SureSpace: orchestrating beacons and witnesses to certify device location

João Tiago
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa
Lisbon, Portugal
joao.marques.tiago@tecnico.pt

Samih Eisa
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa
Lisbon, Portugal
samih.eisa@inesc-id.pt

Miguel L. Pardal
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa
Lisbon, Portugal
miguel.pardal@tecnico.ulisboa.pt

ABSTRACT

Location-aware mobile applications are increasingly popular and useful. However, as more services rely on location, there are concerns that users may misreport their location to gain undue advantages. One way to prevent such location spoofing is to rely on location certification systems. For example, SureThing uses Wi-Fi or Bluetooth beacons and ad-hoc witnesses to allow a user to make proof of location at a specific time and place. This approach can be extended to smart spaces, such as smart buildings, managed by platforms like DS2OS. In this work, we present *SureSpace*, a new system that combines location certification with smart space management, to verify the location of users in rooms inside smart buildings. The new system relies on a prover mobile device and on existing infrastructure in the room to act as signal beacons and witnesses. The system is evaluated and shown to be effective using light and audio signals to achieve security by diversity and thwart location spoofing attacks.

CCS CONCEPTS

• **Hardware** → **Sensors and actuators**; • **Information systems** → **Location based services**; • **Security and privacy** → **Access control**;

KEYWORDS

Internet of Things, Location Certification Systems, Smart Space Orchestration, Signal Processing

ACM Reference Format:
João Tiago, Samih Eisa, and Miguel L. Pardal. 2022. SureSpace: orchestrating beacons and witnesses to, certify device location. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, .* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3477314.3508382

1 INTRODUCTION

Mobile applications have gained popularity in the last decade, given the increased ubiquity and pervasiveness of mobile devices in the everyday lives of people. Meanwhile, the Internet of Things (IoT) [13] is emerging as a network of interconnected smart devices that collect and consume information for management and decision-making purposes in smart environments, depending upon characteristics like network availability or coverage area [5]. Mobile devices can interact with IoT devices for added value services, but security is a critical concern. Context attributes like identity, time and location, need to be trusted to allow for good security decisions when granting access to resources [1]. In particular, the use of *location* in mobile applications is increasingly popular and useful. However, in most cases, location is collected by the device itself, and the user may tamper with the system to misreport location and gain undue advantages when accessing services. For example, in a laboratory facility inside a building, the user may perform this attack to activate an equipment but without being inside the room, as was required by the security policy.

To prevent location spoofing, *location certification mechanisms* can be deployed to prove that a user is at a specific location, either geographical or logical, at a specific time. SureThing [8] is a proof of location system built with Java on the Android platform that uses location measurements collected with GPS, Wi-Fi and Bluetooth Low Energy (BLE) to certify location. SureThing makes use of *witness* devices to verify and attest to the presence of users in crowded physical spaces with diverse devices. In its current version, it does not use existing infrastructure available at a location.

With this work, we introduce *SureSpace*, an extended version of the original SureThing, designed to engage in smart environments in a secure way. To generate a location proof, *beacons*, which are on-site devices, broadcast unique signals meant to be captured by the *prover* device and by *witness* devices. If the captured signal matches the original signal to a certain threshold, the location proof is deemed valid. Different approaches and techniques are supported for signal processing and matching. To discover, configure and control beacons inside the smart space, we use DS2OS [11], a smart space management framework. It provides a device discovery mechanism to handle the high dynamism of smart environments, allowing beacons and witnesses to be added or removed at run-time, and used for location proofs.

In terms of security properties [3], the main objective of this work is to preserve the *integrity* of location information, i.e., prevent location spoofing; *confidentiality* is addressed by the use of standard secure communication technologies at the data-link and transport levels; *availability* is addressed by having the technique localized in specific rooms with limited communication range, and through the use of redundant devices.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC '22, April 25–29, 2022, Virtual Event,
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/10.1145/3477314.3508382

2 BACKGROUND

In this Section, we present the two systems that were composed to build SureSpace: SureThing and DS2OS.

2.1 SureThing

SureThing [8] is a location certification system for Android devices that lets users prove their location. Its design and architecture are influenced by other location certification systems, such as APPLAUS [15], and Crepuscolo [6], to the extent that they share some core components. To locate the user, SureThing supports different *location estimation techniques*, such as Geo Proof (geographic location obtained from GPS or ANLP[1]), Wi-Fi Proof (via Wi-Fi fingerprinting), and Beacon Proof (using BLE beacons).

The goal of location certification is to prevent location spoofing, by requiring proof of a claimed location. Using their Android smartphone, the *prover* says that it is at a specific location, and the system challenges the claim, by asking for *evidence*, and/or one or more *witnesses* at the location. The proof mechanism starts by collecting some sort of *signal*, that is unique to the location at a specific time, for example, something that is being transmitted over the air. The prover collects the signal, with errors, and keeps the evidence for later verification. When the prover wants to make proof of location and time, they are challenged by the verifier. The verifier can ask for the signal or some of its features, and compare them to observations made by witnesses at the same location, or compare with a known signal template.

Since the original SureThing paper, the work is being extended to become a general-purpose framework. There have been other applications implemented with the approach, some with fixed witnesses [14], other with beacons that transmit pseudo-random signal sequences and ad-hoc witnesses [10]. There is also work to provide *privacy protection* to users and witnesses through the use of *differencial privacy* [7] and *pseudonyms* [4]. However, until this current work, SureThing lacked a way to leverage multiple signals and the device orchestration capabilities available in smart buildings.

2.2 DS2OS

DS2OS [11] is a smart space orchestration framework focused on the development of services for smart spaces. Smart devices deliver valuable data (e.g. light or temperature conditions), and perform useful work (e.g. turn a light or heating on). These capabilities can be used to implement scenarios where smart devices work together towards a common goal. The coordinated management of smart devices, known as smart space orchestration, is possible if smart devices can be interconnected to share data over a network.

In DS2OS, services are logical processes that deliver functionality in a smart space, usually grouped in two categories: *adaptation services*, that provide an interface to a smart device, and *orchestration services*, that implement the logic behind pervasive use scenarios, e.g., a single command to make a full room preparation for presentation mode, including lights, temperature, shades and sound; and then another command to revert back to meeting mode. In practice, each smart device requires an adaptation service and smart devices are coordinated by an orchestration service. Regardless of their category, services have unique identifiers in DS2OS, named after the nature of the service (e.g., the adaptation service for a temperature sensor *could* go by `temperatureadaptationservice`).

