A Privacy-Aware Architecture to Share Device-to-Device Contextual Information

Juan Luis Herrera*, Javier Berrocal*, Juan M. Murillo*, Hsiao-Yuan Chen† and Christine Julien†
*University of Extremadura, Spain. [jlherrerag, jberolm, juanmamu]@unex.es
†University of Texas, Austin, United States. [littlecircle0730, c.julien]@utexas.edu

***Abstract*—Smartphones have become the perfect companion devices. They have myriad sensors for gathering the context of their owners in order to adapt the behaviour of different applications to the device's situation. This information can also be of great help in enabling the development of social applications that, otherwise, would require a costly and intractable deployment of sensors. Mobile Crowd Sensing systems highly reduce this cost, but realizing this vision using traditional centralized networking primitives requires a constant stream of the sensed data to the cloud in order to store and process it, which in turn leads to the individuals about whom the data is sensed losing control over the privacy of the data. In this paper, we propose an architecture for a device-to-device Mobile Crowd Sensing system and we deepen on a new privacy model that allows users to define access control policies based on their context and the consumer's context.**

***Index Terms*—Privacy, Mobile Crowd Sensing, Device-to-Device, Pervasive computing, mobile applications.**

I. Introduction

Context-aware applications are increasing in their importance on mobile phones because of their adaptability and ease of use. These applications behave differently depending on the user's context, thus adapting themselves to provide services that can be most useful to their users. To do so, these applications feed on contextual information, which includes data such as the location or identity of the user, time-related data or the activity being performed [1]. Context-aware applications can even infer higher-level information from combinations of lower-level data in order to offer the most relevant information or services to a specific user.

These applications are nowhere more evident or important than in new smart city infrastructures [2]. In these environments, context-aware applications must also be social, that is, the contextual information from different users must be shared to enable collaboration in the smart city. One approach would be to deploy infrastructure-owned sensors to collect information from the inhabitants of the smart city. However, such an approach requires the deployment and maintenance of many sensors—an approach that is costly and intractable. Another possibility is to leverage the sensors that the city's inhabitants carry with them in any case—sensors connected to their wearable devices and smartphones. We refer to these devices throughout as *companion devices*. These devices are already imbued with myriad sensors, from accelerometers to GPS receivers. Existing applications use these personal devices to support individual user's needs, but it is also possible to offer them and the data they collect to be consumed by other nearby users and their devices [3]. For instance, the user's location, which is normally used by the smartphone to provide the user services about where they parked or information about the bar, shop, or restaurant where the user currently is, can also be shared so that other users can know if a bar is crowded or to let the city council to know which are busiest streets so that public policies can be better planned, for instance to make pedestrian routes more accessible for disabled people.

While this device-to-device cooperation is technically feasible, there remain unsolved challenges. First, there is a need for a technological architecture that allows companion devices to offer the information they sense to other devices, providing a gateway to access the surrounding context information. What's more, the architecture must provide mechanisms to allow different devices to communicate opportunistically. Second, these virtual context profiles potentially contain particularly sensitive and private information, such as the location or trajectory of the user or the activities they perform, that could be used for malicious purposes. Therefore, it is not enough to have an architecture that allows devices to offer their contextual information to others, but the architecture should also ensure the privacy of the user in a way that is tunable by the user directly.

An initial approach could be to constantly upload the information sensed by user's devices to a cloud service that is able to create virtual profiles and distribute context information, but this would imply a constant data flow, would require a constant Internet connection from every device, and would result in an increased response time due to the latency added by the communication with the cloud [4]. In addition, such an approach requires all of the communicating parties to place their trust in a shared third party service provider. We explore an alternative that mitigates these concerns by storing the virtual context profiles in the users' companion devices, and providing services for allowing other devices to consume that information by means of different sharing mechanisms (device-to-device, opportunistic networks, etc.). An additional access control layer is required to empower users to explicitly control what and how other devices and users are authorized to access the data. In the end, a full system will likely entail a hybrid of both approaches, one that mixes cloud interactions with opportunistic ones. In this paper, however, we isolate the latter and focus only on enabling privacy-preserving device-to-device context sharing.

In this paper, we present PADEC (Privacy-Aware DEvice

Communication), an architecture that allows companion devices to create and provide context information so that other devices can query that data to enable their applications to adapt their behaviour to their users' needs and surroundings. This architecture includes elements to store this data in mobile devices, offer this information to neighboring devices, communicate with other devices through different communications mechanisms, and control the access to this data by using user-set policies through a novel system based on contextual and dynamic keys, keyholes and locks.

The remainder of this paper is structured as follows: Section II presents the motivation behind the PADEC architecture. Section III shows the high-level architecture of PADEC. Section IV provides the details of an example communication abstraction, while Section V describes our proposed access control privacy layer in detail. Finally, Section VII concludes the paper.

