TÉCNICO LISBOA

SureThing: User Device Location Certification

João Ricardo Pais Ferreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Dr. Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof Dr. José Luis Brinquete Borbinha
Supervisor: Prof. Dr. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Dr. Miguel Nuno Dias Alves Pupo Correia

November 2017

Success is no accident. It is hard work, perseverance,
learning, studying, sacrifice and most of all,
love of what you are doing or learning to do.

- Pelé

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus pais pelo seu apoio ao longo destes anos e por me terem dado tudo aquilo que precisava para completar esta jornada académica e pessoal. Um agradecimento ao meu irmão Luís que sempre foi uma inspiração para mim, tanto a nível pessoal como profissional. Agradeço ao resto da minha família, em especial aos meus avós.

Agradeço ainda o amor e apoio por parte da minha namorada, Mariana Raimundo, ao longo desta jornada. Percorremos este caminho juntos e estou realmente agradecido por isso. Obrigado ainda aos meus dois grandes amigos, Frederico e Rui, que permanecem comigo desde criança.

Obrigado a todos aqueles que estiveram envolvidos neste percurso académico. Em particular, quero agradecer aos meus colegas e amigos Ivo Pires, Raquel Cristovão e Pedro Lagarto pelos projetos em que trabalhámos juntos e por todas as ideias discutidas. Espero sinceramente poder voltar a trabalhar convosco no futuro.

Finalmente, gostaria de agradecer ao meu orientador Miguel Pardal pela sua orientação, paciência e sabedoria ao longo deste projeto. A sua ajuda, ideias e conselhos foram fundamentais para o desenvolvimento deste trabalho. Quero ainda deixar uma palavra de agradecimento ao meu colega Diogo Calado, com quem semanalmente partilhei a sala de reuniões sobre a tese.

A todos aqueles que estiveram e continuam ao meu lado - Obrigado.

Acknowledgments

I want to thank my parents for their support throughout the years and for giving me everything that I needed to complete this academic and personal journey. Thanks to my big brother Luís which has always been a great personal and professional inspiration for me. I also want to thank the rest of my family, in particular to my grandparents.

I am grateful for the love and support of my girlfriend, Mariana Raimundo, throughout this journey. We made this path together and I am really thankful for that. Thanks also to my two really good friends, Frederico and Rui, that remain with me since I was a child.

I also have to thank all the people that were involved in this academic quest. In particular, I want to thank my colleagues Ivo Pires, Raquel Cristovão and Pedro Lagarto for the projects that we made together and for all the ideas that we discussed. I sincerely hope to work you again in the future.

Finally, I would also like to acknowledge and thank my advisor Miguel Pardal for his guidance, patience and wisdom throughout this project. His help, ideas and advices were fundamental to the development of this work. I also want to give a word of acknowledgement to my colleague Diogo Calado, with whom I shared the thesis reunion room almost every week.

To each and every one of you – Thank you.

Resumo

As aplicações baseadas em contexto do mundo real têm vindo a aumentar de popularidade e tendem a ser cada vez mais utilizadas. Hoje em dia, grande parte da população transporta consigo dispositivos móveis com grande capacidade de processamento e com sensores que permitem capturar dados e obter informação. Um dos tipos de contexto mais usados no desenvolvimento destas aplicações é a *localização*, ou seja, a posição no mundo onde o utilizador da aplicação se encontra. Determinados serviços, como informação sobre os pontos de interesse mais próximos, apenas são oferecidos ao utilizador se este estiver numa certa localização. Contudo, para implementar serviços com um maior valor, como a venda de um produto ou a abertura de uma porta para um local de acesso restrito, é necessário verificar e garantir com um maior grau de certeza a presença do dispositivo do utilizador no local. Neste trabalho, é proposto o *SureThing*, uma solução de certificação de localização que permite emitir provas de que o dispositivo se encontra, de facto, num determinado local. O SureThing utiliza diferentes técnicas de estimação de localização e recorre a outros utilizadores do sistema como testemunhas para certificarem a presença de um dado utilizador e a credibilidade das medições de localização. Foi desenvolvido um protótipo do SureThing na plataforma Android, que é atualmente a plataforma mais usada para o desenvolvimento de aplicações móveis. O protótipo foi avaliado em relação aos tempos de resposta, à precisão das estimativas de localização e à viabilidade das trocas de provas entre utilizadores que não se conhecem previamente. Os resultados demonstram que a nossa solução é útil e viável em cenários práticos no mundo real.

Palavras-chave: Segurança Móvel, Aplicações baseadas em Contexto, Estimação de Localização, Prova de Localização, Internet das Coisas.

Abstract

Context-based mobile applications are rapidly gaining popularity. Nowadays, we carry in our pocket mobile devices with large processing capabilities and with sensors that capture data and obtain information about the real world. One of the most used types of context is *location*, which is the position in the world where the user of the application is in. Location-based applications caused the emergence of services that are offered to the users only when they are at specific locations. For example, a user can receive information about nearby points of interest. To implement services that are more valuable, like a product sale or opening a door to a closed room, it is necessary to verify the presence of the user's device in a way which can be reliably trusted by the service providers. In this work we propose *SureThing*, a location certification solution for services that need strong assurances about the physical location of a user. SureThing allows the creation of proofs that provide evidence that the user's device is at a claimed location. SureThing relies on different location estimation techniques and on other users of the system as witnesses that can testify to the presence of the user and to the credibility of the location measurements. A SureThing prototype was implemented in the Android platform which currently is the most used platform for mobile development. The prototype was evaluated regarding response times, accuracy of location estimations, and feasibility of proof exchanges between users who did not know each other beforehand. The results show that the solution is both useful and feasible in practical real-world scenarios.

Keywords: Mobile Security, Context-Awareness, Location Estimation, Location Proof, Internet of Things.

Contents

List of Tables . . . xvii

List of Figures . . . xix

List of Acronyms . . . xxi

1 Introduction **1**

1.1 Contributions . . . 2

1.2 Motivating Use Case . . . 3

1.3 Dissertation Outline . . . 3

2 Background **5**

2.1 Location Systems Properties . . . 5

2.1.1 Estimation Techniques . . . 5

2.1.2 Physical and Symbolic Location . . . 5

2.1.3 Absolute and Relative Location . . . 6

2.1.4 Scale . . . 6

2.2 Location Estimation . . . 6

2.2.1 Global Positioning System . . . 6

2.2.2 Wi-Fi . . . 7

2.2.3 Cellular Networks . . . 7

2.2.4 Bluetooth . . . 8

2.2.5 QR Codes . . . 9

2.2.6 Ambience Fingerprinting . . . 10

2.2.7 Hybrid Solutions . . . 10

2.2.8 Comparison . . . 11

3 Related Work **13**

3.1 Location Certification . . . 13

3.1.1 Distance-based Methods . . . 13

3.1.2 Location Proof Definition . . . 14

3.1.3 Proof-Based Methods . . . 15

3.1.4 Location Provenance Proofing . . . 18

3.2 Summary . . . 18

4 SureThing 21
4.1 SureThing Entities . . . 22
4.1.1 Choreography . . . 22
4.2 Location Proof Techniques . . . 23
4.2.1 Setup Phase . . . 23
4.2.2 Verification Phase . . . 24
4.3 Witness Models . . . 26
4.3.1 Master Witness Model . . . 26
4.3.2 Mobile Witness Model . . . 27
4.3.3 Self Witness Model . . . 27
4.3.4 Comparison of Witness Models . . . 27
4.4 Collusion Avoidance Mechanisms . . . 28
4.5 Communication Protocol . . . 29
4.5.1 Proof Demands . . . 29
4.5.2 Proof Requests . . . 30
4.6 Summary . . . 31

5 Prototype Implementation 33
5.1 Project Structure . . . 33
5.2 Mobile Application Library . . . 33
5.2.1 Initialization . . . 34
5.2.2 Proof Processing . . . 36
5.3 Verification and Certification Services . . . 37
5.3.1 Verifier . . . 37
5.3.2 Certification Authority . . . 39
5.4 Example Application . . . 39
5.5 Summary . . . 40

6 Evaluation 43
6.1 Location Estimation Accuracy . . . 43
6.1.1 Geo and Wi-Fi Fingerprinting . . . 43
6.1.2 Beacon . . . 45
6.2 Time Evaluation . . . 45
6.2.1 Location Estimation Time Evaluation . . . 46
6.2.2 Time Evaluation of Witness Models . . . 47
6.3 Discussion . . . 48
6.4 Collusion Avoidance Simulation . . . 49
6.4.1 Simulation Setup . . . 49
6.4.2 Simulation Results . . . 50
6.4.3 Simulation Summary . . . 52

7 Conclusion **53**

Bibliography **55**

List of Tables

2.1 Comparison between location estimation techniques. 12

3.1 Comparison between location certification systems. 19
3.2 Comparison between Witness-based proofing systems. 19

4.1 Proof Techniques Fields and Setup Phase. 24
4.2 Comparison between Witness Models. 28

6.1 Location estimation accuracy with beacons. 45

List of Figures

1.1 Smart Place example where a door is automatically open. 1

3.1 Messages exchanged between the Verifier and the Prover during the *Echo* protocol. . . . 14
3.2 Hexagonal room where a single verifier would not have full coverage. 14
3.3 Witness-Based Location Proof Protocol. 16

4.1 Communication between entities in SureThing. 22
4.2 Example of a list of Proof Demands sent by the Verifier. 30

5.1 Project Structure of SureThing. 34
5.2 Class diagram of the classes used in the initialization. 35
5.3 Class diagram with the main classes used by both Prover and Witness. 36
5.4 Screens of the Shopping center's application to give discounts to certified users. 40

6.1 Shopping Center building. 44
6.2 Percentage of successful readings for Geo and Wi-Fi Fingerprinting. 45
6.3 Test scenario with beacons. 46
6.4 Time to get Location Data. 47
6.5 Time spent from requesting proof demand until the Verifier informing the Prover about the proof's acceptance or rejection (Master and Mobile Witness Model). 48
6.6 Shopping center simulated in *Netlogo*. 50
6.7 Acceptance and Denial rate of the Proof Batches. 51
6.8 Maximum quantity of Proof Batches accepted when the Prover is always repeating the same Witnesses. 51

Acronyms

ANLP	Android Network Location Provider
AGPS	Assisted Global Positioning System
AP	Access Point
BLE	Bluetooth Low Energy
CA	Certification Authority
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LP	Location Proof
OS	Operating System
QR	Quick Response
REST	Representation State Transfer
RSSI	Received Signal Strength Indicator
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WLAN	Wireless Local Area Network
WoT	Web of Things
XML	Extensible Markup Language

Chapter 1

Introduction

The use of multiple sensors and actuators, embedded in the environment and connected to computers, enables *context-aware* systems that gather information about the real world for use in virtual systems. In fact, the main goal of the Internet of Things (IoT) [1] is to connect and integrate the real and virtual worlds to allow new and useful applications. If a certain place can sense and act in the real world, it can be called as a Smart Place [2, 3]. Figure 1.1 represents a Smart Place, where the access control is automatic. In this example, a face recognition sensor is responsible for sensing the environment and identifying the user that wants to open a certain door. After the verification of the user, an actuator will act in the environment, opening the room door for the user. This room can be considered a Smart Place, since those sensors and actuators understood the context and acted in a physical place in the real world.