Smart devices produce and consume pieces of unstructured information, known as *context*. To become manageable, context is shaped into *context models*, that represent entities in a structured way. Context models have *context nodes*, i.e. attributes, to describe properties of the entity they are linked to. To determine its type, each context node has a `type` attribute, which is, in fact, a context model itself, usually simpler. To understand this better, consider a context model of a temperature sensor, of type `/sensor/temperature`. The context model should include, at least, two properties: `isOn`, a `/boolean` for the operational status of the sensor, and `value`, a `/number` used to represent the read temperature. The `temperatureadaptationservice` would, then, include a context node of type `/sensor/temperature` to represent the sensor. In other words, all properties of the sensor's context model would be implicitly included in that context node.

Each service has its own context model, and inter-service communication is achieved through the manipulation of context models. Simply put, a context model is a *blackboard*: some services write on it, and other services read from it to achieve their objective. To prevent unauthorized operations, context nodes have read and write permissions. Following the example, all services should be able to change the value of `isOn`, so that the sensor can be turned on. However, `value` must be read-only to all services but the `temperatureadaptationservice`, authorized to update the temperature value. In this light, context models are interfaces for services, a sort of *service contract* that (1) specifies which attributes can be read and/or written, via `get` and `set` operations, respectively, and (2) by whom, according to an *access control policy*. For instance, if an orchestration service wishes to get the current temperature, it would have to call `set('isOn', true)` on the context model of the `temperatureadaptationservice` (to enable the sensor), and then `get('value')` to get the actual temperature.

Context models are stored in a distributed system over a peer-to-peer network, known as the Virtual State Layer (VSL). Peers of the network are called Knowledge Agents (KAs), and each agent persists a subset of the context models. To be granted access to the distributed knowledge, services register to a KA of their choice, that becomes responsible for the respective context models. From that moment on, these context models become available to other services, even if connected to a different agent.

One important feature of DS2SOS is that services can subscribe to specific context nodes to receive a notification when the node value changes. This feature is useful, for instance, when a service wants to take different actions depending on the new value. Another key feature of DS2OS is dynamic service discoverability [2, 12], that allows adaptation services to be discovered by different criteria: type of smart device (e.g. `temperatureadaptationservice`), or type of attribute (e.g. `/boolean`).

3 SURESPACE

SureSpace is a location certification system designed for smart environments. Its architecture integrates elements from SureThing,

[1] The Android Network Location Provider, that uses both cell tower and Wi-Fi to determine the device location.

that enriches SureSpace with an assortment of techniques and technologies for location certification, and the DS2OS middleware, that offers the possibility to configure and orchestrate devices in smart spaces, as well as the possibility to dynamically discover them at run-time.

3.1 Location Representation

Geocodes represent geographic locations on Earth, where each location is assigned a unique identifier, used to distinguish between entities. Usually, geocodes are short and human-readable. In SureSpace, we use geocodes to locate both the prover and the orchestrator within the building. Our problem domain allows locations to be represented with coarser granularity, because the focus is on determining if the prover is within the boundaries of the room, and not necessarily at a specific location within that area.

SureSpace supports two geocode systems: Open Location Code (OLC), and What3Words (W3W). By default, SureSpace uses the OLC geocode system to encode locations into *plus codes* that represent squares on the surface of the Earth. The longer the code, the smaller the square is, and, in its full length of 11 characters, a plus code represents a $3 \times 3\ m$ square. For example, `8CCGPVP5+GMW` describes the reception hall of a building in Lisbon. By shortening the code to `8CCGPVP5+GM`, it describes a larger square containing the reception hall. This feature is useful to represent larger rooms with a single plus code.

3.2 Components

Figure 1 outlines all the components of SureSpace, focusing on communication flows between them. The main components are presented following the expected interaction sequences.

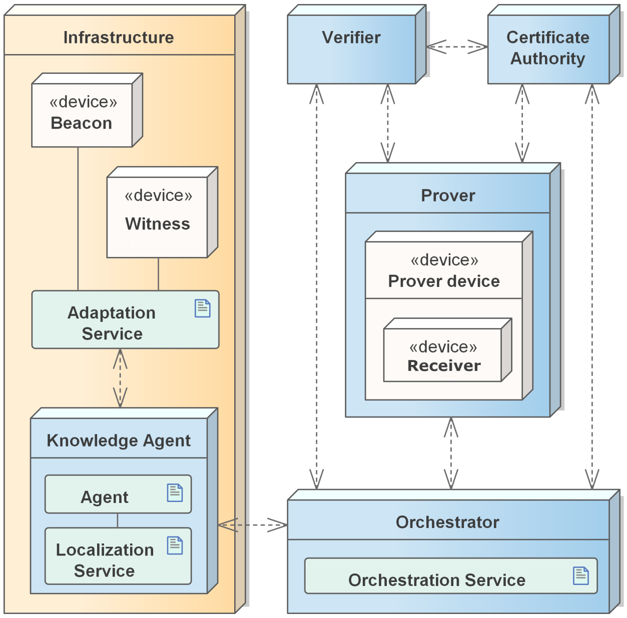

Figure 1: SureSpace architecture.

3.2.1 *Prover.*

The prover is a user of SureSpace that engages with the system in order to prove their location. The prover device is the device used by the prover during all interactions with the system. It includes one or more signal receivers.

3.2.2 *Certificate Authority.*

The Certificate Authority (CA) is the long-term identity provider of the active entities of the system, similar to CAs for website certification in the Internet. Entities must register themselves to the CA to be deemed legitimate and to be able to engage with SureSpace. Each entity generates a public/private key pair. The private key is known only to the entity, and is kept safe and secure on their side. The public key is used to generate a certificate signing request in order to apply for a public key certificate. If the request is approved by the CA, a public key certificate is issued and assigned to the requester entity.

3.2.3 *Orchestrator.*

The orchestrator is the entry point to SureSpace system, because it is the component that the prover first reaches out to. It is at the center of the proof of location, responsible for coordinating the process at the highest level, and preventing malicious communication flows coming from unauthorized parties. It implements diverse logical subprocesses that include: the dynamic discovery of new orchestrated rooms, the dynamic discovery of new beacons and their orchestration, and the delivery of accurate information about a specific location proof. Orchestrators run an *orchestration service* to communicate with the adaptation services of the orchestrated beacons.

