian software [29] as a

[2]We adopt the notation *signed timestamp* from the *General Transparency* project [29] (see Section II).

Server	Client	Operation interface	Description
AS	*RO*	$\{SPT, \text{'reject'}\} \leftarrow UpdatePolicy\left(APA\right)$	Update of an authorization policy
	C	$\{\langle AGA, SGT\rangle, SDT\} \leftarrow Authorization\left(AP\right)$	Request an authorization grant
		$\{AT, \text{'reject'}\} \leftarrow Token\left(AG\right)$	Request an access token
		$APA \leftarrow AccuseDenial\left(SDT, SPT\right)$	Accuse of illegitimate authorization denial
T	*C*	$result \leftarrow Request\left(data, AT, SGT\right)$	Request access to the thing
LS	*AS*	$SGT \leftarrow InsertGrant\left(AGA\right)$	Insert an authorization grant
	RO	$\pi \leftarrow QueryGrant\left(AGA\right)$	Request proof for an authorization grant
RO	*C*	$\{SPT, \emptyset\} \leftarrow VerifyDenial\left(SDT\right)$	Request verification of authorization denial

Table II: Operations framework of the IoT transparency architecture

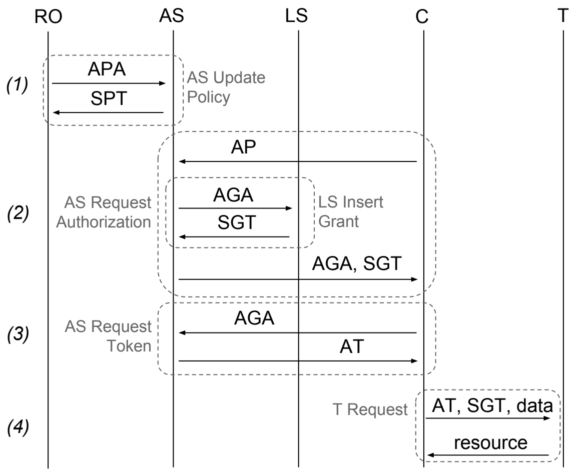

Figure 3: Accountable authorization protocol

back-end, that is a production-ready open-source project maintained by Google as a reference implementation for Certificate Transparency log services (see Section II).

In the following, we describe the details of each operation within the main protocols of the proposed system, that are *accountable authorization protocol*, *verification of denied authorizations* and *verification of issued authorizations*.

C. Accountable authorization protocol

The *accountable authorization protocol* allows clients to obtain the due authorization material to access things from the authorization service. It extends the original protocol for delegated authorizations described in Section III-B to enable accountability. In the following we describe the four phases of the protocol by referring to Figure 3.

(1) The resource owner delegates the authorization service by using the *UpdatePolicy* routine. He builds an authorization policy attestation as the tuple $APA = \langle id, AP, \sigma_{APA}\rangle$, where *id* denotes a unique identifier of the attestation, *AP* is the authorization policy (as defined in the reference protocol) and σ_{APA} is the signature of the tuple $\langle id, AP\rangle$ generated by using the secret key of the resource owner. The service answers by returning the signed policy timestamp *SPT* as a proof of acceptance of the policy, computed as $SPT = \langle \mathcal{H}\left(\text{APA}\right), nb, \sigma_{SPT}\rangle$.

(2) The client requests an authorization grant to the authorization service to access a thing by using the *Authorization* routine. It builds a proper authorization policy, as described in Section III, that is validated by the authorization service with regard to the known authorization policies. If legitimate, the authorization service builds the authorization grant attestation $AGA = \langle \mathcal{H}\left(AG\right), AP, tn, V, \sigma_{AGA}\rangle$, and inserts it in the authenticated log service by using the *InsertGrant* routine. The authenticated log service returns a signed grant timestamp $SGT = \langle \mathcal{H}\left(\text{AGA}\right), nb, \sigma_{SPT}\rangle$ as a confirmation of acceptance of the grant. Note that hash function $\mathcal{H}\left(\cdot\right)$ is used to hide the content of the authorization grant from the authorization service, that is a knowledge that must be maintained secret between the service and the client. Moreover, note that the authenticated log service does not guarantee to insert the grant immediately, but guarantees to insert it within the *nb* time of the *SGT* data. This is a strategy of many architectures based on authenticated data structures to guarantee scalability [7], [8], [9].

(3) The client requests an access token to the authorization service through the *Token* routine by including the authorization grant attestation *AGA*, that is used by the service to compute the access token $AT = \langle \mathcal{H}\left(\text{AGA}\right), AP, \sigma_{AT}\rangle$, where *AP* is the authorization policy included in *AGA* and σ_{AT} is the signature of $\langle \mathcal{H}\left(\text{AGA}\right), AP\rangle$ computed by using the secret key of the authorization service. Note that the hash function $\mathcal{H}\left(\cdot\right)$ computed over *AGA* must be the same used to compute *SGT*, to allow things to validate that *AT* and *SGT* used in the next phase refer to the same *AGA*.