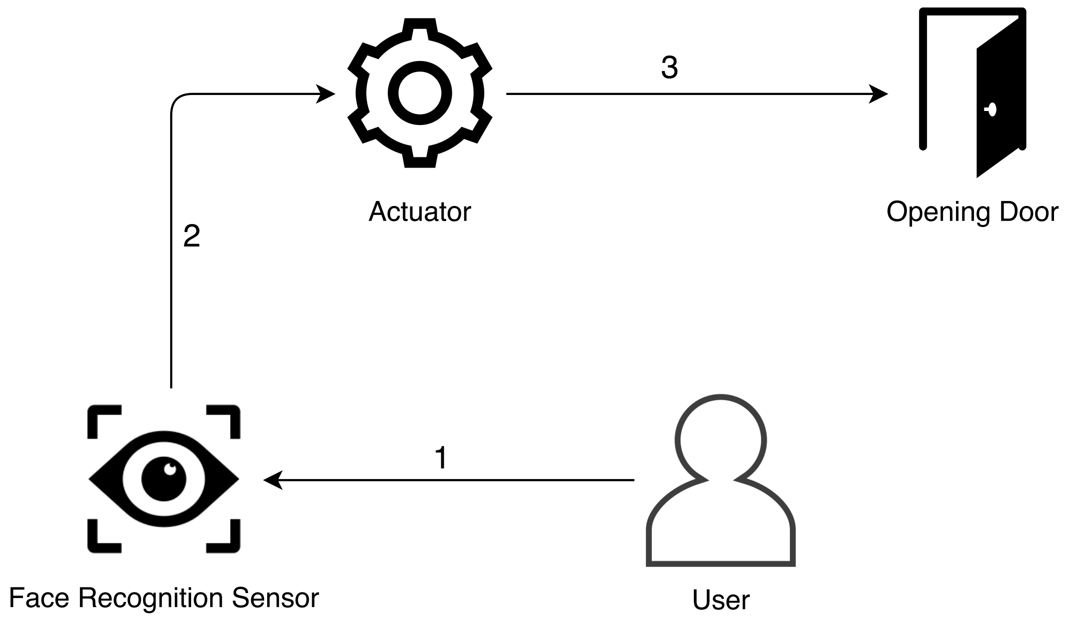


Figure 1.1: Smart Place example where a door is automatically open.

IoT is closely related to *Ubiquitous Computing*, as its main goal is not only to permit that computers can sense the environment without human interaction and act accordingly with it [4], but also to make technology invisible in everyday life. In the previous example, the regular user would not be aware of the presence of a computer, only of the door opening automatically. This also means that solutions should be available anytime and anywhere.

Even though IoT stands for *Internet* of Things, nowadays most scenarios most closely resemble a group of *Intranets* of Things that cannot really communicate with each other, because there are multiple communication protocols, due to different device capabilities and physical interfaces. The *Web of Things (WoT)* has been proposed with the goal of trying to minimize this problem [5]. WoT assumes an existing infrastructure in the Cloud, where the different systems communicate. The communication is between devices and their cloud infrastructure and then between clouds. WoT allows developers to worry mostly about their application logic instead of having to deal with IoT protocols or specific device characteristics. All of that is now managed by WoT using well known tools, like, for example, the Hypertext Transfer Protocol (HTTP). WoT solutions provide a service layer and allow the orchestration of services using that layer without regarding the protocols that are used by each device, whereas in IoT, different approaches have to be taken do deal with different devices. Pahl et al. [2] stated that the current heterogeneity of devices hinders their management, because there is no abstraction to deal with them in a simple way. They propose that the best way to deal with this problem is to have a Smart Place that is controlled by software. This allows a greater level of flexibility in the Smart Place management, because it would be easier to change or add new functionalities to it [1] .

As the IoT vision comes closer to reality, context-aware applications will play an increasingly relevant role in society. Security policies will have to be put in place so that the systems are securely used. Location is one of the most used types of context [6] when developing context-based applications and location information will be required to make policy decisions. Smartphones are the devices that can better take advantage of the environment awareness enabled by the IoT because of their numerous sensors, Internet connection and, most important, because they have a close bound with the human user. When observing the location of a user by using the smartphone, it is possible to provide a personalized service [7]. In a location-based smartphone application, a service of *value* should only be accessed if properly authorized. As an example, consider a service that provides access to products in a vending machine. If the user's device is certifiably at a particular location, at a given time and the payment was made, then permission to retrieve the product can be granted. *Trusted* attributes are required to make secure authorization decisions. A location claim made by the user is not enough to give an intended privilege. Location data needs to be collected and verified so that service providers can have strong assurances about the location.

1.1 Contributions

This dissertation makes the following main contributions. First, we propose the design and implementation of a location proofing solution, named *SureThing*, to be used in smartphone mobile applications. We use a *witness-based* approach to generate location proofs, where the user that needs the proof has to request it from different entities, known as *witnesses*. We propose different witness models,

[1] Pahl et al. [2] use the term Smart Space rather than Smart Place, as we do. Nevertheless, the concept is similar.

with different assumptions regarding the trustiness of each witness. Second, we present diverse location proof techniques that estimate the location of a user by using different technologies together with methods for detecting false location claims. SureThing is a solution that integrates different location estimation techniques to provide higher flexibility to developers. Finally, we evaluate and analyze our system functionality and behavior, to show that it is suitable for real-world applications.

1.2 Motivating Use Case

The following scenario illustrates the practical use of SureThing. Consider a state-of-the-art *shopping center* that wants to improve its customers experience by rewarding regular visitors with special discounts. Alice is a customer that opted-in to the program and is running the loyalty application on her smartphone. Alice arrives at the food court of the shopping center and is deciding where to have dinner. Her presence is detected and her smartphone is asked to prove her location. The history of Alice's presences at the location is also retrieved. Alice's device collects location measurements to make a location claim and, at the same time, requests a location proof to a nearby witness. Bob, another user running the application on his smartphone, was also at the food court and was selected as witness. Alice presents her claim and the proof provided by Bob to the shopping center. The proof is verified, everything is OK, and as a result, Alice receives discounts for restaurants. She can now happily go to one of her favorite places at a very special price. The shopping center has just rewarded a verified regular customer that will, hopefully, come back more often.

SureThing was used by the loyalty application to abstract the different location estimation techniques used and to manage the cooperation between users to provide the witness proofs. SureThing can also be used to provide different levels of location precision according to the needs of the application. Proofs can have a higher location precision, for example, proving that Alice is at a specific store, or lower precision, just to identity a shopping zone like the food court.

We will use this shopping center as a running example to illustrate the design and operation of SureThing in the remainder of this dissertation.

1.3 Dissertation Outline

The rest of this document is organized as follows. Chapter 2 presents the most important concepts related with location and the most relevant techniques for location estimation. Chapter 3 introduces relevant techniques for location proofing that were used in previous works. Chapter 4 introduces SureThing's design and features. Chapter 5 details the implementation of our prototype. Chapter 6 presents the evaluation conducted to validate the design and implementation of our system. Finally, Chapter 7 presents our conclusions and future work.

Chapter 2

Background

In this Chapter, we introduce some properties of location systems and location estimation techniques that will be relevant in the development of SureThing. Section 2.1 presents some of the most important location concepts. In Section 2.2 we present some location estimation techniques that allow location readings to be obtained.

2.1 Location Systems Properties

Nowadays, location is one of the most used types of context when building mobile applications. Before starting the development of these type of applications is important to understand some properties related with location. Hightower et al. [8] created a list of location systems properties and we will highlight the following: *techniques* to obtain location; *physical and symbolic* location; *absolute and relative* location and *scale*.

2.1.1 Estimation Techniques

There are three different techniques for calculating the location of a user. The techniques are not mutually exclusive and can be used together:

- **Triangulation** - Use distance or angle measurements between known points;
- **Proximity** - Location given by being close to a given point;
- **Scene Analysis** - Examine the environment and be able to infer location from it.

2.1.2 Physical and Symbolic Location

Two different types of location can be provided by a location-based system:

- **Physical** - Provides physical positioning (e.g. latitude and longitude coordinates);
- **Symbolic** - Abstract and logical notions of location (e.g. in the university; on a bus).

As before, the two properties are not mutually exclusive. Physical locations can be interpreted and moved to a symbolic representation. The same physical location can also be interpreted as two different symbolic locations by different applications.

2.1.3 Absolute and Relative Location

A location can also be classified according to its relativeness. There are two ways of classifying it:

- **Absolute** - A common shared reference system is used to understand the location (e.g. all systems with Global Positioning System (GPS) use latitude and longitude);
- **Relative** - Each object or user to be located has its own reference system (e.g. Alice is near Bob; Alice is to the left of Bob).

An absolute location can be transformed into a relative one if there is a relative point that we can use to describe the location. The opposite is also possible since a relative location can be turned into an absolute one, but only if we know the absolute locations of the relative points that we are using to describe the original location.

2.1.4 Scale

The scale of a location-based system takes into account two measurements: the coverage area per unit of infrastructure installed and the capability of the system to handle an increasing number of objects. The cost of installation of further infrastructure and the middleware complexity when working with high amounts of data are two of the obstacles for scaling up location-based systems.

2.2 Location Estimation

In this Section we will present location estimation techniques that make the system aware of where the user is. We present the following techniques for determining the location of a user: *GPS*; *Wi-Fi*; *Cellular Networks*; *Bluetooth*; *Quick Response (QR) Code*; *Ambience Fingerprinting* and *Hybrid* mechanisms.

2.2.1 Global Positioning System

GPS is one of the most well known and widely used location mechanisms. It achieves measures with high accuracy outdoors, by using a triangulation process between signals that the device receives from various satellites [9]. However, GPS signals can be reflected or scattered when they hit something solid, like buildings or tunnels [9]. Because of this, GPS cannot obtain good results when the user is indoors, where signals are 10 to 100 times weaker than outside or do not exist at all. Even if the device can acquire some satellite, signals suffer from multi-path effects, resulting in an incorrect location that misleads the receiver [10].

Assisted Global Positioning System (AGPS) is an extension to the GPS technology and it helps to minimize some of the problems stated before. In areas with large buildings or indoors, where it is difficult to acquire a signal, AGPS receives these signals through different channels, via cellular network towers or base stations who also have GPS receivers that are constantly getting data from the satellites. Therefore, the device of a user can receive a location fix sooner than with regular GPS [10], because the communication is done with closer entities. This also reduces the processing done by the device of the user and therefore it reduces the device battery consumption.