3.2.4 *Knowledge Agent.*

A Knowledge Agent (KA) is a node in the distributed knowledge network of DS2OS (cf. 2.2). More specifically, KAs are context repositories that persist relevant information used by the orchestrator. To deliver room-level orchestration, each orchestrated room has its own KA, that behaves like a *proxy* to the room . No rooms share the same KA. Agents hold information about their geographical location by running a *localization service*, so that the orchestrator can discover them by location when looking for new orchestrated rooms. Since beacons register themselves to the closest KA in their vicinity, new beacons are easily discoverable and accounted for. During a proof of location, an orchestrator will need information about beacons and witness devices, like the value of specific configuration attributes. That information becomes available to other KAs, since they are all nodes in the same distributed knowledge network.

3.2.5 *Beacon.*

A *beacon* is a device embedded into the trusted infrastructure ready to be used in a proof of location (e.g. a smart bulb). By default, SureSpace is not aware of existing beacons by themselves, since they may not be directly connected to the system. Thus, each beacon requires a *proxy* (adaptation service) to become visible and controllable by the system. During the proof of location, each beacon generates and broadcasts exactly one *signal* to be captured by, at least, one corresponding receiver in the prover device.

3.2.6 *Signal.*

A *signal* is something produced by a beacon that can be received by a corresponding receiver. For instance, visible light, emitted by a smart bulb (the beacon), and acknowledged by a light sensor (the receiver), could be used as a signal. The definition of signal, however, is left open to avoid narrowing down the possibilities to a small set of conventional signals, paving the way for future "out-of-the-box" ideas. Theoretically, any equipment can be used as a receiver, provided it is able to receive signals from a specific beacon.

3.2.7 Proof Evidence. A signal is generated based on a set of *quirky properties*, that feed a deterministic signal generator. If these properties are disclosed, the original signal can be easily replicated. Naturally, signals have different characteristics/features, and are susceptible to deterioration induced by multiple factors during their transmission. Moreover, receivers have limited capabilities, and might not be able to acknowledge all characteristics of the signal, but only a subset of them. Thus, what the receiver gets is not the original signal, but a *degraded* representation of some of its characteristics. In some cases, part of the characteristics of the signal can be successfully derived from the analysis and/or processing of the degraded representation. To the derived information we call *proof evidence*, because it represents the available information of the original signal.

3.2.8 Receivers and Witnesses. To determine the legitimacy of the location claim, we measure the accuracy of the proof evidence by quantifying similarity between the original signal, that was transmitted by the beacon, and the degraded representation, that was received by the prover. To do that, we either need a template of the original signal, or we require another representation of the signal, captured by a witness that is embedded into the infrastructure at the location. If the prover signal and witness signal representations share the same set of signal characteristics, then we conclude that the signals are similar, and that the prover received the transmission of the original signal. The beacon and witness devices are part of the trusted infrastructure, and they are controlled by the same adaptation service.

3.2.9 Verifier. The verifier is the last entity to be engaged in SureSpace. It measures the reliability of a location claim by assessing the legitimacy of the proof evidence presented by the prover. The verifier implements adequate criteria to compare different representations of the same signal. However, there is no pre-determined assessment criteria, because (1) the definition of signal itself remains abstract enough to encompass a variety of beacons, and (2) application-specific criteria might have to be taken into account. Regardless of the implementation details, the verifier must output a boolean value that represents the assessment result. If `true`, the location evidence is deemed reliable, and, thus, the location claim is accepted as a location proof.

3.3 Communication between Entities

Depending on their domain, entities communicate using different underlying communication protocols. DS2OS entities (KAs and services) communicate with the VSL via REST connectors (using HTTPS for security). Communications between the SureSpace entities (CA, orchestrator, and verifier) are supported by gRPC (in Java)[2], following a remote invocation paradigm [9]. gRPC uses Protocol Buffers (*protobuf*) to provide a platform-independent representation of the remote interfaces, and it was chosen because of its efficiency, and loose coupling between clients and servers. Moreover, communications between SureSpace entities share a common payload format, illustrated in Figure 2, that (1) allows for a standardized process of message validation, and (2) fosters the implementation of security protections for integrity, authentication, and non-repudiation. All messages hold information about their source (field `sender`), and destination (field `receiver`), to prevent message forwarding. Replay attacks are mitigated by using a securely generated random number (field `nonce`). For integrity, authentication, and non-repudiation purposes, a *digital signature* is generated over the message (field `signature`) using `SHA-512` with the private key of the source entity. To reduce the number of interactions with CAs, the certificate of the source entity is attached to the body of the message prior to the signing.

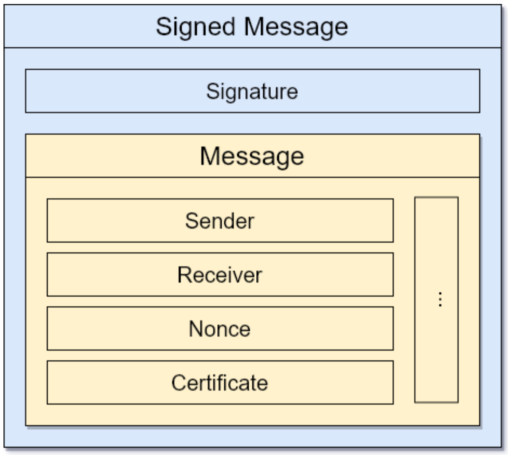

Figure 2: Message structure shared in SureSpace.

3.4 Entity Identification

Each SureSpace entity is assigned a public key certificate that is part of a certificate chain that terminates with the SureSpace Root CA. Public/private key pairs are generated using `2048-bit RSA`, and certificates are signed using `SHA-512`. Moreover, each entity is identified by a hierarchical identifier, unique within the SureSpace domain.

4 LOCATION CERTIFICATION

The location certification process encompasses three main stages: *pre-authorization*, *proof*, and *verification*.

4.1 Pre-authorization Stage

A proof of location requires orchestration of a subset of beacons scattered across a smart location, like a room. The orchestration requires context information to be readily available at proof-time, like which beacons are engaged with the proof of location.

The objective of this pre-stage, represented in Figure 3, is to produce an *authorization*, requested by the prover and issued by an orchestrator, that works as a *token* used later to trigger the proof stage, while containing relevant metadata necessary for the proof of location.