II. Motivation and Background

Mobile Crowd Sensing (MCS) refers to the reliance on personal device resources to support gathering information about humans and their surroundings [5]. This information is commonly used to create a collective intelligence to provide services to the users based on the collected information.

In the following subsection, we present a running example of an MCS and our motivations behind a privacy-aware framework to provide personalized MCS services.

A. Running example

Imagine a smart city that provides an application to be used by both residents and tourists. The app provides information about restaurants and bars by collecting and sharing crowd-sensed information. The app collects and stores users' context information, such as locations, activities (e.g., eating, walking, working), presence of others (e.g., friends and family), etc. The app can use this personal information to provide recommendations, nudges, or other services to the user based on the user's own contextual history [6].

Other users could benefit from the availability of this crowd-sensed contextual information. For instance, a tourist visiting the city could use the app to query nearby residents for a restaurant recommendation. Also, the app can be used to query other users' devices (and their stored context information) to find what is popular nearby, potentially with a day-specific special or happy hour. Additional contextual information (e.g., the tourist's food preferences, the size of the tourist's dinner party, or where the tourist ate lunch) could also be added to the query to improve the results. Residents can also benefit from being able to query the contextual information sensed by other residents (or even visitors). For instance, individuals could query for information about where their friends or family members might prefer to get drinks on a Friday evening so they can make a recommendation for a group outing.

B. Motivation

The presented running example requires a large amount of contextual information from many different devices. To get and process this information, one approach can be to offload all of the context information to the cloud, an approach akin to that of Google Cloud's Places [7]. For instance, MCS platforms such as PartcipAct [8] and Vita [9] gather the users' contextual information using their smartphones and store it on a server so that complex algorithms can be executed. However, this style of approach requires users' devices to constantly stream information to a cloud server and enables the third party owner of the cloud server to know and track the situation of each individual.

When the crowd-sensed data is offloaded, the owner of the data loses control over it. The third party becomes responsible for protecting and sharing the data. In these situations, users tend to eschew sharing their data with others. However, research has shown that, while users might not be willing to share their spatiotemporal data publicly, they are more prone to share the information with others who are physically nearby [10], [11].

An alternative approach is to *on-load* the contextual information to the user's device, keeping all stored mobile crowd-sensed information persistent only on the device belonging to the user. [6], [12], [13]. Once the information is on-loaded to a user's companion device, it can be shared opportunistically with other users nearby. This approach comes with its own set of challenges. First, because companion devices like smartphones may be resource constrained in terms of battery, computation, and storage, the device needs an intelligent mechanism to constantly sense contextual information and maintain an accurate yet lightweight representation of that context. Second, each companion device needs to offer some kind of interface to other devices so they can consume the stored information on demand. Third, both devices need a common communication medium.Finally, the shared information is still sensitive and private, and the users should retain control over what other nearby devices have access to.

The first challenge has been addressed by Paco [6], an on-loading context storage mechanism that uses a companion device's location sensor and other on-board context sensors to create a spatiotemporal context database stored entirely on the user's companion device. Paco leverages the spatiotemporal tags associated with contextual information to determine how novel a sensed data item is before storing it; spatiotemporally reducing the memory footprint of the sensed context. Paco exposes a query interface that allows applications to access the contextual data at varying levels of abstraction, opening the possibility of defining access rules for snapshots of a user's contextual data.

For the second challenge, i.e., offering interfaces to make the contextual data available to other devices, APIGEND [14] allows for easy code generation and deployment of Application Programming Interfaces (APIs) in companion devices. These APIs are composed of a series of endpoints that can be exposed and called from other devices. These endpoints wrap some business logic, which, for this work, will be the ability to access some context data and return it to the device that called the endpoint.

In this paper, we tackle the remaining two challenges: unifying the communication medium and providing context-sensitive approaches to controlling access to the stored contextual data. For defining the communication mechanism, there are a variety of protocols and platforms that can be use for device-to-device communication. Classic request-response protocols such as HTTP are not ideal in this situation due to the highly dynamic an unpredictable nature of the opportunistic device-to-device environment. Simply put, request-response protocols are host-based, and thus, one must know the address (normally the IP address) of a device to perform a request on it. This is not so simple when the device is mobile, since these devices often have private IPs that are unreachable from the outside.

There are a variety of opportunistic or publish-subscribe protocols that can be used to solve these problems. In this paper we focus on publish-subscribe protocols, but different techniques can be used. This communication model is promising and privacy-preserving, since it is possible to use a secure but distributed intermediate broker to mediate requests and responses [15]. To be clear, this is different than a fully centralized approach because the collaborating parties are interacting directly with one another and can secure their end-to-end communications. In the centralized approaches, information is released in the clear to the third party to allow the third party to perform data processing and analysis.