2.2.2 Wi-Fi

Nowadays, almost every domestic or commercial place has its own Wireless Local Area Network (WLAN) allowing that two devices communicate with each other in a wireless way. The user can move around and still be connected to the network. With this in mind, some Wi-Fi based positioning techniques were developed. One of the most well known is called Wi-Fi fingerprint [11, 12]. Consider the case where we have a shopping center that has full Wi-Fi coverage by having multiple Access Points (APs) spread through out its area. The main goal of Wi-Fi fingerprinting is to construct a map of Wi-Fi signal strengths obtained from the different APs in a given location in the shopping mall. When this map is fully constructed and the user enters in the WLAN, it is possible to measure the signal strength in the user's device and then compare this measurement with the values that are stored in the map. Since there should not exist two equal values saved in the map, we can assume that the user is in the location with the most similar signal fingerprint [11]. This technique has two main drawbacks: signal strength fluctuation (a physical change in the environment may cause the need for a modification in the previous map and that change has to be done manually) and the fact of requiring a heavy deployment of APs for full coverage of the area and better fingerprinting of Wi-Fi signal strength [12, 13]. This is probably only practical in dense spaces, like the mentioned shopping mall.

The previous disadvantages were addressed in systems like SAIL, introduced by Mariakakis et al. [13]. This system focused in creating Wi-Fi location techniques based on readings from only one AP. SAIL estimates the distance between the user and an AP using the propagation delay of the signal that travels across the two entities. After this first calculation, it employees a method called dead-reckoning [14]. Dead-reckoning is a technique used to compute the displacement of a user between two locations by using his smartphone sensors, like gyroscope and accelerometer.

2.2.3 Cellular Networks

As in Wi-Fi location systems, it is also possible to use cellular networks to create a fingerprint for each location that we want to identify. This fingerprint is created with the received signal strength of each cell tower. In an offline phase, as in Wi-Fi fingerprinting, the fingerprints for each identifiable location should be obtained and saved. This allows the comparison between the fingerprints provided by the user's smartphone and the ones that were previously collected.

Ibrahim et al. [15] developed *CellSense*, a system that uses the obtained Received Signal Strength

Indicator (RSSI) from the different cell towers to construct a fingerprint. The main advantage of this method is the independence from infrastructure as in GPS solutions, since that the necessary infrastructure to support mobile communications is already developed. However, this method has a low accuracy by itself and location-based systems based on cellular networks are frequently combined with GPS and Wi-Fi location techniques to provide a better positioning accuracy.

2.2.4 Bluetooth

Smartphones have the ability to use Bluetooth to communicate with other devices (a Bluetooth beacon or even another smartphone). Bluetooth can also be used to determine where the user is located. Bluetooth localization systems are mainly based on having Bluetooth beacons spread over the area we want to cover. If a user can detect one of these beacons it is possible to have an approximation of where the user is, since the Bluetooth range is quite limited (typically less than 10 meters). Consider the example from Section 1.2, where we have a large shopping center and we want to identify the store in which the user is. If, for example, store *A* has its own Bluetooth beacon (device that sends signals over Bluetooth technology), it is good to assume that if the user can communicate with the beacon from store *A*, it is because he is there. The strength of the signal that the user receives from the beacon can also be used to assure that the user is at store *A*. However, there are a few drawbacks in this type of implementation. Due to its small area coverage, objects (and therefore users) are only detected in a limited range. So, if we want to cover a very large area (for example, the whole shopping center) we will need multiple beacons, which many times is not feasible, due to hardware or installation costs [16].

When implementing location-based applications using Bluetooth, it is important that the beacons are placed in locations that allow identifying a certain area with a good level of certainty. For example, if a user can detect the same beacon from two different rooms, it is difficult to know in which room the user is. Although it is not always easy to determine where to place the beacons, Baniukevic et al. [16] presented an algorithm for selecting the ideal positions for deploying these beacons, since these have a big impact in the final result.

One example of a system that mainly uses Bluetooth location techniques is BIPS, introduced by Anastasi et al. [17]. The main goal of BIPS was to cover an entire building, offering a service that allows a given user to know where another user is and to calculate the shortest path to him. This is done by giving each room its own Bluetooth cell. A room is defined as a space that fits into a circle of 10 meters, since it is the maximum range of Bluetooth. The Bluetooth cells were also connected via Ethernet to a central server that would retain the information about the location of each user. The main idea behind this system is that every workstation where the cells are has to discover whose mobile devices are in the coverage area. When some device is discovered, this workstation sends to the central server the most current location of the user that it just found out. So, when user A wants to discover where user B is, he can contact the central server, in order to know the most recent location of B. Although it shows that Bluetooth-based solutions can do a good job in discovering where some user is, the main drawback is that if the area we want to cover is very large, we need to have multiple Bluetooth cells across the building to have full coverage.

In the last few years, a new version of Bluetooth has emerged. This variant is called *Bluetooth Low Energy (BLE)* and it is more power efficient than the original Bluetooth [18]. BLE is mainly used in situations where the amount of data to be transferred is low. Due to its short range of communication, BLE is well suited for detecting location by proximity.

The *iBeacon* protocol [19] is an example of the use of BLE to determine the location of a user. The key device of this protocol is a BLE beacon. This device is constantly emitting a sequence of bytes, that can be used by the applications to derive the place where the user is. The bytes emitted by the beacon represent three different values: an *Universally Unique Identifier (UUID)*, a *major* and a *minor*. The UUID is composed by 16 bytes and it identifies the organization to whom the beacon belongs (e.g. Shopping Center *SC*). All the beacons belonging to *SC* will have the same UUID. The major value has 2 bytes and is the group identifier of the beacon (e.g. Book Shop *BS*). All the beacons with the same major value will belong to the same store *BS*. Finally, the minor value clearly identifies one beacon and it is also composed by two bytes. If the location of the beacon is known we can derive the location of the user. Two beacons cannot have the same UUID, major and minor value. Their minor value can be the same, if they do not belong to the same group. Location-based systems using these iBeacon protocols have been developed [3]. The logical location of each beacon is known so when a user contacts a beacon and detects its emitted values, it is possible to understand where the user is located. The main disadvantage of this type of systems continues to be the same as in the original Bluetooth: the more area we want to cover, the more beacons we need to install. Also, a user's smartphone should not be always looking for beacons, because this procedure will consume its battery resources. The moment to start looking for beacons and monitoring the space is also a very important decision when developing location-based applications using BLE beacons.

2.2.5 QR Codes

QR code is a two dimensional bar code that can encode data that can be optically read later. In the past years, QR codes were adopted in many different systems, for their fast readability and for having a greater storage capacity than normal bar codes [2]. Montenegro et al. [20] created QR-Maps, a system that uses QR Codes in order to provide a user's location. It uses four main elements: a QR code, with a text to be decoded; a smartphone, only with connection to the Internet and capability to decode QR codes; a location server where the smartphone does its request and a map server that simply contains custom maps using the Google Maps API. This system was tested on a university campus and various QR codes were spread throughout the area. The process begins when a user, with his smartphone, scans a QR code that contains a simple information, like for example the word *BuildingA5*. This word means that this QR code is located in Building *A* of the campus and the number 5 is a reference to the position inside that building (the floor, for example). The user sends this data to the location server, that responds with a specific Uniform Resource Locator (URL) for the given request. This URL is passed to

[2]QR codes: `http://www.denso-wave.com/qrcode/qrstandard-e.html`

the map server, that responds with the desired custom indoor map. After knowing his location, the user can use QR-Maps to check the rest of the map and get directions to other room.

The main advantage of this solution is its simplicity. One of the main drawbacks is that the process of locating the user only begins when he scans the QR code. For example, in the BIPS system, presented in Section 2.2.4, if we had Bluetooth enabled in our smartphone, the system will determine our location without the need for the user to interact with the system, which means that the system behaves in a more automatic way.

2.2.6 Ambience Fingerprinting

Ambience Fingerprinting is a location technique that focuses in recognizing logical locations (for example, that a given user is in store A) instead of physical ones (the store coordinates). With smartphones having numerous sensor capabilities, it is possible to use some of them to derive the logical location of a user.

Azizyan et al. [21] presented *SurroundSense*, a system that uses ambience fingerprinting to derive where the user is located. The main idea behind the system is to use data obtained from sensors, like sound, light and color (by using the phone's microphone and camera). By using the device accelerometer it also takes into account the user motion within the place, which can also help to identify it. If we think in two different stores, a restaurant and a grocery shop, the motion pattern of a user is quite distinct. In the first, the user stays seated most of the time while in the latter a user walks around the place. After getting all the different types of data, the user forwards it to a central server. The final goal of *SurroundSense* is therefore to construct a fingerprint map for all of the places that it should be able to identify. By combining all previous data it is possible to reach a unique identifier for each place. One of the main advantages of this type of solution is that it does not require any type of previous installed infrastructure like, for example, Bluetooth-based solutions. The system has two phases: the first one is the *filtering* phase, where it discards fingerprints that are not similar to the one that the user sent. The second phase is known as *matching* phase. The system analyzes the fingerprint sent by the user and returns the most similar fingerprint stored in the database. The first fingerprint on the list is then returned to the user, yielding its logical location. This approach for measuring the user's location is independent from the infrastructure (although it benefits from, for example, having a Wi-Fi connection). Although, it also has some disadvantages, for example, for two very similar places it is possible that the user is misled by getting a wrong result. Also, if we want to add *N* more places to that system, multiple readings should be done in order to capture the different fingerprints and save them into the central server database. The fingerprint map needs to account for variations from day to day or even from hour to hour, which makes it more difficult to maintain a well populated fingerprint.

2.2.7 Hybrid Solutions

As an alternative method for determining the location of a user, it is also possible to create an hybrid solution that joins different methods. Baniukevic et al. [16] presented an hybrid mechanism that consists in both the use of Wi-Fi and Bluetooth techniques to guarantee better results when calculating where the user is located. Wi-Fi fingerprinting is used in this hybrid system, measuring the Wi-Fi signal strength in the area we want to cover. Bluetooth beacons are located at particular places to give a correct user's location where Wi-Fi measurements could get wrong results by themselves. It is possible that even two far way positions share similar Wi-Fi fingerprints, which can lead to untrusted results. By adjusting Bluetooth beacons position, it is possible to separate these similar positions into different partitions, therefore putting their Wi-Fi fingerprints into distinct radio map zones. So, with a limited number *n* of Bluetooth beacon it is possible to create *n* disjoint partitions. Following this, the original map will also be divided in *n* smaller maps, each one belonging to a Bluetooth partition. When the user detects a Bluetooth beacon, his location is given only by the beacon itself, which guarantees a good positioning accuracy. When the user leaves the partition of that Bluetooth beacon, the system has to decide the current partitions where the user can be. After that, it only checks for Wi-Fi fingerprints on those given partitions, which eliminates the problem where two faraway positions had similar fingerprints (because the other partition was not chosen when the user changed his partition). This merging of both Wi-Fi and Bluetooth in the same solution leads to a technique that can join the best of the two methods. While it continues to be suitable for indoor location, the positioning accuracy is high and it has a large coverage area even with few Bluetooth beacons. Estimating the user's location using only Bluetooth when the user is connected to a beacon is simpler and takes less energy than using Wi-Fi. However, when the user is changing from Bluetooth partitions, Wi-Fi based location techniques allow to only need a few Bluetooth beacons, provided that the latter are correctly placed through out the area, maximizing the area covered by Bluetooth.

The Android Network Location Provider (ANLP)[3] is also a hybrid network-based location system that uses both cell tower and Wi-Fi access point information in order to better determine the location of a user. It uses both technologies to construct a fingerprint of the place where the user is. It can be used in Android applications. This method responds faster and uses less battery than GPS solutions and achieves more precise results in areas with more Wi-Fi access points because the fingerprint will be more precise.