[2] https://www.grpc.io/

First, the prover device determines its compatibility with known beacons (*Device compatibility check* step). A beacon is deemed supported by the prover device if and only if it has, at least, one compatible receiver (usually a sensor, like a light sensor). Consequently, the set of supported beacons depends on the hardware properties of the prover device. The prover device estimates its location resorting to external mechanisms (*Device location estimation* step). For instance, GPS can be used if the signal is strong enough. Upon locating the prover device, GPS coordinates are then converted to a plus code (refer to 3.1). In the absence of a suitable localization system (like in GPS-constrained environments), scanning an on-site QR code with the geocode of the room might be sufficient to locate the prover within the building. In the end of this step, the prover requests a proof authorization to the orchestrator (*Request authorization* step).

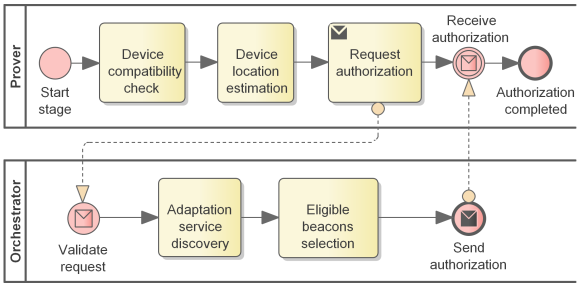

Figure 3: Process diagram for the pre-authorization stage.

Following the validation of the request, the orchestrator determines which beacons are available at the location reported by the prover (step *Adaptation service discovery*). Different orchestrated rooms offer different beacons, discoverable via adaptation services delivered within that room. To determine which beacons are available at the reported location, the orchestrator (1) lists all orchestrated rooms, (2) determines the closest one to the prover, and (3) discovers which beacons are available in that room.

Finally, the orchestrator selects beacons eligible for the proof of location (*Eligible beacons selection* step). A beacon is eligible if and only if (1) it is supported by the prover device, and (2) available at the prover location. A beacon selection policy might filter eligible beacons, depending on policy criteria (e.g. the security level of the room). In the end of this step, the orchestrator generates an authorization token, stores it for future use, and sends it to the prover.

4.2 Proof Stage

This stage, represented in Figure 4, starts when the prover submits the proof authorization token to the orchestrator (step *Submit authorization*).

Beacons generate signals based on a set of quirky properties, used to feed a deterministic signal generator. To minimize the likelihood of broadcasting the same signal twice, a seed value is generated in an unpredictable way (step *Generate random seed*), and it is used to populate quirky properties with pseudorandom values derived from it.

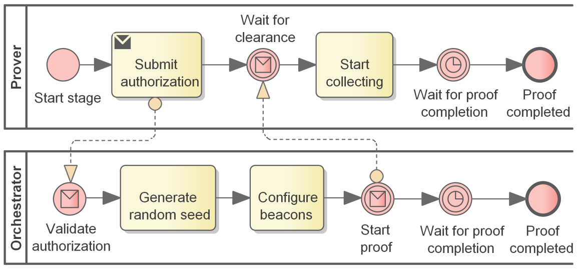

Figure 4: Process diagram for the proof stage.

The orchestrator locks the adaptation services of the selected beacons, and applies new pseudorandom values to the quirky properties (step *Configure beacons*). When beacons are ready, they start broadcasting their signal. At the same time, the prover starts receiving the signal, as well as witnesses in the infrastructure. The prover devices stores the received signal information, and the witnesses capture and share their own signal information with the orchestrator, by updating the corresponding properties of their context models in the VSL, since they are part of the infrastructure.

Network latency has direct impact on the level of synchronization between orchestrator and prover, since only upon clearance from the orchestrator will the prover initiate the process. If desynchronized, "dead times" may occur in the beginning (beacons are broadcasting, but the prover is waiting for clearance) and in the end of the process (beacons have ceased their activity, but receivers remain active). This issue is mitigated in the verification stage, without relying on clock synchronization.

4.3 Verification Stage

Theoretically, a proof of location is accepted if the proof evidence is *complete* enough (with regard to all the signals broadcast by engaging beacons), and *accurate* enough (if the extracted information is in line with the information of the original signal). Even in optimal conditions, signal degradation will decrease the accuracy of the proof evidence. Internal factors (e.g. receiver sensitivity) and external factors (e.g ambient noise, topology of the orchestrated room) might lead to a misrepresented, yet legitimate, proof evidence. To account for such errors, a *margin of error* is considered when comparing the trusted representation of the original signal with its degraded representation.

To verify the location claim, the prover submits the evidence of the different signals to the verifier. A final judgment is made, either *accepting* or *rejecting* the location claim. If the claim is accepted, it becomes a location proof.

5 IMPLEMENTATION

In this Section, we present the development platform and the details about beaconing and verification.

5.1 Development Platform

We developed a prototype of SureSpace as a Java project, using the JDK (Java Development Kit) version 11, and used Maven 3.6.3

to manage the building process and the code dependencies. We implemented a custom version of Ardulink 2, a Java solution for coordinating Arduino boards[3](required for the experimental setup), and implemented the Certificate Authority, the Orchestrator, and the Verifier from scratch (some code was partially inspired by existing work [8]). We also implemented all DS2OS services (adaptation, localization, and orchestration services) by extending available templates. To bootstrap the VSL, KAs need to be launched one by one to become peers (refer to 2.2). In practical terms, a KA is bundled as an executable JAR file. However, the code of a KA was no longer compatible with our setup, so we modified it, and recompiled it. The prover was implemented as an Android mobile application, for a richer user-experience, compiled in Java 8 against API level 30.

5.2 Beaconing Technique

To determine if a signal is appropriate for proof of location, we classify it based on two metrics: *difficulty of replication*, and *difficulty of acknowledgment*. Signals should be difficult to replicate without knowing the quirky properties used for their generation. If signals have noticeable patterns or are reused, a malicious party can easily replicate them. At the same time, signals must be versatile enough to account for common limitations shared by compatible receivers, so that signals can be easily received by most witnesses.

We propose a technique based on *time fragmentation* to reach an equilibrium between these two metrics of difficulty of replication and difficulty of acknowledgement. During the proof stage, each signal is broken into a fixed number of consecutive, same-length fragments, and each fragment is generated based on a set of quirky properties.