Within publish-subscribe communication models, Google's Firebase [16] is one popular approach. Another possible approach is to use the open MQTT protocol [17], but it requires setting up a dedicated broker rather than relying on the central broker provided in Firebase. On the other hand, this approach can reduce the reliance on the third party to mediate communication. In this work, we explore allowing applications to optionally use either of these publish-subscribe approaches.

The meat of this paper focuses on the final challenge: addressing and protecting the privacy of users' shared contextual information. The contextual information that must be exposed to enable the application uses described above is potentially highly sensitive. This information can be used by malicious parties, for instance, to spy on the user [18]. This necessitates a privacy layer that provides user-tunable access control for the contextual information. A wide variety of access control models exist, one of the most interesting being the NIST-standard Role-Based Access Control (RBAC) [19]. Other works have extended RBAC to consider context, such as Team-Based Access Control (TMAC) [20] or Dynamic Sharing and Privacy-Aware RBAC (DySP-RBAC) [21]. However, these models were designed for collaborative working environments. Smart cities with decentralized device-to-device communications demand new techniques that allow access control to be dynamic and to consider the identity (and context) of both the provider and the consumer of information. In this work, we present a new access control model designed for these environments: Dynamic Context and Identity-Based Access Control (DCIBAC).

III. Architecture Overview

In this section, we provide a high-level overview of the complete PADEC architecture, before drilling into the details of each of its constituent components. Figure 1 provides a pictorial representation.

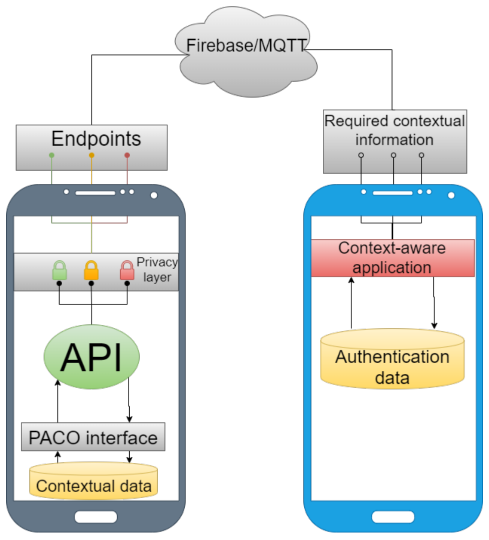

Figure 1. PADEC high-level architecture

In PADEC, a user's companion device can collect, store, and share contextual data. In addition, a context-aware application executing on a companion device can request contextual data from other nearby devices. In practice, devices will commonly perform both tasks. For simplicity, Figure 1 distinguishes these two roles; the device on the left is acting as a context collector and provider, while the device on the right consumes that context information via a remote connection. In PADEC, a device can only consume information if it also provides endpoints; this "sharing economy" style is PADEC's approach to incentivizing device participation. To describe the PADEC architecture, we walk through each element in the figure, starting at the bottom left of the figure and working up, to the right, and then back down.

The first layer focuses on storing the contextual information gathered by both internal and external sensors. In PADEC, we implement this layer using the Paco contextual data storage framework [6]. In Paco the gathered information is indexed within a database based a spatiotemporal context stamp that includes the coordinates and time at which the context information was collected. This spatiotemporal information is associated with the relevant semantic information (e.g., reviews, ratings, etc.) or data sensed by others sensors (e.g., temperature, noise, etc.). The key contribution of Paco is reducing the size and complexity of the stored contextual information so that it can be stored and queried entirely on the companion device. For this purpose, Paco also provides an API for issuing spatiotemporal queries for the stored information.

The second layer allows other devices to consume the information stored by Paco (API in Figure 1). This layer allows the deployment of OpenAPI-based APIs, to ensure that the endpoints exposed to applications are properly defined and documented so that remote applications can easily access them. These endpoints serve as the gateway between external devices and the local Paco data store. They serialize all of the exchanged information and they provide a first line of defense protecting the private spatiotemporal information stored within the Paco data store. To reduce the effort required to implement these APIs, we rely on the APIGEND tool [14]. This tool takes as input the API specification and generates the skeleton of an API for Android mobile devices and microcontrollers, so that developers only have to implement the business logic to invoke the query methods exposed in Paco's interface. For the time being, we simply expose the generic Paco interface, which allows queries asking the *probability of knowledge* of the Paco data store about a region, to retrieve the context data items contributing to that knowledge, or to trace a trajectory of the data owner through space and time. In the future, we could use APIGEND to expose more application-specific endpoints to further simplify the use of PADEC. For instance the endpoints used for a smart traffic application might focus queries on the density of cars on roadways, while endpoints for a smart tourism application might expose thematic walking tours.