2.2.8 Comparison

After presenting the main techniques to determine where the user is located, it is possible to summarize the results in Table 2.1. We think that all the location estimation systems presented have some advantages and disadvantages. GPS is free from the installation of further infrastructure but it is not capable of guaranteeing high accuracy results indoors. Wi-Fi infrastructure is now adopted almost everywhere and it is a good solution for indoor cases. However there are problems, for example, regarding signal strength fluctuation if we are working with Wi-Fi fingerprints in dynamic and changing environments. Cellular Networks are also a valid option since smartphones are connected to mobile networks. This method can provide a fingerprint for the place as Wi-Fi does and is independent from the installation of new infrastructure. However, it has a lower accuracy than Wi-Fi based solutions. Bluetooth and QR

[3]Android Network Location Provider (ANLP): `https://developer.android.com/guide/topics/location/strategies.html`

	GPS	Wi-Fi	Cellular Networks	Bluetooth	QR Codes	Ambience Fingerprint
Indoor Suitable		✓		✓	✓	✓
Technique	Triangulation	Triangulation, Scene Analysis	Triangulation, Scene Analysis	Proximity	Proximity	Scene Analysis
Physical	✓					
Symbolic		✓	✓	✓	✓	✓
Absolute	✓					
Relative		✓	✓	✓	✓	✓
Coverage	Anywhere with GPS signal	50m per AP	Anywhere with mobile coverage	10m per beacon	Need for user scanning	Undefined [4]
Infrastructure Dependent	[5]	✓[6]	[5]	✓	✓	

Table 2.1: Comparison between location estimation techniques.

codes are the most accurate positioning systems, since the user has to be in the range of the Bluetooth beacons or close to the QR codes to scan them. However, both of these solutions involve the addition of further infrastructure in the environment, augmenting the installation costs of the system. Ambience Fingerprint is independent from the infrastructure but it is hard to maintain a well populated and updated fingerprint map. As we can see, all location estimation techniques have some strong and weak points, and can adapt differently to distinct situations.

[4]The application developer has to populate a database with ambient information about the places that he wants to identify. The coverage area depends on how many places the developer identified.

[5]Location systems depending on GPS and cellular networks depend on external infrastructure. They do not depend at any infrastructure from the areas where we want to prove a user's location.

[6]Location systems based on Wi-Fi need installed infrastructure in the place but this infrastructure is mainly adopted.

Chapter 3

Related Work

In this Chapter we will present relevant location certification techniques to guarantee that the locations obtained are valid and certified. Section 3.1 goes through each of these techniques and it introduces new concepts associated with them. Section 3.2 makes a comparison between the presented techniques.

3.1 Location Certification

In Section 2.2 we discussed various techniques to determine where a user is located. Here, our main objective is to find methods that enable a user to show to another entity that he really is in the place where he claims to be. If we consider a system that benefits users based in their location (for example, more discounts for regular customers), malicious users will have a motivation to lie about their location [22]. In the current Section we will analyze some systems with the main goal of proofing the user's location and show what are their advantages and disadvantages.

3.1.1 Distance-based Methods

Early approaches to provide location verification were many times based on measuring the latency between two different entities. One example of this approach is the *Echo* protocol, proposed by Sastry et al. [23]. This protocol was designed to give users the ability to prove that they are in a specific location, without having to deal with complicated problems related to infrastructures or to the distribution of cryptographic keys to every user that wants to use a specific service.

Echo is based on the time that a Verifier V and a Prover P take to exchange messages between them. Figure 3.1 presents the messages exchanged between V and P during the protocol. The simplest case to understand this protocol is the following: consider that we have a circular room and we have V in the center. V will then create a circular region R, that covers up all the room area. A prover P comes to the room and wants to show to V that he is really in there, so he sends to V his location L. When V receives this message, it will respond to P with a random generated nonce N. After receiving N, P will simply re-transmit it to V. The latter will accept that L is inside R only if the time between sending the nonce to P and receiving his answer is less or equal to a given calculated value. So, it is guaranteed

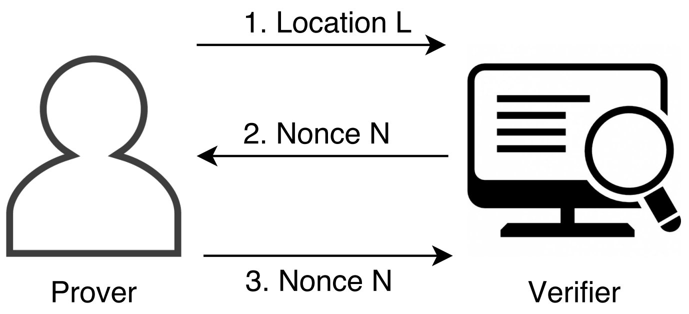


Figure 3.1: Messages exchanged between the Verifier and the Prover during the *Echo* protocol.

that *P* is in *R* if the time that he takes to exchange messages with *V* does not exceed the defined limit. If it does, it means that *P* is more than *d(V,L)* meters from *V* (therefore, outside the room). The nonce exchanged between *V* and *P* prevents the latter from anticipating his answer, because he cannot guess the correct value of *N*. However, processing delay has to be taken into account. If we consider some delay for computing and checking the exchanged nonce between *V* and *P*, *V* will need to shrink the area *R*. The shrinkage measure is directly proportional to the processing delay that has to be previously known. By doing this, the protocol can reject some correct claims (for example, from the border of the circle), but it will continue to never accept malicious or wrong claims.

Another issue of this solution is related to the area that the protocol wants to cover. In the previous example, we presented a simple case, where a single verifier can occupy all of the circular room area. If we have an irregular shaped room, like in Figure 3.2, multiple verifiers would be needed to have full coverage. Each of the verifiers will compute its maximum circle inside the room and if well distributed, the conjunction of circles from all verifiers can fill up the entire room or at least, a big share of it. As illustrated in Figure 3.2, one verifier would not be enough to have full coverage of the hexagonal room. The main advantage of this kind of methods is their simplicity, because they do not need to work with cryptographic mechanisms or to rely on pre-available infrastructure. However, like presented before, not all correct claims are accepted, depending on how well the space is covered. This solution is also not scalable, because for very large spaces we will need multiple verifiers.

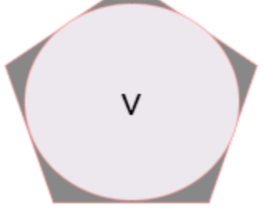


Figure 3.2: Hexagonal room where a single verifier would not have full coverage.

3.1.2 Location Proof Definition

The definition for *Location Proof (LP)* in ubiquitous systems was first given by Saroiu et al. [22]. The main idea behind location proofs is that they should be a piece of information that ensures that a user was at a given place, at a given time. If the user presents that LP to an external service, the latter can have more assurance about the verity of the user's claim. According to [22], a location proof should have five main fields:

- **Issuer** - Entity who certifies the user's presence in a given place. Identified by the entity's public key;
- **Recipient** - User's device. Identified by its own public key;
- **Timestamp** - When the LP was generated. With this field it is possible to know not only if a user was in a given place, but also when he was. If this field did not exist, a user that entered one time in a certain place would forever have a LP for that specific location;
- **Location** - Identify the user's location, for example with latitude and longitude coordinates;
- **Signature** - Signature of the LP done by the issuer, to guarantee that it is not possible to change the content of the LP.

We can then define a LP as the following tuple: [*I, R, T, L, S*] where each letter represents one of the fields above.

3.1.3 Proof-Based Methods

Many location proofing systems were built based on the definition of LP given in the previous Section. In the following Section we introduce some works by authors who took this definition and used it or extend it. Usually, there are five main entities in location proofing systems:

- **Prover (*P*)** - Node that wants to prove its location, by gathering LPs from its neighbors;
- **Witness (*W*)** - Node that agrees to give a LP to the Prover. It will generate a LP and send it to the Prover that requested it. It can have an incentive for making valid assertions, like for example, to gain benefits from a service provider [24];
- **Database Server (*DS*)** - database that stores LPs that it receives from the Prover for later retrieval;
- ***Certification Authority (CA)*** - It can link pseudonyms with real identities and it certifies the public keys of the users of the system. It is assumed to be trustful;
- **Verifier (*V)*** - Service provider that can ask for a LP of one specific user.

The most common problems in location proofing systems based on Witnesses are known as *collusion attacks* [24, 25, 26, 27, 28]. These attacks consist in two nodes or more nodes working together in order to generate false location proofs. If two users combine to deceive the system and say that they are in location *X*, when in fact they are in location *Y*, the Verifier needs to have a methods to check the verity level of that LP.

We now introduce and compare the following location-proofing systems: *APPLAUS*, *CREPUSCOLO* and *"Who, When and Where"* [7]. These systems use Witnesses to endorse the location proofs provided to the Verifier. Each system has different approaches to deal with collusion attacks.

APPLAUS

Zhu et al. [25] created one location proofing system known as *APPLAUS*. Its design and communication protocol has many similarities with the design presented in Figure 3.3. The Prover starts the protocol by requesting location proofs to nearby Witnesses via Bluetooth. This technology was adopted to support the communication between users due to its short communication range. Each witness has a certain probability of accepting a request and answer it. If it decides to respond, as *Witness 1* and *3* in Figure 3.3, the Witness will send a LP to *P*. This proof usually contains the fields presented in Section 3.1.2. The proof will be forwarded by *P* to a database for later retrieval. By doing this, *V* can do the location verification step in the moment that it finds appropriate. The CA mediates the communication between the *DS* and *V* to guarantee the anonymity of the users. A certified Verifier can ask for location proofs to the database where proofs are saved. By using pseudonyms for each mobile device, *APPLAUS* guarantees that the user's privacy is protected, because it is not possible to link a LP directly to its owner. It is assumed that before using this system, mobile devices established contact with a CA that generates a certificate for the user's public key.

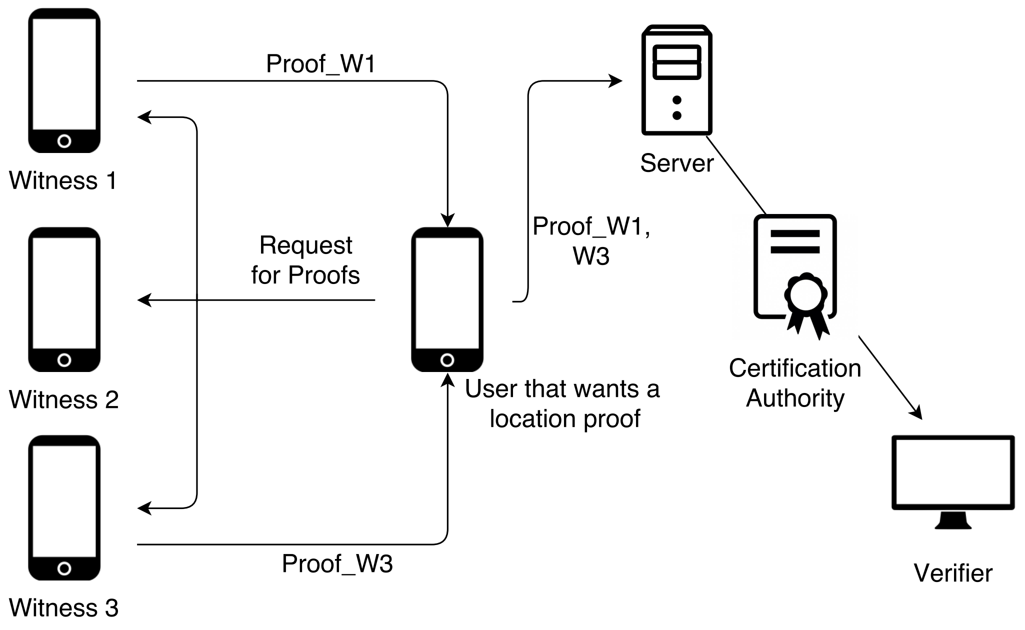


Figure 3.3: Witness-Based Location Proof Protocol.