For simplicity, we adopt the following notation:

- $b \in B$ denotes beacon b, in the set of supported beacons, B
- $w \in W_b, b \in B$ denotes witness w, in the set of witnesses compatible with beacon b, W_b
- $f_{b,i} \in F_b, b \in B, i \geq 1$ denotes the i-th fragment of the set of fragments that compose a signal broadcast by beacon b, F_b
- $q \in Q_{f_{b,i}}, b \in B, i \geq 1$ denotes quirky property q, in the set of quirky properties used to generate fragment $f_{b,i}$, $Q_{f_{b,i}}$
- $S_b = f_{b,1} \parallel f_{b,2} \parallel \ldots \parallel f_{b,n}, b \in B, i \geq 1$ denotes a signal with $n \geq 1$ fragments, broadcast by beacon b

Algorithm 1 proposes a pseudocode of the technique. Since each fragment is very likely to have a different set of quirky properties, we assume that no two fragments share the same set of quirky properties, resulting in the creation of unique signals.

Algorithm 1 Pseudocode of the technique.

```
$s \leftarrow$ UnpredictableSeed();
for all $b \in B$ do
    $random \leftarrow$ Random($s$);
    for $i \leftarrow 1, 2, \ldots, |F_b|$ do
        for all $q \in Q_{f_{b,i}}$ do
            $q \leftarrow$ PseudorandomValue($random$);
        end for
    end for
end for
```

5.2.1 Light Signal Time Fragmentation.

Consider a light source (e.g. a LED) used as a beacon for a proof of location, b_{light}, that can switch between states on and `off` with period $P \in [P_{MIN}, P_{MAX}[$, measured in a convenient unit (P_{MIN} and P_{MAX} are, respectively, the lowest and the highest supported periods). In the proof stage, the beacon broadcasts a signal, S_{light}, split into n d-seconds fragments. During each fragment $f_{light,i}$, $i = 1, 2, \ldots, n$, the beacon switches between states with a pseudorandom period P_i (the quirky property), derived from the seed, forming a *power-on and power-off* sequence with rates that vary between fragments.

Theoretically, P_i can take any value in range $[P_{MIN}, P_{MAX}[$, and nothing prevents two pseudorandom periods from being equal or close enough to generate similar or indistinguishable fragments. To avoid this, we adopt an approach that (1) prevents reusing the same period and (2) reduces the probability of picking periods too close in the range.

As receiver, we consider a light sensor, w_{light}, capable of measuring light intensity in a convenient unit, at a sampling rate not less than ${P_i}^{-1}$, $\forall i$. Plotting the measurements over time offers a representation of S_{light} based on one of its properties (light intensity). The analysis of that representation may confirm significant variations in light intensity, which are an interpretation of the power-on and power-off sequence.

5.2.2 Audio Signal Time Fragmentation.

Consider an audio source (e.g. a speaker) used as a beacon for a proof of location, b_{audio}. The beacon can be programmed to play any song out of a predefined set of m songs, and $songId \in \{0, 1, \ldots, m-1\}$ is the index of the song to be played. This song is any regular song that plays on the radio, and we consider it over any synthesized melody because a song is more easily tolerated by the human ear for extended periods of time. During its activity, the beacon broadcasts a signal, S_{audio}, with a single d-seconds fragment, and a pseudorandom $songId \in Q_{f_{b_{audio}},1}$ (the only quirky property) is derived from the seed.

As receiver, we consider a sound sensor, w_{audio}, capable of measuring sound amplitude in a convenient unit. Plotting the measurements over time offers a representation of S_{audio} based on one of its properties (audio amplitude). The analysis of that representation may confirm variations in amplitude and frequency that match the song being played.

5.3 Supported Beacons

Adding support for a new beacon requires writing its context model with all the configurable properties (the witness requires its own context model too), and implementing the adaptation service, so that the beacon can be discoverable.

Currently, SureSpace supports two beacons: light beacon, and audio beacon; and next we go over the steps required to support them.

5.3.1 Light Beacon.

Based on the example in 5.2.1, we considered a light beacon capable of switching between states on and `off` with a configurable period. Figure 5 is a simplified context model of the beacon, where `isOn` is a boolean used to control the beacon (if set to `true`, the beacon is working), and `switchingPeriod` is the time it takes for the beacon to switch between states (in seconds).

[3] https://github.com/Ardulink/Ardulink-2

```
<model type="/complex/beacon">
   <isOn type="/basic/boolean"/>
   <switchingPeriod type="/basic/number"/>
</model>
```

Figure 5: Simplified context model of a light beacon.

As witness, we considered a light sensor capable of measuring light intensity at a configurable sampling rate not less than `switchingPeriod`$^{-1}$. Figure 6 is a simplified context model of the witness, where `intensity` is the measured light intensity (in a convenient unit), `intensitySamplingRate` is the sampling rate at which the sensor is reading (in Hertz), and `isOn` is a boolean used to control the witness (if set to `true`, the witness is working). The orchestration service subscribes the `intensity` attribute on the witness context model. Every time the attribute value changes because of a new measurement, the orchestration service is notified. At that moment, the value is timestamped (in milliseconds), and stored in the orchestrator to compose the trusted representation of the light signal.

```
<model type="/complex/witness">
   <intensity type="/basic/number"/>
   <intensitySamplingRate type="/basic/number"/>
   <isOn type="/basic/boolean"/>
</model>
```

Figure 6: Simplified context model of a light witness.

As receiver, we consider any device capable of measuring light intensity.

5.3.2 Audio Beacon. Based on the example in 5.2.2, we considered an audio beacon capable of playing a specific song out of a set of songs stored in a raw format without compression (like WAV). Figure 7 is a simplified context model of the beacon, where `isOn` is a boolean used to control the beacon (if set to `true`, the beacon is working), and `songId` is the index of the song to be played.

```
<model type="/complex/beacon">
   <isOn type="/basic/boolean"/>
   <songId type="/basic/number"/>
</model>
```

Figure 7: Simplified context model of an audio beacon.

As witness, we considered a sound sensor capable of measuring the sound amplitude at a specified sampling rate. Figure 8 is a simplified context model of the witness, where `amplitude` is the measured sound amplitude (in a convenient unit), `amplitudeSamplingRate` is the sampling rate at which the sensor is reading (in Hertz), and `isOn` is a boolean used to control the witness (if set to `true`, the witness is working).