Without additional protections, the deployed endpoints can be consumed by any PADEC-enabled connected device, which can be a problem for the privacy of the stored data. The next layer in the device on the left of Figure 1 implements a dynamic and multi-dimensional access control system for every endpoint (labeled "Privacy layer" in the figure). This layer enables the definition of access control policies that restricts a remote party's access to the context information. Access can be allowed at various levels of abstraction based on whether the requester and the provider meet concrete user-defined rules. These rules can define different contextual situations that have to be satisfied for the information to be shared. For instance, for our smart city example, a tourist may only be able to access stored information about the best restaurants if he is currently in the city. Furthermore, the level of detail of the information provided may also be different depending on different contextual situations. In our example, a tourist might be able to uncover a part of town that is likely to have many good restaurants rather than discovering exactly which restaurants the context provider has frequented.

The very top of Figure 1 shows the layer that manages the communication between the devices. In the current implementation of PADEC, we support two different publish-subscribe communication protocols (Firebase and MQTT) that can be employed individually or in combination. As future work, we plan to support other technologies such as direct device-to-device communication or opportunistic networks. By providing multiple options under the same umbrella, PADEC makes it possible for different devices in different situations to consume the provided information in the way most suited for the situation. The required infrastructure for implementing these protocols is also generated using the APIGEND tool.

In the following sections we will focus on the addressed challenges: how the communication medium is unified and the contextual access control policies are defined.

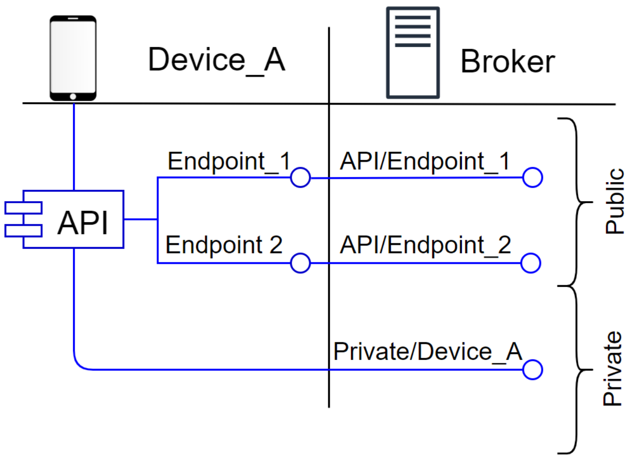

Figure 2. Communication Channel structure.

IV. Communication model

The communication layer at the top of Figure 1 mediates information sharing among devices to allow the exposed APIs to be consumed by other devices. The main responsibilities of the communication layer are to support different communication protocols for invoking the endpoints and to provide a first line of defense protecting the exchanged data.

In the current implementation of PADEC, the communication layer unifies diverse publish/subscribe protocols. Concretely, PADEC currently supports MQTT and Firebase Cloud Messaging. Communication through such protocols is commonly based on a central element, i.e., a "broker", that manages the entire communication. Devices can interact with the broker by establishing *topics* that create communication channels. With entities that *publish* information to a given channel, and *subscribers* receiving that information.

These roles are used in PADEC to request or send information by publishing in a channel and to receive the published information by being subscribed to that channel. Different channels are used to simulate the request-response communication pattern. To organize the communication among the myriad different entities and to enable strict access control, PADEC defines a topic structure that is automatically initialized when a device connects to the broker or when the device creates a new API endpoint or application component. This structure is shown in Figure 2.

PADEC creates two types of topics: private and public. The first time a device registers with the broker, a private channel is created, dedicated to communication destined to that device. Any device can publish in this channel, but only the registered device (the "owner" of the topic) is subscribed to it and is the only one that is notified of any published information. This private channel can be useful for multiple purposes. For instance, an endpoint provider can use its private channel to receive the private messages required to exchange the information used to determine access control of a requesting device. Likewise, a device invoking an endpoint can use its private channel to get the results of the invoked endpoint. More generally, when any device knows the ID of another device and wants to establish a device-to-device communication, it can use this private channel for the communication exchange. For devices not knowing the private channels, public channels for each provided endpoint are also created.

PADEC automatically creates public channels when a new API is deployed; the system creates a new topic for each provided endpoint. These channels are open, and any device can publish a message to them. In general, a message in one of these channels indicates a desire to invoke the particular endpoint. Only the device providing the endpoint is able to subscribe to and consume messages published to the topic. This provides one level of privacy for the devices invoking the endpoints; only devices hosting the target API will be able to see the request, preventing other devices from reading the content of the device's endpoint invocation. At the same time, this architecture provides a form of *spatial decoupling* in which the device wanting to invoke the endpoint does not need to know the details of the hosting device, like its address or location. These channels only support the invocation of endpoints, any subsequent communication for exchanging the intermediate messages or the requested data is maintained through private topics with the aim of not leaking private data. In addition to the decoupling that this provides, it also enables PADEC to create more sophisticated endpoints out of the aggregation of base endpoints to provide higher level services. For example, to ascertain whether a specific bar or area of the city is crowded, a requesting device can invoke a public endpoint offering information about the target location. Any device within this area willing to share crowd information can provide the endpoint and be subscribed to the channel. Upon receiving a notification of the published invocation, all of the devices providing the endpoint will reply, and the requesting device can aggregate the information to acquire a more robust measure of crowd density.