One of the main advantages of *APPLAUS* is its ability to work without depending on the location infrastructure. By using a peer-to-peer approach between devices, a user's device only has to find some other device that can give it a LP. The only step of the process where network connection is needed is when the LP is sent to the location proof Database Server. However, the system cannot always deal with

[7]Khan et al. [24] did not give a name to their location proof assertion protocol, so we identify their work with a reference to the authors.

collusion attacks. In *APPLAUS* system, the *DS* checks if there are other LPs generated from the same place, by any other pseudonyms as witnesses. If there are, but those pseudonyms did not generate any proof to A or B, the latter are assumed as possibly colluders and the generated LP is marked accordingly. *APPLAUS* also relies on the number of users in the site. It has the information of how many users are at a particular site which can be used to find if a user is lying about not finding enough different witnesses.

CREPUSCOLO

With collusion attacks being one the main problems of *APPLAUS*, a new solution was given in the *CREPUSCOLO* system, introduced by Canlar et al. [26], so it can deal with more elaborate collusion attacks. *CREPUSCOLO* also focuses on a peer-to-peer method like *APPLAUS*, but it adds a new component to the system: a trusted Token Provider *TP*, that is responsible to generate a token *T* that will provide an endorsement to the LP. When the prover sends a broadcast to the witnesses asking for a LP, now it also sends a message to the *TP*, asking for a token. In the verification phase, an extra step will be done by the verifier *V*. Alongside with the LPs that it receives about the required user, it also receives the generated tokens for that user. Now *V* can accept or deny the location claim based on the LPs that it received but also from the tokens generated by *TP*. This extra step on the verification stage makes possible to deal with more sophisticated collusion attacks, like for example wormhole attacks [8]. An example of it will be two malicious provers, *P1* and *P2*, that try to collude to prove, for example, that *P1* is in the same location as *P2*, when in fact he is not. *P1* sends his information to *P2* and then *P2* broadcasts the received message, with *P1's* pseudonym. The witnesses have no idea about who is asking for a location proof and they will grant it. However, if a *TP* is on range with *P2* it will generate a new token, that *P2* will send to the location proof server. The idea of a token involving the prover can be, for example, a surveillance camera that takes a picture of the prover in that specific location, at a given time. When a Verifier *V* wants to check if the location claim of prover *P1* is true, it will receive the token that will not show any proof of the presence of *P1* [9]. Although LPs seem legit, the token will make *V* deny the location claim. This system extended the idea behind *APPLAUS* [25], adding the Token Provider to deal with more complex attacks. By having those *TPs* well spread through out the area we want to cover, it is possible to achieve some good results at such attacks. However, if the Prover cannot reach any *TP*, the system will not show any improvement.

"Who, When and Where"

Khan et al. [24] [10] introduced a new method for creating a collusion resistant protocol. Like the systems presented before, it also focuses on witnesses, that can prove that a user was at a given place, at a given time. However, in this work, another entity is added to the system. This entity is the Location Authority (*LA*) that is responsible for giving LPs for a particular area. They also consider the hypothesis

[8]In a wormhole attack, an attacker tunnels packets received at one point of the network to another point. The attacker can later replay those packets from that second point.

[9]Canlar et al. [26] do not specify if the token, represented by the photo in the example, is automatically verified by the system or if the verification has to be made by a human.

[10]Khan et al. [24] did not give a name to their location proof assertion protocol, so we write the title of this Section as a reference to the name of their paper.

of this *LA* being a malicious entity. The main change from the previous systems is that the Prover does not broadcast a request asking for LPs. The Prover sends a message directly to the Location Authority that is responsible for selecting a random witness in that site to provide an assertion for the Prover's claim. This prevents provers from selecting target witnesses so that they can collude with each other. Even if they do so, the addition of a *LA* between their communications will make it very hard to execute a collusion attack. After generating an asserted proof, the Witness sends it back to the *LA* so it can then forward it to the Prover. These communications between entities have to occur within a time threshold previously defined, to avoid replay attacks. As explained in more detail in [24], if two of these entities collude, the system will be able to detect it. The case where the three of them collude (*P*, *W* and *LA*) is not considered.

3.1.4 Location Provenance Proofing

In some cases, it is not only important to prove that a user was at a given place, at a given time, but it is also relevant to know the history of locations for a user over a given time window (*location provenance*). We cannot look at a location proof separately from the others, because now, for example, chronological order between them is important.

Consider the case where in a given competition, athletes have to go through different checkpoints, in a given order. Each time an athlete reaches a checkpoint, a location proof is generated. Knowing if the user went through all checkpoints is not enough because a verifier would need to know the order in which the athlete visited them. So, it will request all the proofs generated in the time frame of the competition. Khan et al. [29] introduced *OTIT*, a model that addresses several methods to solve this type of problem, where location provenance is important. They introduced new requirements for this type of location proofing systems, like chronological order preservation and proof subset verification, since the user may want to show only a part of his ordered location proofs acquired in the past, instead of showing them all. Khan et al. [29] describe methods to create and store ordered location proofs, as for example Bloom filters [11] or hash chains [12].

3.2 Summary

The different location certification systems and protocols addressed in Section 3.1 are presented in Table 3.1. Most of the location certification systems are inspired in the notion of LP given by Saroiu et al. [22], that is discussed in Section 3.1.2 of this dissertation. Nevertheless, earlier protocols, like Echo, tried to create this proof concept based on the time that two entities take to communicate with each other which can also be a good endorsement for a LP. The idea of having the users of the system working as witnesses for other users and certifying their location claims, is also a method used by multiple systems, as illustrated in Table 3.1. From all the presented systems, OTIT is the only one that deals with location

[11]A Bloom filter is a probabilistic data structure used to decide whether an element is part of a set.
[12]An hash chain is the successive application of an hashing function to data. The hash of the proof *i* is used when calculating the hash for the proof *i* + 1.

	Certification Type	Users as Witness	Ordering of Proofs
Echo	Distance-based	No	No
APPLAUS	Proof-based	Yes	No
CREPUSCOLO	Proof-based	Yes	No
Khan et al. [24]	Proof-based	Yes	No
OTIT	Proof-based	Not Specified	Yes

Table 3.1: Comparison between location certification systems.

	Communication between Prover and Witness	Additional Entities [13]	Resists Collusion
APPLAUS	Direct	None	No
CREPUSCOLO	Direct	Token Provider	No
Khan et al. [24]	Indirect	Location Authority	Yes

Table 3.2: Comparison between Witness-based proofing systems.

provenance, being able to establish relations between proofs for the same user. Therefore, OTIT is the only presented system that is able to easily understand the path made by a user.

The differences between the three systems which have their users also behaving as witnesses to other users are presented in Table 3.2. Although these systems are inspired in the same concept of location proof, each system has its own advantages and disadvantages. In *APPLAUS* and *CREPUSCOLO* the communication between user and witness is made in a direct way. This means that there is no intermediary in the communication, which makes it faster. Both of them make efforts to minimize collusion attacks, but cannot eliminate the attack in all cases. Khan et al. [18] has an additional entity, a Location Authority, that is responsible to mediate the communication between user and witness, making this communication indirect. This feature is the one that also makes the system resistant to collusion attacks, when two entities try to collude to deceive the system.

[13] A Prover, a Witness, a Server to store the proofs, a CA and a Verifier are common to all the systems in Table 3.2.

Chapter 4

SureThing

With the growth of IoT in the last few years, location based applications acquired more importance, since location is one of the most used types of context about the real world. For some cases, the simple estimation of the location of a user may not be enough. When there is more value at stake, methods to prove that a claimed location is true are necessary and the proofs should be robust. Regarding security properties, SureThing aims at providing the *integrity* of exchanged messages between entities. SureThing guarantees that the location proof can not be tampered by an attacker.

SureThing is a location proofing solution that other developers can use in their own mobile applications. Mobile application developers and service providers will find in SureThing the possibility to ask for location proofs based on different evidences, and with different degrees of assurance. SureThing frees programmers from worrying about how to get location information and how to prove it. By doing so, developers can spend more time enhancing their application logic. We envision that a common solution that can adapt to most scenarios is a great help to developers. SureThing uses different technologies, such as GPS, Wi-Fi or Bluetooth in order to understand where the user is located. Our system also relies on the definition of *witnesses* [24, 25, 26, 27] in order to help proving the location of a given user. These witnesses are other users of the system, carrying their smartphone that is running SureThing's application. Witnesses are in the same location as the user who needs a location proof. One of the main advantages of this witness-based approach is the independence from the infrastructure placed at a given space. It allows to create location proofs by using the community present at a given space. Two of the most important challenges in this approach are what to do when there are no witnesses available nearby to certify the presence of a user and how to deal with collusion attacks.

A SureThing prototype was implemented and an evaluation to the prototype was made, to validate our solution. This prototype includes a mobile application library, specifically for the Android Platform and two remote services, one for the service provider that needs to receive the proofs and a CA.

4.1 SureThing Entities

SureThing is a witness-based system, which causes the existence of two main entities: a *Prover P* and a *Witness W*. We decided to use the same nomenclature as APPLAUS [25] or CREPUSCOLO [26], that were described in Section 3.1.3. *P* is the user that wants to proof his location and will have to gather proofs from his neighbors. *W* is a user that is nearby *P* and that will give him a proof of his presence in the place. It is important to notice that a user of our system can, at a given point, ask for some location proof for himself, but can also, in some cases, be a witness for others.

For the trust root of SureThing, we assume that there is a *Certification Authority CA* that is responsible for generating a public key certificate for each user. Each user has its own private and public keys. The *CA* certifies all the needed keys among the users, being a trusted entity. We assume that each user of the system has to have a unique identifier when he is registering in the system.

A *Verifier V* in SureThing is some entity that needs a location proof from a user. *V* is providing a service to *P*, depending on his location, and has to receive location proofs that indicate that *P* is really in the claimed place.

4.1.1 Choreography

Figure 4.1 presents how the entities in SureThing entities communicate with each other when a LP is requested. In step *1* the Prover sends a service request to the Verifier. This request informs *V* that *P* wants to gather location proofs. *P* sends this request to receive information about what is the type of proof that the Verifier wants. *V* saves a time-stamp indicating the moment when it received this proof demand request.