Since we have access to the songs in a raw format, it is possible to use this source as a witness signal instead of a captured signal. In practice, this is as if the signal captured by the witness did not contain any errors. Every `amplitudeSamplingRate`$^{-1}$ s, the witness reads the sound amplitude from the sound file, and updates the `amplitude` attribute on its context model, which has been subscribed by the orchestration service. Every time the attribute value changes, the orchestration service is notified. At that moment, the value is timestamped (in milliseconds), and stored in the orchestrator to compose the representation of the audio signal.

```
<model type="/complex/witness">
   <amplitude type="/basic/number"/>
   <amplitudeSamplingRate type="/basic/number"/>
   <isOn type="/basic/boolean"/>
</model>
```

Figure 8: Simplified context model of an audio witness.

As receiver, we consider any device capable of measuring sound amplitude.

5.4 Verifier Implementation

To verify a location proof, the verifier quantifies similarity between different representations of the same signal: the one received by the prover device, and the one captured by the witness in the infrastructure. If more than one beacon is used (and, thus, more than one signal is involved), individual similarity estimates are weighted for a final similarity estimate. The same can be done to support multiple witnesses.

Based on the beacons we support, we use the MATLAB Engine API for Java to quantify similarity. Next, we detail the approach used for comparing representations of the same signal, for both light signals and audio signals.

5.4.1 Light Signal Similarity. Representations may be sampled at different rates, impeding their comparison. We bring them to a common rate by upsampling the representation with the lowest frequency, using linear interpolation. This process produces an approximation of the representation that would have been obtained by sampling at a higher rate.

Because clock synchronization is not a requirement, potential delays between representations may exist. To align them without relying on timestamps, we use correlation to determine where representations overlap the most, and then align them.

At last, we normalize both representations, and calculate the linear correlation coefficient between them, $corr_{light\ 1}$, given by Equation 1

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1}$$

where r is the linear correlation coefficient, n is the number of samples in the representations (they have the same size), x_i and y_i are the sample points, and $\bar{x}$ and $\bar{y}$ are the means of the samples. This coefficient measures the linear relationship between the two representations, and is used as an estimate of their similarity.

5.4.2 Audio Signal Similarity. Audio and light signals are fundamentally different, and so are their representations. For that reason, we cannot follow the previous approach. Instead, after normalization, we use dynamic time warping to resample and align the audio representations. This algorithm stretches the two representations onto a common set of instants, such that the sum of the Euclidean distances between corresponding points is smallest. Then, we calculate the linear correlation coefficient between the aligned representations, $corr_{audio\ 1}$, and use it as a first estimate of their similarity.

To improve the estimate, we calculate the power spectrum of the two representations. Simply put, a power spectrum is a frequency-domain interpretation of an audio signal representation because it describes the distribution of power (sound amplitude) into frequency components. In this context, this information is relevant because each song has a different time-frequency structure. Thus, we calculate the linear correlation coefficient between the power spectra of the two representations, $corr_{audio\ 2}$, and use it as a second estimate of their similarity.

The two estimates are weighted for a final similarity estimate, given by Equation 2

$$w_1 \times corr_{audio\ 1} + w_2 \times corr_{audio\ 2}, w_2 = 1 - w_1 \tag{2}$$

Weights w_1 and w_2 must be tuned based on a training set, since they are necessarily beacon- and witness-dependent.

5.4.3 Combined Signal Similarity. For a final similarity estimate, individual similarities are estimated according to Equation 3

$$\begin{aligned} w_3 \times corr_{light\ 1} + \\ w_4 \times (w_1 \times corr_{audio\ 1} + w_2 \times corr_{audio\ 2}), \\ w_4 = 1 - w_3 \end{aligned} \tag{3}$$

Weights w_3 and w_4 need to be tuned.

6 EVALUATION

In this Section, we present the experimental setup used to evaluate the SureSpace prototype, describe the evaluation criteria, and discuss the evaluation results.

6.1 Experimental Setup

For the experimental setup we used inexpensive equipment, with less accuracy, but more representative of commodity equipment that we expect to find in a smart building. We opted for a small, yet representative, orchestrated area of our laboratory, represented in Figure 9.

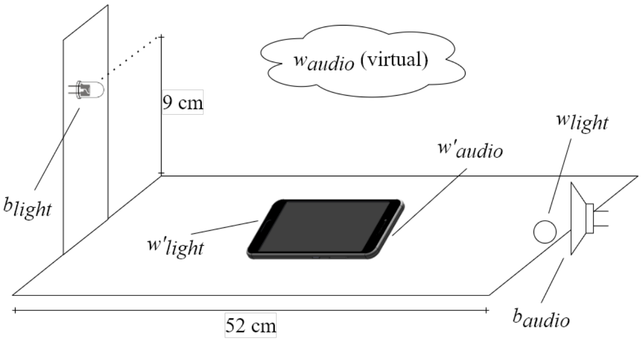

Figure 9: Experimental setup components and orchestrated area.

Recall the notation introduced in 5.2, where b denotes a supported beacon, and w a corresponding receiver. Based on the supported beacons, we used a Grove Chainable RGB Led V2.0 as light beacon, b_{light}, a Grove Light Sensor V1.2 as light receiver, w_{light}, and a JBL GO 2, connected to a Grove MP3 V2.0 module, as audio beacon, b_{audio}. The audio witness, w_{audio}, relies on a template sound file, eliminating the need for physical capture equipment (cf. 5.3.2). For connectivity reasons, b_{light}, w_{light}, and the Grove MP3 V2.0 module were all connected to a Grove Base Shield V2.0 for an Arduino Uno board. The prover device was a Huawei Mate 20 Pro Android smartphone, shipped with Android 10, equipped with a built-in ambient light sensor, the light receiver, w'_{light}, and a microphone, the audio receiver, w'_{audio}. During the proof of location, the prover device is steady in the center of the orchestrated area, as depicted.

We built a dataset by running 80 location proofs under the same *controlled scenario*. Each location proof delivered *four* signal representations (two representations per signal, a trusted one and a degraded one). To be assessed, a location proof is submitted to the verifier (refer to 5.4).

In some cases, the verifier may classify location proofs incorrectly, either by accepting a location proof that should be rejected (false positive), or, conversely, by rejecting a location proof that should be accepted (false negative). To evaluate SureSpace, we consider (1) the false positive rate (*FPR*), which is the percentage of all negatives that still yield positive, (2) the false negative rate (*FNR*), which is the percentage of positives that yield negative, and (3) the success rate, which is the percentage of correct classifications. *FPR* and *FNR* are inversely proportional to the success rate, and the verifier should focus on minimizing them.