In PADEC, we implement this same functionality with both MQTT and Firebase Cloud Messaging. The former can be deployed in networks without any external connectivity to the Internet, while the latter relies on access to the Firebase cloud servers. Currently, the broker deployment is out of the scope of the paper. Nevertheless, for MQTT a hierarchical model can be used in order to improve the scalability. In the next section, we describe the PADEC privacy model defined to dynamically control the access to the information.

V. Privacy model

For PADEC's privacy layer, we propose a new access control model named Dynamic Context and Identity-Based Access Control (DCIBAC) that is designed with decentralized smart environments in mind, while inheriting some elements from access control models such as RBAC [19] or DySP-RBAC [21]. As its name implies, DCIBAC uses information about the identities and context of coordinating parties to determine whether access is granted. In PADEC, this means that information is used about both the endpoint provider and the requester to determine whether the invocation is allowed and how the invocation result can be presented, e.g., in terms of abstraction or granularity of the data returned. In contrast to approaches for access control of data shared in a cloud-based system, because PADEC performs access control locally, it can compare a requester's contextual information to the data owner's very private contextual information without requiring the provider to release the personal data to the third party that completes the access control decision.

In DCIBAC, pieces of information that need to be protected are called *objects*, and the agents that either own or try to access objects are *entities*. DCIBAC is based on putting *locks* over objects, so only authorized entities can access them. The entities who try to access objects are named *requestors*. The authority controlling if some specific information can be provided to the requestors is distributed and delegated to the providers. In PADEC, the data is the contextual data kept in a Paco data store, and the entities are the devices requesting remote access to information stored in Paco.

DCIBAC's *locks* are implemented via user-defined rules that can impose conditions, mainly on the context or identity of the owner of the object or on those of the requester. For instance, a user may allow access to information about the specific restaurant they are in at the moment to be accessed only by their near family. A user may allow their boss to access their position but only during working hours. These conditions can be combined using logical *or*, *and* and *not*, thus allowing for complex and rich conditions. For instance, it is possible to allow friends to access information about the restaurant the user is in only during Friday nights and there is not a scheduled appointment in the user's calendar for Friday night. DCIBAC also allows locks to include conditions on the intended use of the information, for instance the particular application consuming the information. A user's instantaneous location data may not normally be released to a stranger's device, but it might be released to smart traffic app if there is a nearby traffic accident.

To enable devices to request access to a protected object, DCIBAC uses, as far as we know, a novel concept called *keyhole*. Before requesting access to the object, the requester must first get the keyhole of the object. This keyhole identifies the information that the requester must send to open the lock request access. For instance, if a lock allows access to friends as long as they are nearby and their purpose is not marketing, its keyhole would be the position of the requester, their identity, and their purpose. This interaction does not reveal the exact conditions of the lock, only the type of information on which the decision is based. This message sent from the requester with the needed information is termed a *key*.

Figure 3 summarizes the workflow that has to be followed in order to validate an endpoint's defined access control policies.

1) The requesting device invokes an endpoint publishing a message in the associated topic.
2) The provider checks whether the endpoint is available in

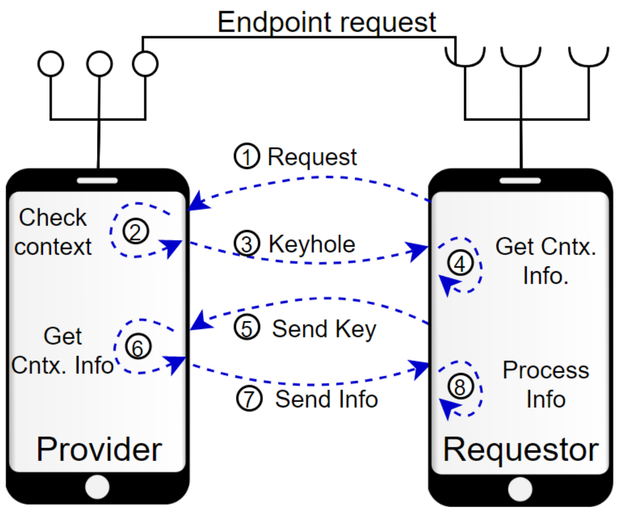

Figure 3. Communication workflow.