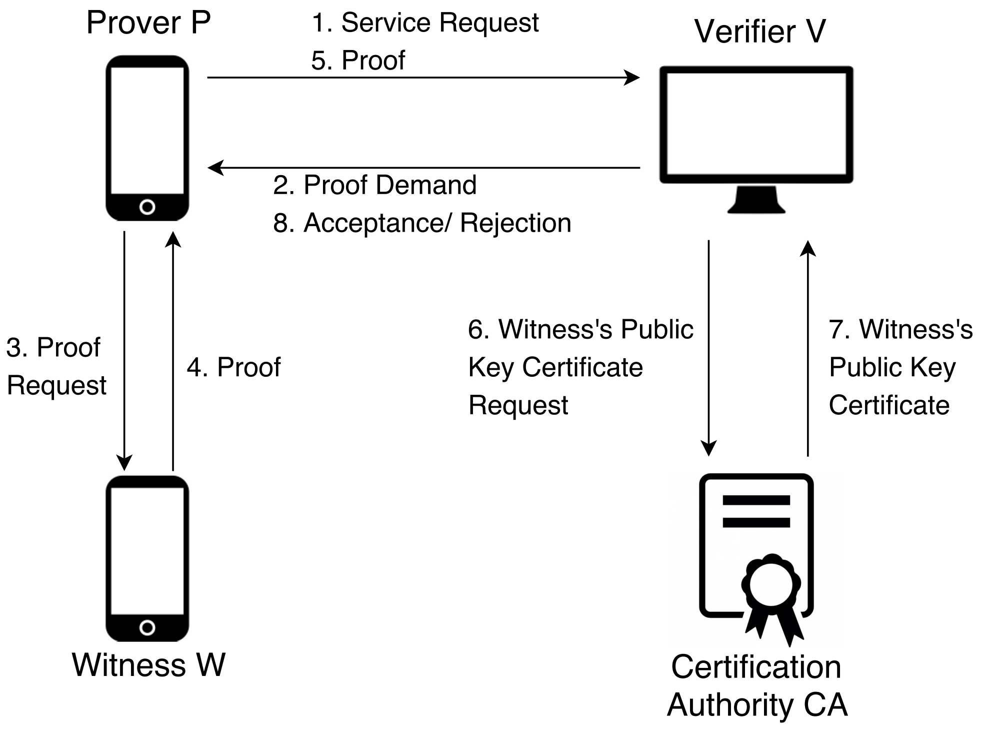


Figure 4.1: Communication between entities in SureThing.

The Verifier will respond to the Prover in step 2, with its **Proof Demands**. We will go in further detail about the content of a Proof Demand in the following sections. When *P* receives the Proof Demand from *V*, it will start his witness discovery process. After a witness is found, *P* will send to it a **Proof Request** in step 3. While searching for witnesses, the Prover was simultaneously getting its location data. In step 4, *W* returns a location proof LP to *P*. *W* signed the LP with his private key to guarantee non-repudiation and integrity in the exchange of the proof. In step 5, the Prover simply forwards the proof that he received to the Verifier. Step 6 and 7 indicate the interaction between *V* and *CA* after the Verifier receives a proof. *V* needs to check the signature in the LP so it requests to the *CA* the Public Key Certificate of *W*. After evaluating the proof, the Verifier decides to accept or reject it. In step 8, it informs *P* about this decision. If the proof was accepted, *P* can now receive his privilege.

4.2 Location Proof Techniques

A location proof (LP) in SureThing is a piece of information that can attest that a user is at a given place. A LP that is sent to the Verifier contains the following attributes:

- **Prover ID** - Identifier of the entity who needs a proof;
- **Witness ID** - Identifier of the entity who testifies the presence of the Prover in a given place;
- **Location of the Prover** - Location data of the Prover obtained through his smartphone sensors;
- **Location of the Witness** - Location data of the Witness obtained through his smartphone sensors;
- **Nonce** - Arbitrary and never used before number that was previously sent by the Verifier. It will ensure that old proofs cannot be reused;
- **Signature** - Digital signature of the proof done by the Witness to guarantee content integrity and authenticity.

Both the Prover and the Witness have to collect location data so that the Verifier can compare the information given by the two and check if it yields the same place.

Three different proof techniques were developed as shown in Table 4.1. These techniques are the following: **Geo**, **Wi-Fi** and **Beacon**. We chose these techniques because the technologies required to use them - GPS receiver, Wi-Fi adapter and Bluetooth radio - are widely available in commercial smartphones.

4.2.1 Setup Phase

Each of the proof techniques requires a *setup* stage to be performed by the service provider before locations proofs can be made. The setup includes adding logical descriptions of the physical places. For example, in the shopping center scenario, it would be up to the center staff to perform the setup procedures for each technique to be used and to label the places adequately e.g. food court, cinema, etc.

	Location Data Included in the Proof	Setup
Geo Proof	Geographic Location (obtained from GPS or ANLP)	Collect geographic coordinates with a correlated radius and associate them with their corresponding logical place
Wi-Fi Fingerprint Proof	Wi-Fi Fingerprint	Collect Wi-Fi fingerprints for each place and associate
Beacon Sensing Proof	Closest beacon ID detected	Associate beacon values with their corresponding logical place

Table 4.1: Proof Techniques Fields and Setup Phase.

When using Geo proofs, different geographic location measurements have to be collected from inside the area that we want to identify. Each location is defined by geographical coordinates, as latitude and longitude, and also has an associated radius. With that information, we have the area divided in multiple circles and the Verifier can then check if the locations given by Prover and Witness are inside the same area.

In Wi-Fi proofs, multiple readings are done to construct a fingerprint. A Wi-Fi fingerprint is a list of access points with an associated signal strength value. Each identifiable zone will have its own fingerprint.

For the Beacon Sensing Proof, SureThing relies on Bluetooth beacons that are constantly emitting a value. In the setup phase each beacon value has to be associated with the corresponding place where the beacon is.

4.2.2 Verification Phase

The tasks performed by *V*, to verify a proof, are presented in Algorithm 4.1. The Verifier always has to perform two tasks, regardless of the proof technique being used: first, it has to perform the verification of the proof's digital signature, made by the Witness (line 4 of the algorithm); secondly, the Verifier must check if the nonce received in the proof is the same that it previously sent to the Prover (line 5). The Verifier then proceeds to check the type of proof technique used to generate the LP.

If the location was obtained by the *Geo technique*, *V* will make a comparison between the client and witness's locations to determine if they are not separated by more than a threshold defined by the developer. This step is represented in line 9. In the setup phase, the developer has to define a maximum valid distance between Prover and Witness. After guaranteeing that the locations presented in the proof are near each other, the Verifier gets the symbolic places associated with them. If both locations yield the same logical place, the Verifier accepts the proof. Otherwise, it rejects it. The verification step made in line 9 prevents the Verifier from spending time evaluating and discovering the symbolical places in cases where two users indicate two very apart locations.

In the *Wi-Fi technique*, the Verifier begins to evaluate the proof in a similar way. It gets the symbolical

Algorithm 4.1: Verifier Tasks Algorithm

```
begin
    $proof \longleftarrow proofSentByTheProver$
    $proofTechnique = proof.getTechnique()$
    if $IsCorrectlySigned(proof, witnessID)$ then
        if $ValidNonce(proof)$ then
            if $proofTechnique = GEO$ then
                $clientGeoLocation \longleftarrow proof.getClientGeoLocation()$
                $witnessGeoLocation \longleftarrow proof.getWitnessGeoLocation()$
                if $locationsCloseToEachOther(clientGeoLocation, witnessGeoLocation)$ then
                    $clientPlace \longleftarrow GetSymbolicPlace(clientGeoLocation)$
                    $witnessPlace \longleftarrow GetSymbolicPlace(witnessGeoLocation)$
                    if $clientPlace = witnessPlace$ then
                        $return\ ACCEPT$
            else if $proofTechnique = WIFI$ then
                $clientWifiFingerprint \longleftarrow proof.getClientWifiFingerprint()$
                $witnessWifiFingerprint \longleftarrow proof.getWitnessWifiFingerprint()$
                $clientPlace \longleftarrow GetSymbolicPlace(clientWifiFingerprint)$
                $witnessPlace \longleftarrow GetSymbolicPlace(witnessWifiFingerprint)$
                if $clientPlace = witnessPlace$ then
                    $fingerprintInMap \longleftarrow GetWifiFingerprintInMap(clientPlace)$
                    if $EnoughAccessPointsInCommon(clientWifiFingerprint,$
                    $witnessWifiFingerprint, fingerprintInMap)$ then
                        $return\ ACCEPT$
            else if $proofTechnique = BEACON$ then
                $clientBeaconID \longleftarrow proof.getClientBeaconID()$
                $witnessBeaconID \longleftarrow proof.getWitnessBeaconID()$
                $clientPlace \longleftarrow GetSymbolicPlace(clientBeaconID)$
                $witnessPlace \longleftarrow GetSymbolicPlace(witnessBeaconID)$
                if $clientPlace = witnessPlace$ then
                    $return\ ACCEPT$
    $return\ REJECT$
```

places associated with both fingerprints provided by the Prover and Witness. For each AP that is within the fingerprint provided, *V* checks how much the RSSI associated with it differs from the RSSI values previously saved for the identifiable places. The place that has a fingerprint with the smallest euclidean distance to the fingerprint provided in the proof is selected as the place where the user is. This process is executed twice (for both the Prover and Witness). If they yield the same place (line 19), the Verifier has to do one more verification step. The fingerprints given by *P* and *W* are compared with the original fingerprint saved in the setup phase for that logical place and *V* compares how much APs the fingerprints have in common. For example, if the three fingerprints have information about 10 APs but the Prover's fingerprint and the fingerprint built in the setup only share 1 AP, it is probable that *P* is not really in there. *P* should have detected more APs that were saved in the setup phase. This percentage threshold can be adjusted. In more dynamic environments, where there are constantly new APs being installed, a larger threshold can be acceptable. In very stale environments, the percentage of common APs should be close to 100%.

When using the *Beacon Sensing technique*, the Verifier only has to get the logical places associated with the beacon identifier that it is provided by Prover and Witness. Each beacon identifier has been associated with a logical place and the Verifier checks that association in this step. The Verifier accepts the proof if the same identifier is provided by both entities.

4.3 Witness Models

When a Prover needs a location proof, it will ask for a LP to one or more devices nearby, known as *witnesses*. Previous solutions that used a witness-based approach focused on random mobile witnesses. These witnesses are other users that have the same application on their smartphone. Usually, in systems that use random mobile witnesses, users can behave as Provers or Witnesses for other Provers. We envision that, although it is a good solution, since it allows to gather proofs from close peers without any installment of further infrastructure, it will not fit in some use cases for location-aware applications. For example, if *P* is at a place with low attendance, it is possible that a Witness of this type will never generate him a proof, since *P* will never find a Witness. We also envision that in some cases we can differentiate between trusted and untrusted witnesses.

SureThing offers two main types of witness models: *Master* and *Mobile*. If no witnesses are available nearby, the developer has the opportunity to include in his application a secondary witness model, known as *Self* Witness model, that will generate a weak LP. It will be up to the Verifier to decide if the Self Witness is acceptable or not, given the request context. We present our witness models below:

- **Master** - certified witness that can be *trusted* by the Verifier;
- **Mobile** - *untrusted* random witness. The Verifier does not trust the proofs gave by mobile witnesses because they can be colluding with *P*;
- **Self** - *P* acts like his own witness and generates a proof for himself.