To ensure the reproducibility of the experiments, the duration of the proof stage was set to 30 seconds in all proofs of location (a reasonable value in a human time-scale that works in a possible meeting room scenario). Regarding the audio component, the beacon b_{audio} could choose between 20 different predefined songs, all sampled at 44.1 kHz (refer to 5.2.2). Regarding the light component, the light signals were broken into two 15-seconds fragments, and beacon b_{light} could switch between states with a pseudorandom period $P \in [0.5, 7.5[$ (refer to 5.2.1). The upper bound of the range is $7.5s$ ($= \frac{15s}{2}$) to ensure that light signal fragments generate, at least, one complete `on-off` sequence, i.e., the beacon is powered on, and then it is powered off for the same amount of time at least once. The lower bound of the range is $0.5s$ to ensure compatibility with all light receivers (based on the information we collected, w'_{light}, the Android ambient light sensor, is the light receiver that reports the lowest sampling rate of $\approx 2Hz$).

6.2 Optimal Weight Tuning

We started by dividing our dataset into two subsets: a training set, and a test set. Following the Pareto principle, the training set accounted for 80% of the dataset (i.e. 64 location proofs), and the testing set accounted for the remaining (i.e. 16 location proofs).

We started by tuning weights w_1 and w_2 using the training set (refer to 5.4.2). We crossed audio signal representations from all location proofs in the training set, ending with 4096 combinations

(64 × 64), from which 64 should be accepted (the number of legitimate location proofs), and 4032 should be rejected (the number of fabricated location proofs). We varied w_1 (recall that $w_2 = 1 - w_1$) to find the optimal combination that would minimize $FPR + FNR$. Figure 10 plots the sum as a function of w_1. The local minima is at $w_1 = 0.661$, which means that $\langle w_1, w_2 \rangle = \langle 0.661, 0.339 \rangle$ offers the best success rate ($FPR = 33.33\%$ and $FNR = 37.50\%$). Replacing w_1 and w_2 in Equation 2, the audio signal representations similarity estimate is given by Equation 4

$$0.661 \times corr_{audio\ 1} + 0.339 \times corr_{audio\ 2} \tag{4}$$

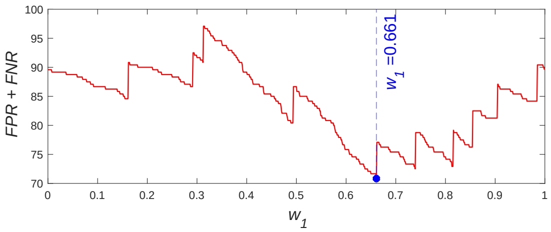

Figure 10: Optimal value of w_1.

Then, we tuned weights w_3 and w_4 (cf. 5.4.3). Following the same approach, we crossed audio and light signals from all location proofs in the training set, and varied w_3 (recall that $w_4 = 1 - w_3$) to find the optimal combination that would minimize $FPR + FNR$. Figure 11 plots the sum as a function of w_3. The local minima is at $w_3 = 0.436$, which means that $\langle w_3, w_4 \rangle = \langle 0.436, 0.564 \rangle$ offers the best success rate ($FPR = 7.50\%$ and $FNR = 6.25\%$). Replacing all weights in Equation 3, the final similarity estimate is given by Equation 5

$$\begin{aligned} 0.436 \times corr_{light\ 1} + \\ 0.564 \times (0.661 \times corr_{audio\ 1} + 0.339 \times corr_{audio\ 2}) \end{aligned} \tag{5}$$

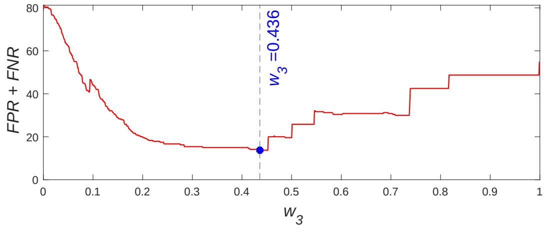

Figure 11: Optimal value of w_3.

6.3 Approach Effectiveness

We validated our approach by testing our signal representation similarity estimate against the testing set. We crossed light and audio signal representations from all location proofs in the testing set, ending with 256 combinations (16 × 16), from which 16 should be accepted (the number of legitimate location proofs), and 240 should be rejected (the number of fabricated location proofs). In the end, the verifier classified location proofs correctly in 94.78% of the cases, with rates $FPR = 5.06\%$ and $FNR = 15.63\%$.

For demonstration purposes, we consider the two light signal representations of a location proof accepted by the verifier. Figure 12 plots the normalized light intensity read by the witness, w_{light}, and the prover receiver, w'_{light}), shown in different colors. It becomes evident that the light signal is split into two fragments, with the second fragment starting at around 15 s. At that moment, the rate at which b_{light} changes between states on and off decreases.

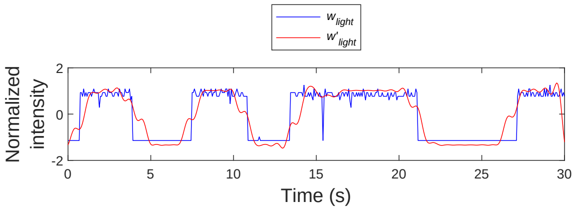

Figure 12: Light signal representations overlapping.

The witness signal representation is sharper and has more spikes than the prover signal representation. This difference can be explained by the very different sampling rates at which both work. Based on the information we collected during the experiments, w_{light} works at ≈ 14 Hz, while w'_{light} works at ≈ 2 Hz. For that reason, the witness is aware of the beginning of the second fragment, while the prover misses it, since it occurs in-between samples. Notwithstanding, representations overlap, suggesting both the prover and the witness were exposed to the same light conditions.

Additionally, consider the two audio signals representations of a location proof accepted by the verifier. Figure 13 plots, in different colors, the normalized sound amplitude read by the witness based on a template, w_{audio}, and the prover receiver, w'_{audio}. Although both representations overlap, the witness representation has stronger and neater variations in sound amplitude, while the prover representation looks more sketchy, with periods of constant sound amplitude. Once again, this is a consequence of sampling the same sound at different sampling rates. Based on the information we collected during the experiments, w_{audio} works at ≈ 30 Hz, while w'_{audio} works at ≈ 12 Hz. The power spectra of both representations, depicted in Figure 14, shows prominent peaks in magnitude occurring around the same frequencies. This suggests that the prover and the witness were capturing the same song being played by b_{audio}.