the provider's current context. For instance, a user may disallow access to an endpoint that shares their recently taken photographs while on an outing with their family.
3) If the endpoint can be consumed, the provider sends to the requester the keyhole that specifies any needed contextual information.
4) The requesting device acquires the contextual information required by the keyhole using the device's own local data store, on board sensors, or by invoking a remote third party service.
5) The requester packages the obtained information as a key that can be used to attempt to open the lock protecting the endpoint.
6) The endpoint provider executes the lock to check that the provided information is sufficient to grant the requester access to the protected endpoint. The provider invokes the endpoint if the lock is successfully opened.
7) After invoking the endpoint, the provider publishes a notification on the requester's private channel with the information requested, if access is allowed, or with a rejection notice if not.
8) Finally, the requesting device processes the obtained information.

Only the first message is published on a public channel, specifically the public channel associated with the target endpoint. The subsequent messages use the devices' private channels. This process allows us to create and support a dynamic and multidimensional privacy model able to provide or reject the consumption of information depending on the context of both the provider and the requester.

The final concept in DCIBAC is the definition of *access levels* of a lock. In Paco, each request to retrieve information from the Paco data store is made with a certain profile. This profile determines the degree of spatiotemporal granularity with which the request is allowed to view the contextual data store and the maximum number of requests an individual requester is allowed to make. These restrictions further protect the privacy of the owner's spatiotemporal information.

These profiles are mapped to access levels in DCIBAC locks. A single lock has by default a single access level, but it can also have many levels. These access levels are ordered, with the first being the level that releases data from the Paco data store with the least precision. Each level can have its own conditions, meaning a lock with multiple levels may be associated with each lock. Each level has a keyhole associated.

Since each access level can use different contextual information, it is possible that a single lock is associated with multiple keyholes. When a lock exposes multiple keyholes, it exposes some information about how its access decisions are structured. While this can aid requesters in understanding the conditions of access it also potentially exposes decision logic of the endpoint owner. Navigating this trade-off is left flexible for the endpoint owners to navigate.

When a user invokes an endpoint, all the keyholes associated with the endpoint's lock are sent to the requester together with the precision of the information they will get. The requester can analyze this precision in order to create the key for the keyhole with the required precision. This also allows the requester to leak, with the key, the minimal contextual information to get the needed information.

As an example, consider a lock with two access levels: level 1 that provides abstract, granular information for unknown nearby devices and level 2 that provides more detailed information for close family. Two different keyholes would be created, one per access level. A keyhole for level 1 (position) and another one for level 2 (position and identity). While this makes sure the requester knows the information required to open each level of the lock (meaning the requester is able to protect its own private contextual information more), the requester can also discern some of the logic relating to the lock's access control policies.

Figure 4 shows this example. In this example, DCIBAC secures a single endpoint that gives access to the restaurants visited by the user. To control access to this endpoint, the user uses a lock with two access levels. Nearby strangers can request access information about the restaurants that the endpoint owner likes (level 1) and only nearby close family members can access information about how often the endpoint owner frequents particular restaurants. Two keyholes are created,such that the keyhole for level 1 of the lock requires the requester to provide position information, while the keyhole for level two requires both position and identity. In the example process depicted in Figure 4, the device of a tourist at the same bus stop as the endpoint owner will try to access good places to have a drink. To do so, the tourist's device first publishes the endpoint request to the endpoint's public topic (*1*) and receives a message generated by DCIBAC containing the keyholes of the lock (*2*). When this response is received, the tourist's phone will decide which level it needs to get access to, will collect its current position using its GPS sensor and place this information in a message that will serve as the tourist's key (*3*). The key is then be sent to the secure endpoint (*4*) via the

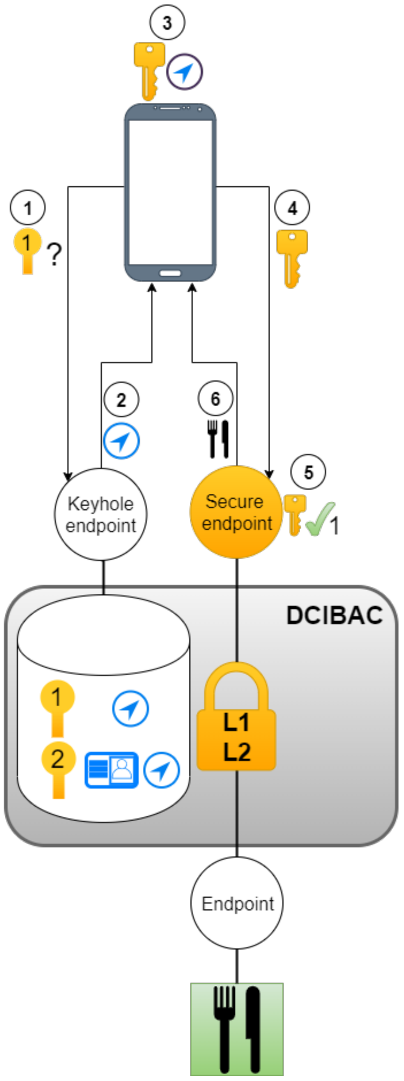

Figure 4. DCIBAC example usage.