The Verifier indicates to the Prover what is the model that it should use when gathering location proofs. The Prover will then initiate its search for witnesses that correspond to the profile requested by the Verifier. The Verifier does not command the Prover to choose a explicit witness, it only states the type of witnesses that should be used.

4.3.1 Master Witness Model

The most important feature about the *Master* Witness is that it can be *trusted* by the Verifier. Consider our shopping center example, where in a given book store all the employees have their smartphone equipped with a SureThing-based application. These employees are the master witnesses. The clients that enter in the book store can request to them proofs of their location. Only the clients near the employees should be able to receive a privilege from that store.

The *Master* Witness Model eliminates the necessity of developing and working with collusion avoidance mechanisms, since we assume that we can trust the master witness. However, this model can have some drawbacks. First, this method can only be applied in specific conditions, like in the example described above, where we can clearly identify a witness (e.g. the store employee or the CEO of a company) that will provide stronger assurances about the location of a user. Secondly, if there are too much Provers for just a few Master Witnesses, this can introduce a bottleneck in the system.

When registering with the CA, users that will act as master witnesses provide their Bluetooth address. The Verifier keeps a map with all the pairs <*witnessID*, *address*>. *V* needs to keep this information to inform *P* about how to discover and contact the master witnesses.

4.3.2 Mobile Witness Model

Mobile witnesses are heavily inspired on the idea of selecting random nearby devices that act as witnesses for a given Prover *P*, as proposed in other works [24, 25, 26, 27]. Mobile witness solutions can fit very well in crowded places, like the mentioned shopping center. If *P* is at a place with many other users, they can all exchange location proofs between them to achieve some kind of privilege. However, the Verifier cannot trust the proofs given by mobile witnesses without further verifications. Mobile witnesses are users that are registered in the system but the Verifier cannot trust their intentions.

This model does not depend in a central witness, since every user can behave as a witness. With this approach, there will no bottleneck in the system. It is also a solution for situations where a distinctive witness cannot be found or used. Nevertheless, it arises security problems, since collusions between users are now a possibility. Collusion avoidance mechanisms will be discussed in Section 4.4.

4.3.3 Self Witness Model

If the Prover did not found any master or mobile witness to grant him a location a LP, we offer a third witness model where the Prover will generate a proof for himself. This model should only be used as a last resort model and only at particular situations.

Consider our shopping center example. These type of spaces are often crowded, so the possibility of finding master or mobile witnesses is very high. In a situation like this, the Self Witness Model will not be a good solution, because *P* should have found at least some witnesses. If we are developing applications for environments with less attendance and with less value at stake, this model can be helpful.

Wi-Fi Fingerprinting and Beacon Sensing proof techniques are more suited to this model than the Geo technique. The first two require *P* to have some exclusive information about the space, such as the Wi-Fi Fingerprint or the closest beacon value that is being emitted. However, for this model to create more robust proofs, this proof data (e.g. beacon value) needs to change more often, so that *P* cannot re-use it.

4.3.4 Comparison of Witness Models

In Table 4.2 we present a summary and a comparison between the three proposed witness models. The compared features are: *trustiness* by the Verifier; bottleneck potential; need for collusion avoidance mechanisms and the best scenario to use the model.

	Trusted by the Verifier	Bottleneck Potential	Needs Collusion Avoidance Mechanisms	Best Scenario
Master	Yes	High	No	Low attendance, where there are trusted users that can act as witnesses
Mobile	No	Low	Yes	High attendance
Self	No	None	No	Low or none attendance, with no users that can act as Master witness

Table 4.2: Comparison between Witness Models.

4.4 Collusion Avoidance Mechanisms

One of the problems that arise from mobile witnesses solutions is the possibility of collusion between *P* and *W*. If both Alice and Bob lie to the Verifier, and indicate the same location information, *V* must have mechanisms to detect such malicious acts. Since our Mobile Witness Model (Section 4.3.2) uses random witnesses we had to include some methods to avoid collusions in our solution. We propose two different collusion avoidance mechanisms in our prototype: *Witness Redundancy* and *Witness Decay*.

In the *Witness Redundancy* mechanism, *P* has to gather proofs from *k* different witnesses instead of only one. This parameter *k* should vary with the value of the service that is being offered and with the level of attendance of the place where the proof is going to be generated. The Verifier is the entity that decides the value of the parameter *k*. The *APPLAUS* system uses a similar method for discovering malicious witnesses.

In the *Witness Decay* mechanism, each Witness gives proofs with different values to a given Prover. If *P* is always getting proofs from the same *W*, its proofs gradually become less valuable. If *P* does not gather proofs with enough value to satisfy the Verifier, the latter will not provide any service to *P*.

$$V_{\text{xy}} = \begin{cases} V & \text{if } N_{\text{xy}} = 0 \\ V - \frac{{N_{\text{xy}}}^k}{U} & \text{if } N_{\text{xy}} > 1 \end{cases} \tag{4.1}$$

The proof value is calculated with Formula 4.1 where V_{xy} represents the proof value given to user *x* by Witness *y* and *V* represents the maximum proof value. N_{xy} represents the number of times that *y* testified the presence of *x* and *U* is the total number of users in the place. If the Witness never testified the presence of the Prover (N_{xy}=0), the generated proof will have the maximum possible value. New proofs created by the same Witness to the same Prover will have less value ($N_{\text{xy}} > 1$). The probability of finding the same Witness again is given by $\frac{1}{U}$. We multiply this probability by the number of times that the Witness gave a proof to the Prover, in order to penalize proofs where Witnesses and Prover are repeated. The parameter *k* can be specified by the application to increase or decrease the speed of the Witness decay, as needed in a particular use case. After calculating this decay factor, we subtract it from the maximum proof value possible, in order to obtain the final value of the proof.

4.5 Communication Protocol

After the introduction of both location proof techniques and witness models (Section 4.2 and 4.3 respectively) we can revisit how the communication between entities occurs with further detail.

4.5.1 Proof Demands

Since the Verifier is the authority responsible for accepting or denying the proof, the Prover will send a **Proof Demand** request to the Verifier when he wants to receive information about what is the type of proof that the Verifier wants to be generated. A Proof Demand is a document, in machine-readable format, to express the requirements of a successful proof. The Verifier sends to the Prover a Proof Demand with the following fields:

- *Witness model*: indicates the witness model to be used when gathering proofs. If the witness model is the *Master* Witness model, there is one more communication step between the Prover and the Verifier, where the first asks for the Master Witnesses addresses;
- *Proof technique*: indicates the proof technique to be used by both Prover and Witness when collecting proofs;
- *Nonce*: random and never used before number, to avoid replay attacks;
- *Number of witnesses*: this parameter is only sent when the proof demand uses a Mobile Witness model and the collusion avoidance mechanism being used is the Witness Redundancy.

With this information, *P* will ask for meaningful proofs to the witnesses nearby. *V* can send more than one Proof Demand to *P*. Consider the scenario where *V* sends to *P* the proof demands presented in Figure 4.2. *V* sends two ordered proof demands. The Prover will try to satisfy the first demand by gathering proofs where the proof technique being used is Wi-Fi Fingerprinting and the witness model is the Master model. For the sake of the example, consider that are three master witnesses registered in the system. The Prover can contact any of these master witnesses to fulfill the first demand. If *P* does not find a Master witness to generate a proof, *P* will try to satisfy the second demand where the witness model being used is the Mobile model. If the proofs presented by *P* are from master witnesses, *P* only has to present one proof. If proofs were created by mobile witnesses, *P* has to gather two witnesses. *V* will also send a random nonce to the Prover to avoid re-use of proofs in the future. In this case the Verifier offers a second opportunity to the Prover to satisfy a proof demand if it did not succeed while trying to complete the first demand. If the service provider represented by the Verifier only wants to accept proofs from Master witnesses, only the first demand in the example should be sent to the Prover.

P starts the witness discovery process after receiving the necessary information. If the demand being executed is the one containing the Master Witness model, *P* will only establish contact with a

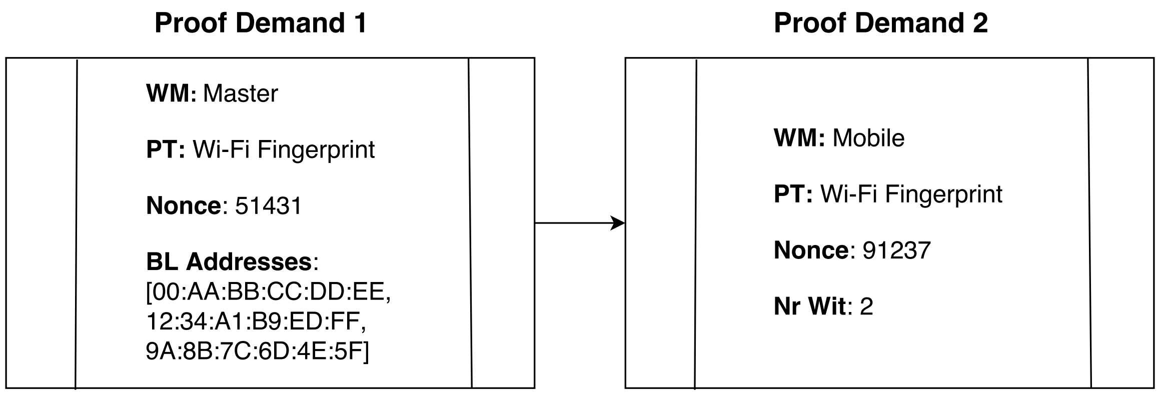


Figure 4.2: Example of a list of Proof Demands sent by the Verifier. *WM* - Witness Model; *PT* - Proof Technique; *BL Addresses* - Bluetooth Addresses of the Master Witnesses; *Nr Wit* - Number of Witnesses

device with a Bluetooth address from the list that it received from *V*. Even if some malicious device spoofs its Bluetooth address to look like a master witness, the Verifier will later understand that the proof was not signed by a real master witness, via the digital signature verification, and will decline the proof.

4.5.2 Proof Requests

After a witness is found, *P* will send to it a **Proof Request**. This Proof Request object is composed by four fields:

- The proof technique within the Proof Demand that *P* received from *V*;
- Nonce within the Proof Demand;
- Identifier of *P*;
- Location data of *P*.

P sends to the Witness *W* the proof technique presented in the proof demand because *W* has to know how to obtain its location. *V* will also add the nonce to the proof, so that the Verifier can confirm the freshness of the LP. This nonce is a random and never before used number. The Verifier memorizes the nonces associated with the requests from each Prover and pairs them with the timestamp of the request. Once the Verifier receives a proof associated with a given nonce, it will erase the association between the nonce and the Prover, so no more proofs with the same nonce can be accepted. The identifier of *P* will be used in the construction of the proof by *W*. The location data of the Prover is also sent to *W* in this step. While searching for witnesses, the Prover was getting its location data in parallel, that can be physical coordinates (Geo proof technique), a Wi-Fi fingerprint or a Bluetooth Beacon Identifier.