(4) The client accesses the thing by sending a valid access token *AT*, the signed grant timestamp *SGT* and the due payload *data* of the request. The thing accepts the request if and only if the values corresponding to $\mathcal{H}\left(\text{AGA}\right)$ included in *AT* and *SGT* are the same. Then, the thing validates *AT* by verifying the authorization policy, as in the reference protocol (see Section III-B) and the signature included in the access token by using the known public key of the authorization service. Finally, the thing also validates the signature of the signed grant timestamp *SGT* by using the known public key of the authenticated log.

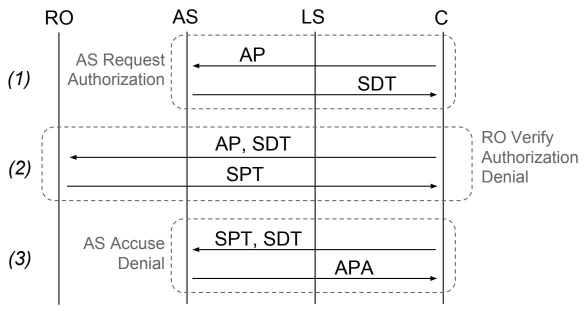

Figure 4: Verification of illegitimate authorization denial

D. Verification protocols

We propose two verification protocols: to verify that no authorization has been granted to unauthorized clients, and to verify that all authorizations have been granted to authorized clients.

The first protocol allows a resource owner to verify if the authorization service has issued authorizations to unauthorized clients. To this aim, a resource owner can act as a monitor of the authenticated log service, downloading the whole authenticated data structure periodically. This procedure is similar to that of monitors for the Certificate Transparency system and could be easily deployed by using existing software implementations. An alternative would be to use the more efficient *QueryGrant* routine offered by the authenticated log service. In this case, the authenticated log service must maintain authenticated data structures that are efficient with regard to the considered queries [14]. As an example, a resource owner might want to monitor all the authorizations released within a certain time frame for a specific thing or for a specific client.

The second protocol allows clients to verify if the authorization service has illegitimately refused to issue authorizations despite existing authorization policies released by the resource owner. This verification procedure includes three phases, that we describe by referring to Figure 4.

(1) We assume that the authorization service denies access to a thing by a client that invoked the *Authorization* routine, and thus it answers by returning a signed denial timestamp $SDT = \langle \mathcal{H}(AP), nb, \sigma_{SDT} \rangle$. Please note that the *nb* value represents the time instant at which the authorization service refuses the authorization request.

(2) To verify that the authorization denial was legitimate, the clients uses the *VerifyDenial* routine offered by the resource owner by sending *AP* and *SDT* previously sent and obtained by the previous invokation of the *Authorization* routine. The resource owner verifies if the *SDT* is correctly bound to the *AP* by re-computing the hash value $\mathcal{H}(AP)$ on its side, and verifies if the *AP* complies to the authorization policies previously established by using the *UpdatePolicy* routine (see Section IV-C). If an authorization policy exists that can assess that the client had legitimate access to the thing, then the resource owner returns the signed policy timestamp *SPT* that the authorization service returned as an answer to the *UpdatePolicy* routine. Note that, for *SPT* to be considered valid, the release timestamp *nb* included within *SPT* must be greater than the one included in *SDT*.

(3) The client uses the *SPT* received by the resource owner, together with the *SDT* received in phase *(1)*, to send a request to the *AccuseDenial* primitive offered by the authorization service. The protocol allows the authorization service to defend itself against the accusation by returning an authorization policy attestation *APA* that is more recent that the *SPT* and that can assess that the denial was legitimate. Note that this is only possible if the resource owner actually updated its policies and misbehaved in phase *(2)* by not operating correctly. If such an *APA* does not exist, then the authorization service behaved incorrectly.

E. Comparison with related approaches

We present a comparison of the proposed system with regard to alternative approaches proposed in the literature by referring to Table III. We consider four proposals that have strong security guarantees, based on cryptographic primitives, and consider similar threat models where the authorization service is considered malicious: *transparency* refers to the system proposed in this paper; *Bitcoin log* refers to the Catena protocol illustrated in [11], proposed in the two architectural variants based on a device that runs a blockchain *full node* or the Simplified Payment Verification protocol (*SPV*); the *Ethereum smart-contract* approach refers to the systems proposed in [19], [26]. For a better overview of the protocols please refer to Section II.