endpoint's private topic. Once received, DCIBAC will check if the key is valid according to the lock and access level. Since the tourist is near the user and access level 1 was requested, access is granted (*5*). The secured endpoint will then invoke the necessary request on the endpoint's Paco data store and then send the information about the favourite restaurants of the user to the tourist's phone (*6*).

The permission to access an object in DCIBAC is dynamically checked and granted during execution time. This means that every time a requester wants to access some information, it must send its key, and there is no guarantee that a key will stay valid for any period of time, since the number of queries might be restricted depending on the access level.

VI. Discussion

Context-aware and MCS applications have proven useful during the last few years to distribute sensing and computation tasks in order to develop social and collaborative environments at an affordable way. For instance, [22] presents a framework to opportunistically offload computation in smart vehicles by accounting for context information. This framework uses the context of nearby vehicles, such as their speed or direction, to determine the best candidate to execute some tasks. [23] presents another framework that offloads data from social networking services to mobile devices based on their social context to maximize the quality of service of those services. These are also commercial examples of social applications that need contextual information from nearby users, such as Facebook Safety-Check [24].

A common approach to develop these platforms is to offload the data, or most of the data, to the cloud. This entails added resource consumption and a significant potential loss of privacy when the user's private contextual data resides in the Internet. These approaches also require a constant data flow, which can be problematic when the user has a limited mobile data plan. Such an approach also requires the user's device to be permanently connected to the Internet, which may not be possible in rural areas or even in urban canyons. This approach also relies on the third party infrastructure provider for protection and storage of data, and this third party provider is able to know and track every single device using the application. These challenges motivate PADEC's approach to onloading computing and data storage, which in turn enables device-to-device opportunistic provision of data access endpoints.

However, even when the data is stored locally, the user's privacy can be at risk, since other devices could potentially access that data if endpoints are not protected. To protect this information from malicious parties, storing and sharing contextual information is not enough, and an architecture that also accounts for privacy is required.

PADEC addresses these concerns head-on by providing an architecture that empowers users to store sensed contextual information in their own devices and to define dynamic and multidimensional access control policies that allow them to indicate what information can be accessed, by whom, and under what contextual circumstances.

This architecture also resolves other associated challenges, such as making it easier for developers to build mobile crowd sensing systems and applications that are especially complex due to device-to-device communication. PADEC's mechanisms for resolving these ancillary challenges comprise a series of modules that abstract developers from the implementation details on how to store information efficiently, how to expose the stored information to other users, and how to implement the communication workflow.

An open remaining challenge is to create methods to help and guide users in the definition of privacy policies, making the keyholes and locks that are fundamental to PADEC easier to use and more robust. Ideally, application users will be guided in establishing these rules in a clear, simple, and accurate way. Further, users should also be given a clear image of what information they are exposing or not depending on the established rules.

VII. Conclusions

In this paper, we presented PADEC to simplify the consumption of contextual information from other devices for context-aware and Mobile Crowd Sensing systems, which are important not only for the development of social applications, but also in paradigms such as the Internet of Things, Web of Things, or Human-in-the-Loop applications that require some devices to collaborate and adapt their behaviour depending on the context and the users' needs.

PADEC on-loads spatiotemporally tagged contextual information to the owner's device, and then enables this stored data to be queried by other devices. PADEC relies on the Paco framework for onloading the contextual data and APIGEND to define flexible API endpoints that expose Paco interfaces for remote queries. In this paper, we focused on providing user-tunable protection of the exposed endpoints by defining DCIBAC, a flexible access control mechanism that relies on the users' identities and context to determine whether access is allowed and what level of granularity of spatiotemporal information should be exposed. DCIBAC allows the definition of access control policies that protect the privacy of both coordinating parties.

Currently, we continue to work on evaluating the effectiveness of the PADEC architecture with respect to both the consumption of resources needed for the companion devices to on-load the computation and the communication load that the defined medium presents. On the other hand, we are also working on how to provide accurate information to users about the information they are exposing and the risks that this entails for their privacy. Finally, as future work, we will also define incentives and policies in order to foster information sharing.

Acknowledgment

This work was supported by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER, UE), by 4IE project (0045-4IE-4-P) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] R. Löwe, P. Mandl, and M. Weber, in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2012*, 2012, pp. 76–81.