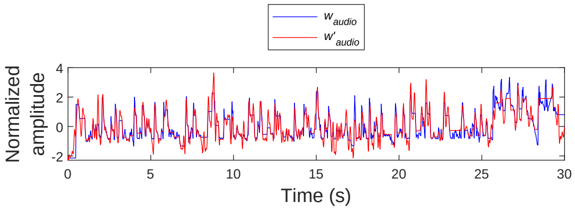

Figure 13: Audio signal representations overlapping.

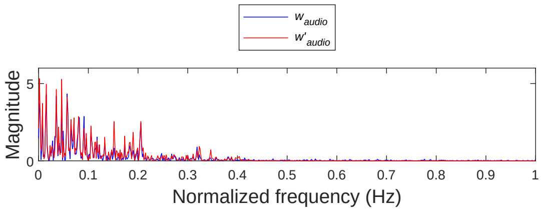

Figure 14: Audio signal representations spectra overlapping.

6.4 Resistance to Attacks

We consider that SureSpace has successfully resisted an attack when it rejects illegitimate location proofs. In other words, we are interested in the number of false positives, and, thus, consider FPR as the metric to evaluate SureSpace's resistance to attacks.

Based on the attacker model, we consider two attackers that try to prove their false presence by manipulating signals:

A1 Receives the signal from exactly one random beacon, but not from the others (beacons may have different radius of action);

A2 Combines legitimate signals representations to derive synthesized proof evidence from them.

Attacker A1 can either (1) receive the light signal but not the audio signal, or (2) receive the audio signal but not the light signal. Based on Equation 5, we confirm that signals have similar weights (0.436 versus 0.564), so we expect location proofs to be rejected if an attacker only provides the representation of one of the signals. To simulate attacker A1, we used the 240 fabricated location proofs that should be rejected by the verifier, and, according to the scenario, (1) or (2), we discarded one of the signal representations when submitting the location proof to the verifier. In both scenarios, we determined $FPR = 0.00$ % — it means that the verifier successfully rejected all location proofs fabricated by the attacker.

To simulate attacker A2, we combined legitimate signal representations from different location proofs to fabricate new illegitimate location proofs. In 6.3, we determined that $FPR = 5.06$ %. In fact, combining signals from different location proofs is not an effective attack because the verifier is capable of telling they were generated for different location proofs (since the likelihood of broadcasting the same signal is low).

7 CONCLUSION AND FUTURE WORK

In this paper, we presented SureSpace, a location certification system for smart environments. It leverages the capabilities of both SureThing, that defines procedures and techniques for location certification, and DS2OS, that provides control over diverse smart devices for orchestration purposes. SureSpace relies on smart devices that broadcast preconfigured signals, and receivers to capture these signals, to certify location at a specific time and place. The system was evaluated in laboratory conditions with inexpensive Arduino-compatible equipment. It was shown to be effective using light and audio signals, with a success rate up to 94.78%. Moreover, we evaluated the resistance to attacks by simulating the capabilities of possible attackers, and thwarted all attempts.

As future work, we consider the possibility of implementing more complex methods for optimal weight tuning, that may further improve the effectiveness of SureSpace. We also consider a new way to verify location proofs based on a *challenge-response* approach where, instead of disclosing the complete captured signal to the verifier, the prover could be challenged to answer queries about specific components of the location proof. For example, and considering the light signal, we could consider questions like "For how long was the beacon in the on state?".

Finally, SureSpace could be further evaluated in a smart building environment outside the laboratory, with a check-in application for physical meetings, with an assessment of the full user experience.

ACKNOWLEDGEMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and through the SureThing project, with reference PTDC/CCI-COM/31440/2017.

REFERENCES

[1] Claudio A. Ardagna, Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. 2006. Supporting Location-Based Conditions in Access Control Policies. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. Association for Computing Machinery, New York, NY, USA, 212–222. https://doi.org/10.1145/1128817.1128850

[2] Georg Aures and Christian Lübben. 2019. DDS vs. MQTT vs. VSL for IoT. *Network* 1 (2019).

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (Jan 2004), 11–33.

[4] Norbert Bißmeyer, Jonathan Petit, and Kpatcha M Bayarou. 2013. CoPRA: Conditional pseudonym resolution algorithm in VANETs. In *10th Annual conference on Wireless On-Demand Network Systems and Services (WONS)*. IEEE, 9–16.

[5] AH Buckman, Martin Mayfield, and Stephen BM Beck. 2014. What is a smart building? *Smart and Sustainable Built Environment* 3, 2 (2014), 92–109.

[6] Eyüp S Canlar, Mauro Conti, Bruno Crispo, and Roberto Di Pietro. 2013. Crepuscolo: A collusion resistant privacy preserving location verification system. In *International Conference on Risks and Security of Internet and Systems (CRiSIS)*. IEEE, 1–9.

[7] João Costa and Miguel L. Pardal. 2020. *A Witness Protection for a Privacy-Preserving Location Proof System*. Dissertation. Instituto Superior Técnico, Universidade de Lisboa.

[8] João Ferreira and Miguel L. Pardal. 2018. Witness-based location proofs for mobile devices. In *17th IEEE International Symposium on Network Computing and Applications (NCA)*.

[9] Kasun Indrasiri and Danesh Kuruppu. 2020. *gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes.* " O'Reilly Media, Inc.".

[10] Gabriel A. Maia, Rui L. Claro, and Miguel L. Pardal. 2020. CROSS City: Wi-Fi Location Proofs for Smart Tourism. In *The 19th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now)*. Bari, Italy.

[11] Marc-Oliver Pahl. 2014. *Distributed Smart Space Orchestration*. Dissertation. Technische Universität München, Munich, Germany.

[12] Marc-Oliver Pahl and Stefan Liebald. 2019. A Modular Distributed IoT Service Discovery. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 448–454.

[13] Keyur K Patel and Sunil M Patel. 2016. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. *International Journal of Engineering Science and Computing* 6, 5 (2016).

[14] Henrique F. Santos, Rui L. Claro, Leonardo S. Rocha, and Miguel L. Pardal. 2020. STOP: a location spoofing resistant vehicle inspection system. In *The 19th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now)*. Bari, Italy.

[15] Zhichao Zhu and Guohong Cao. 2011. APPLAUS: A Privacy-Preserving Location Proof Updating System for Location-Based Services. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1889–1897.