We distinguish three types of characteristics: *security* identifies the peculiar traits of each protocol in terms of security *assumptions* and *guarantees*; *thing requirements* identifies the characteristics of the IoT environments, distinguished in hardware capabilities of the IoT devices (e.g., capability of *execution* of certain algorithms, and *storage* size) and characteristics of the *architecture* where things are operated (e.g., availability of an Internet connection); *protocols timings* identify the performance of the main protocols operations, that are *grant authorization* by an authorization service (the time required by a party to release an authorization that can be considered valid by a thing), *audit delay* (the maximum delay after which a party can operate a verification at the authorization service) and *request verification* (the time required by a thing to validate a request by a client).

The *security* characteristics allow to understand if a solution is able to satisfy the requirements of the considered scenario. As an example, we highlight that approaches based on distributed ledgers do not guarantee only protection against a dishonest authorization service but, due to the native characteristics of the distributed ledgers protocols, also availability of the service. Moreover, protocols based on

		Transparency (*our proposal*)	**Bitcoin log [11]**		**Ethereum smart-contract [19], [26]**
			Full Node	**SPV**	
Security	*Guarantees*	Accountability	Accountability, Availability		Accountability, Availability
	Assumptions	Availability	Consensus	Consensus, SPV proxy	Consensus, Management proxy
Things requirements	*Execution capabilities*	Hash functions, Signature verification	Bitcoin protocols, Signature verification	SPV protocols, Signature verification	Management protocols, Signature verification
	Architecture	Trusted configuration	Bitcoin network	SPV proxy	Management proxy, Trusted configuration
	Storage capabilities	Public keys [*min* 2×32 *bytes*]	Bitcoin blockchain [*hundreds of GBs*]	Bitcoin headers [*tens of MBs*]	Public keys [*min 1 X 32 bytes*]
Protocols timings [*estimated*]	*Grant authorization*	Signatures sign [*tens of ms*]	Signatures sign [*tens of ms*]		Mining $+$ next $10 - 50$ blocks [*about* $2.5 - 12.5$ *minutes*]
	Audit wait time	Maximum Merge Delay (MMD) [*hours*]	Mining $+$ next 6 blocks [*about* 1 *hour*]		Mining $+$ next $10 - 50$ blocks [*about* $2.5 - 12.5$ *minutes*]
	Request verification	Signature verification [*ms or tens of ms*]	Blocks verification [*tens of ms*]	SPV verification [*tens of ms*]	Blocks verification [*tens of ms*]

Table III: Comparison with other cryptographic approaches for accountable IoT authorizations

smart-contracts offers even stronger securities by preventing an authorization service from behaving maliciously (we identify this trait by using the *enforcement* guarantee instead of *accountability* guarantee).

The *thing requirements* allow to identify if each solution can be deployed in a certain scenario. As an example, we note that both blockchain solutions might require a lot of storage at the thing side to maintain the blockchain. This disadvantage could be reduced by using special-purpose blockchain networks of smaller sizes separated from the public ones, or by using the SPV protocol, that however introduce stronger security assumptions (that is, trust in intermediate proxy nodes). The proposed approach is very lightweight as it requires only storage of a few public keys and it does not require any Internet connectivity for the things. As disadvantages, the proposed system requires a trusted setup of these keys (as we discussed in Section IV-B), and the resource owner to be available to verify illegitimate authorization denials.

Finally, the *protocol timings* highlights that the proposed approach might require longer wait times to be able to operate an audit operation after an authorization grant (the *audit delay time* field), that is the Maximum Merge Delay (MMD) value used by the authenticated log service to merge the released authorization grants within its authenticated data structures. However, please note that while the wait time in our approach is deterministic and refers to the release of the authorization grant, in bitcoin-based logs it depends on when miners are able to insert the due transactions in the blockchain. Finally, note that releasing a verified authorization in the smart-contract approaches requires significant additional times because they require that the transactions are inserted in the blockchain before releasing the authorization.

V. Conclusions

We proposed a system that allows to audit authorization procedures operated by intermediate services in smart IoT environments, characterized by many stakeholders, and by devices with low resources placed in constrained environments. The security of the system is based on cryptographic signatures that allow to detect and expose misbehaviors of intermediate services to the public. The proposed design is compliant with ongoing Web standardization efforts in the context of delegated authorization protocols, and contribute to the body of PKI-related transparency literature by allowing authorization material being logged without leaking secrets to the log. In light of the proposed design and analyses, future work will be devoted to implement a proof-of-concept, to be exercised and evaluated in a smart city deployment.

Acknowledgments

The authors acknowledge the financial support of the POR-FESR project n. E91F18000260009 "SmartChain: interoperability and security of the industrial supply chains".

References

[1] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito, “Stack4things: integrating iot with openstack in a smart city context,” in *Proc. IEEE Int’l Smart Computing Workshops*, 2014.

[2] N. Fotiou, A. Machas, G. C. Polyzos, and G. Xylomenos, “Access control as a service for the cloud,” *Journal of Internet Services and Applications*, vol. 6, no. 1, p. 11, 2015.