[2] R. Kamberov, V. Santos, and C. Granell, "Toward social paradigms for mobile context-Aware computing in smart cities: Position paper," in *Iberian Conference on Information Systems and Technologies, CISTI*, vol. 2016-July. IEEE Computer Society, jul 2016.

[3] J. Guillen, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Software*, vol. 31, no. 2, pp. 48–53, 2014.

[4] J. Berrocal, J. García-Alonso, C. Vicente-Chicote, J. H. Núñez, T. Mikkonen, C. Canal, and J. M. Murillo, "Early analysis of resource consumption patterns in mobile applications," *Pervasive Mob. Comput.*, vol. 35, pp. 32–50, 2017. [Online]. Available: https://doi.org/10.1016/j.pmcj.2016.06.011

[5] H. Vahdat-Nejad, E. Asani, Z. Mahmoodian, and M. H. Mohseni, "Context-aware computing for mobile crowd sensing: A survey," *Future Generation Computer Systems*, vol. 99, pp. 321 – 332, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18329583

[6] N. Wendt and C. Julien, "Paco: A System-Level Abstraction for On-Loading Contextual Data to Mobile Devices," *IEEE Transactions on Mobile Computing*, vol. 17, no. 9, pp. 2127–2140, sep 2018.

[7] Google-Cloud, "Places — Google Maps Platform — Google Cloud," 2019. [Online]. Available: https://cloud.google.com/maps-platform/places/

[8] G. Cardone, A. Corradi, L. Foschini, and R. Ianniello, "Participact: A large-scale crowdsensing platform," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 1, pp. 21–32, 2015.

[9] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowdsensing-oriented mobile cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, pp. 148–165, 2013.

[10] Q. Jones, S. A. Grandhi, S. Karam, S. Whittaker, C. Zhou, and L. Terveen, "Geographic place and community information preferences," *Computer Supported Cooperative Work (CSCW)*, vol. 17, no. 2-3, pp. 137–167, 2008.

[11] E. de Matos, R. T. Tiburski, C. R. Moratelli, S. Johann Filho, L. A. Amaral, G. Ramachandran, B. Krishnamachari, and F. Hessel, "Context information sharing for the internet of things: A survey," *Computer Networks*, vol. 166, p. 106988, 2020.

[12] S. Han and M. Philipose, "The case for onloading continuous high-datarate perception to the phone," in *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, 2013.

[13] N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, and K. Papagiannaki, "When david helps goliath: the case for 3g onloading," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 85–90.

[14] S. Laso, M. Linaje, J. Garcia-alonso, J. M. Murillo, and J. Berrocal, "Deployment of APIs on Android Mobile Devices and Microcontrollers," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2020*, 2020, pp. 12–14.

[15] S. Arnautov, A. Brito, P. Felber, C. Fetzer, F. Gregor, R. Krahn, W. Ozga, A. Martin, V. Schiavoni, F. Silva *et al.*, "Pubsub-sgx: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2018, pp. 123–132.

[16] Google, "Firebase API Reference." [Online]. Available: https://firebase.google.com/docs/reference?hl=es

[17] MQTT, "MQTT - Message Queue Telemetry Transport," Apr. 2020. [Online]. Available: http://mqtt.org/

[18] IEEE ComSoc, "Newsweek: We're Surrounded by Billions of Internet-connected (IoT) Devices. Can We Trust Them? – Technology Blog," 2019. [Online]. Available: https://techblog.comsoc.org/2019/10/26/newsweek-were-surrounded-by-billions-of-internet-connected-iot-devices-can-we-trust-them/

[19] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Computer role-based access control models," vol. 29, no. 2, pp. 38–47, feb 1996.

[20] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas, "Flexible team-based access control using contexts," in *Proceedings of Sixth ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*. New York, New York, USA: ACM Press, 2001, pp. 21–27. [Online]. Available: http://portal.acm.org/citation.cfm?doid=373256.373259

[21] A. K. Malik and S. Dustdar, "Enhanced sharing and privacy in distributed information sharing environments," in *Proceedings of the 2011 7th International Conference on Information Assurance and Security, IAS 2011*, 2011, pp. 286–291.

[22] A. U. Rahman, A. W. Malik, V. Sati, A. Chopra, and S. D. Ravana, "Context-aware opportunistic computing in vehicle-to-vehicle networks," *Vehicular Communications*, vol. 24, p. 100236, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209620300073

[23] H. R. Cheon and J. H. Kim, "Social Context-Aware Mobile Data Offloading Algorithm via Small Cell Backhaul Networks," *IEEE Access*, vol. 7, pp. 39 030–39 040, 2019.

[24] "Facebook safety check." [Online]. Available: https://www.facebook.com/about/safetycheck/