[3] T. Ahmad, U. Morelli, S. Ranise, and N. Zannone, “A Lazy Approach to Access Control as a Service (ACaaS) for IoT: An AWS Case Study,” in *Proc. 23rd ACM Symp. Access Control Models and Technologies*, 2018.

[4] Amazon Web Services, “Aws iot,” https://aws.amazon.com/iot/, Amazon, Dec. 2018.

[5] Google Cloud Platform, “Google cloud iot,” https://cloud.google.com/iot/docs/, Google, Dec. 2018.

[6] Google Cloud , “Google access transparency,” https://cloud.google.com/access-transparency/, Google, Dec. 2018.

[7] B. Laurie, A. Langley, and E. Kasper, “RFC6962: Certificate Transparency,” RFC, Jun. 2013.

[8] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, “Enabling security in cloud storage slas with cloudproof.” in *Proc. 2011 USENIX Annual Technical Conf.*

[9] A. Andreoli, L. Ferretti, M. Marchetti, and M. Colajanni, “Enforcing correct behavior without trust in cloud key-value databases,” in *Proc. IEEE Int’l Conf. Cyber Security and Cloud Computing*, 2015.

[10] L. Ferretti, M. Marchetti, and M. Colajanni, “Verifiable delegated authorization for user-centric architectures and an oauth2 implementation,” in *Proc. IEEE 41st Conf. Computer Software and Applications*, 2017.

[11] A. Tomescu and S. Devadas, “Catena: Efficient non-equivocation via bitcoin,” in *Proc. 38th IEEE Symp. Security and Privacy*, 2017.

[12] N. Tapas, G. Merlino, and F. Longo, “Blockchain-based iot-cloud authorization and delegation,” in *Proc. IEEE Int’l Conf. Smart Computing*, 2018.

[13] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig, “Authentication and authorization for constrained environments (ace) using the oauth 2.0 framework (ace-oauth),” https://www.ietf.org/id/draft-ietf-ace-oscore-profile-02.txt, Internet-draft, Dec. Dec. 2018.

[14] A. Miller, M. Hicks, J. Katz, and E. Shi, “Authenticated data structures, generically,” in *Proc. 41st ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, 2014.

[15] L. Ferretti, M. Marchetti, M. Andreolini, and M. Colajanni, “A symmetric cryptographic scheme for data integrity verification in cloud databases,” *Information Sciences*, vol. 422, pp. 497 – 515, 2018.

[16] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” in *IEEE Symp. Security and Privacy*, 2013.

[17] V. Karande, E. Bauman, Z. Lin, and L. Khan, “SGX-log: Securing system logs with sgx,” in *Proc. ACM Asia Conf. Computer and Communications Security*, 2017.

[18] A. Ben-David, N. Nisan, and B. Pinkas, “FairplayMP: a system for secure multi-party computation,” in *Proc. 15th ACM Conf. Computer and Communications Security*, 2008.

[19] O. Novo, “Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT,” *IEEE Internet of Things Journal*, vol. 5, no. 2, Apr. 2018.

[20] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, “Inferring fine-grained control flow inside sgx enclaves with branch shadowing,” in *Proc. 26th USENIX Security Symp.*, 2017.

[21] J. Bonneau, “Ethiks: Using ethereum to audit a coniks key transparency log,” in *Proc. Int’l Conf. Financial Cryptography and Data Security*. Springer, 2016.

[22] G. Alessandro, C. E. Rottondi, G. Verticale *et al.*, “Blast: Blockchain-assisted key transparency for device authentication,” in *Proc. IEEE 4th Int’l Forum on Research and Technologies for Society and Industry (RTSI)*, 2018.

[23] T. Ruffing, A. Kate, and D. Schröder, “Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins,” in *Proc. 22nd ACM SIGSAC Conf. Computer and Communications Security*, 2015.

[24] M. M. Hossain, R. Hasan, and S. Zawoad, “Probe-iot: A public digital ledger based forensic investigation framework for iot.” in *Proc. INFOCOM Workshops*, 2018.

[25] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, “CONIKS: Bringing Key Transparency to End Users.” in *Proc. USENIX Security Symp.*, 2015.

[26] D. D. F. Maesa, P. Mori, and L. Ricci, “A blockchain based approach for the definition of auditable access control systems,” *Computers & Security*, vol. 84, 2019.

[27] IETF, “RFC 6749: The OAuth 2.0 Authorization Framework,” Oct. 2012.

[28] ——, “RFC 6819: OAuth 2.0 Threat Model and Security Considerations,” Jan. 2013.

[29] Google, “Trillian: General Transparency,” https://github.com/google/trillian/tree/67395f8e7cf7f94ef80ac1846ec10c49a4d6550f, Nov. 2018.