The proof returned by *W* to *P* contains the fields described in Section 4.2: *Prover ID*; *Witness ID*; *location data of the Prover*; *location data of the Witness*; *nonce* and *signature of the proof*. *W* has also determined its location data, by using the proof technique in the proof request that it received. *W* signed the LP with his private key to guarantee integrity and non-repudiation in the exchange of the proof.

If the proof demand stated that the number of witnesses should be higher than 1, *P* will wait for other proofs to be received so he can send all at the same time to the Verifier. If after a while it did not receive any more proofs and it has received more than one proof demand, *P* will try to satisfy the next proof demand.

The rest of the process runs exactly as described in Section 4.1.1. The Verifier will analyze the proofs given by the Prover and will decide if it should accept the claim of the Prover or not.

4.6 Summary

SureThing is a location proofing solution that gives developers a common tool to develop new location-based applications where locations can be trusted by service providers. SureThing has four main roles: a Prover, a Witness, a Verifier and a Certification Authority. We designed SureThing to be a witness-based system, where these witnesses help the Verifier to assert the location of a given Prover. Witnesses are mobile users that have a SureThing-based application in their smartphone. This design allows to create a community that will exchange proofs between them and it is also a solution with a high level of independence from the infrastructure of the places that we want to identify.

We introduced the definition of a location proof based on the location data obtained by the Prover and the Witness. A proof should always contain the location data from the two entities, so that the Verifier can later compare them to check if they yield the same place. The obtained location data can be geographical coordinates, a Wi-Fi fingerprint or a Beacon identifier. The Verifier has the responsibility to inform the Prover about what location estimation technique to use. The Prover then gives that information to the Witness. A proof is always signed by the Witness to guarantee integrity and non-repudiation.

We also proposed three different witness models: Master, Mobile and Self. The model that the Prover should follow is provided by the Verifier. The Master and the Witness model are the main models in our design while the Self model is a last-resort, where the Prover will generate a proof for himself. The Master Witness is a certified user that is trusted by the Verifier while the Mobile Witness is a random user of the application. To deal with untrusted mobile witnesses we introduced two collusion avoidance mechanisms: Witness redundancy and Witness decay. In the first, the Verifier specifies how many proofs from different witnesses the Prover should get. In the Witness decay mechanism, a proof from a given Witness loses value when a given Prover starts presenting proofs from the same Witness multiple times.

In the next Chapter we will present a prototype of SureThing, that transforms the presented design in a full working system.

Chapter 5

Prototype Implementation

In order to turn our design into a working system and to evaluate it, we developed a SureThing prototype that we will describe here. As presented in Chapter 4, our design for SureThing relies on four main roles: Prover, Witness, Verifier and Certification Authority. We implemented a SureThing prototype with three main components: a library for mobile applications (Prover and Witness), a remote Verifier and a remote Certification Authority. We decided to make Prover and Witness communicate via Bluetooth, due to its short range of communication, since users need to be near each other to provide the same location data. The Verifier and the Certification Authority act as Representation State Transfer (REST) web services. We will discuss in detail how we built each of these components and how they interact with each other.

5.1 Project Structure

Figure 5.1 presents the project structure of SureThing. We have the three main components represented as the Verifier, the Certification Authority and the SureThing's Mobile Application Library. These entities were written using the same programming language, Java. To avoid the repetition of code all of them use a shared library. This shared library stores the classes that are used in more than one different project. For example, if the Verifier and the Mobile Application library have to use the same class, the latter will be defined in the shared library. New trusted location-aware mobile applications have to use the *SureThing's Mobile Application Library* to take advantage of SureThing's capabilities. This is the library that developers will reuse when developing a location-based mobile application based in SureThing.

5.2 Mobile Application Library

We decided to implement a SureThing prototype for mobile devices running the Android Operating System (OS), the most widely used mobile operative system [14]. Developers can use this Android library

[14] `http://www.businesswire.com/news/home/20170403005635/en/Android-Overtakes-Windows-Time-%E2%80%93-StatCounter`

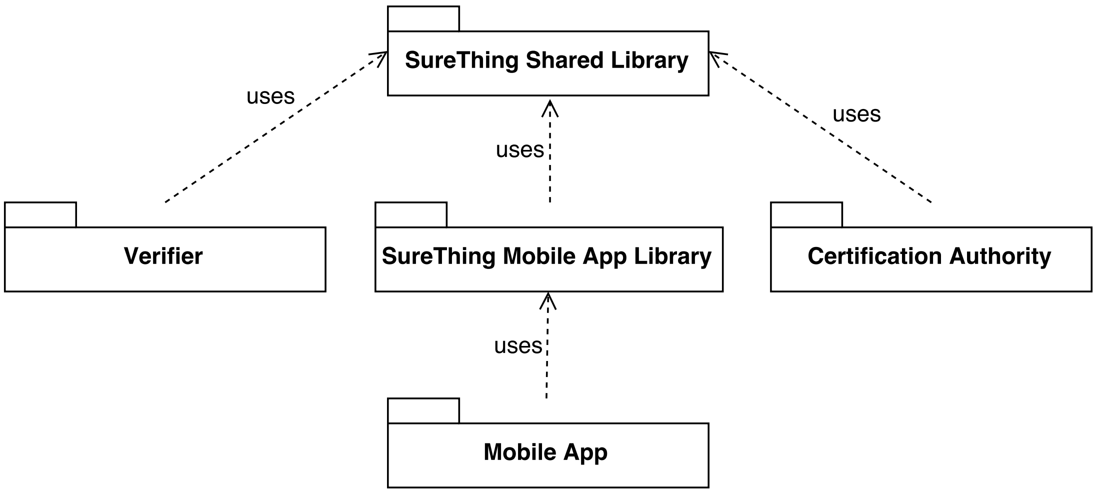


Figure 5.1: Project Structure of SureThing.

to build their own trusted location-based mobile applications. Since it is a library that will be used by Android applications, the programming language used was *Java*. *Android Studio* [15] was the IDE used, since it is the most common used IDE for Android applications and it is suggested by the Android community. *Gradle* [16] was the build and dependency management tool used, since it is already integrated with Android Studio when developing an Android project. Bluetooth was chosen for communication between mobile devices, mainly due to its short range. It is also a widely used technology, present in almost every smartphone model. Normally, the Bluetooth specifications require a pairing process between devices before any data is transferred. However, this specification is not a good fit for the purposes of SureThing. In most cases, Prover and Witness will not know each other, so they cannot share any information to pair their devices. We also want our solution to have the least user intervention possible. We use Bluetooth communications without pairing. The potential security implications of this option are addressed in the application layer. We decided to create a common library for both Prover and Witness to use. We created only one application to be used by both entities. Since in our proposed Witness Models, a user can be both Prover and Witness, even at the same time, this is a good design for our solution. We need to combine the benefits for the Prover with the giving back to the community made by the Witness. We assume that smartphone users are willing to share their locations, network bandwidth and battery.

Figures 5.2 and 5.3 present the main class diagrams that support this library. We divided in two diagrams for better understanding of how the system behaves.

5.2.1 Initialization

Figure 5.2 presents the main classes that are used when SureThing starts to execute in the developer's application. The *Proof Manager* class is the main entry point of SureThing's library. The developer must initialize this object in his application, as shown in Listing 5.1. The *Proof Manager* contains a *User* object, that basically contains the identifier of the Prover/ Witness. The method *beginProofProcess()* will

[15]Android Studio is an Integrated Development Environment (IDE) developed by Google and designed specifically for Android development: `https://developer.android.com/studio/index.html`
[16]Gradle is an open source build automation system: `https://gradle.org/`

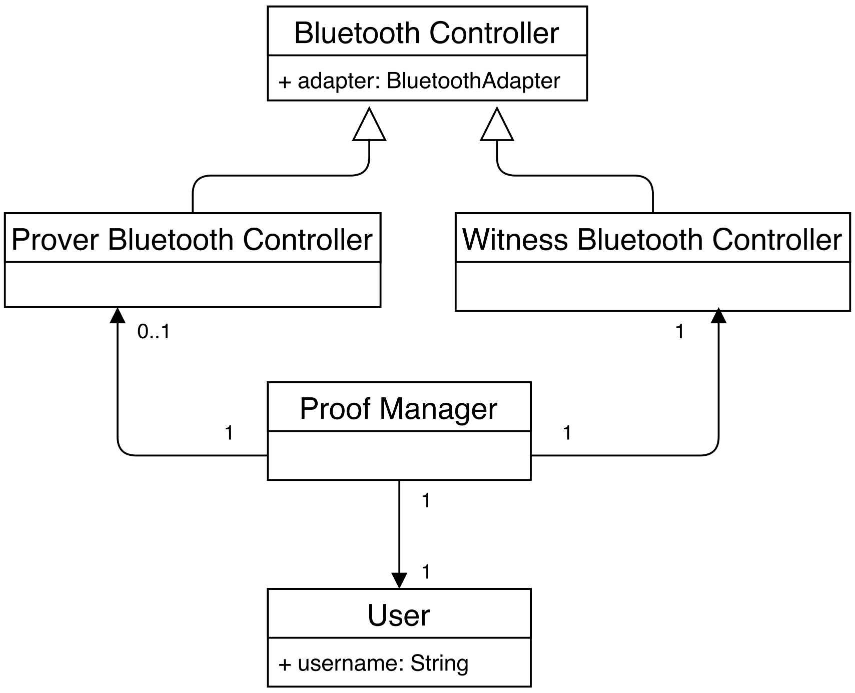

Bluetooth Controller
+ adapter: BluetoothAdapter
Prover Bluetooth Controller
Witness Bluetooth Controller
0..1
1
Proof Manager
1
1
1
1
User
+ username: String

Figure 5.2: Class diagram of the classes used in the initialization.

start the proofing process with a call to the Verifier to get the current proof demands.

Two very important classes are connected with the *Proof Manager*: two *Bluetooth Controllers*, one for the Prover and another for the Witness, since that a given user may at a given point play the role of a Prover and/or a Witness. We called these objects *controllers* because they will control how and when the Bluetooth communication should operate. The *Prover Bluetooth Controller* will be initialized when the Prover starts to receive demands from the Verifier. This class is responsible for the lookup of witnesses via Bluetooth. After discovering them, it will launch secondary threads to establish contact with the witnesses. The *Witness Bluetooth Controller* initializes a secondary thread to listen to proof requests via Bluetooth. The adapter in the *Bluetooth Controller* class is an instance of the object *Bluetooth Adapter* [17]. This object offers fundamental Bluetooth functionalities, such as devices discovery or instantiation of a socket to listen to connection requests. Before initializing the Bluetooth Controllers, SureThing checks if the Bluetooth is turned on and if the device is set to discoverable mode, so it can be found by other devices. If Bluetooth is off or the device is hidden, a dialog box is shown to the user, asking for his permission to turn on Bluetooth and/or to set the device to discoverable mode.

Listing 5.1: Proof Manager Initialization and proofing process start point.

```java
User user = new User("Alice");
Context context = this; // Current Activity
ProofManager proofManager = new ProofManager(user, context); // Prover's Bluetooth Controller
    initialized in the Proof Manager class;
proofManager.beginProofProcess();
```

[17] *Bluetooth Adapter* class: `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html`

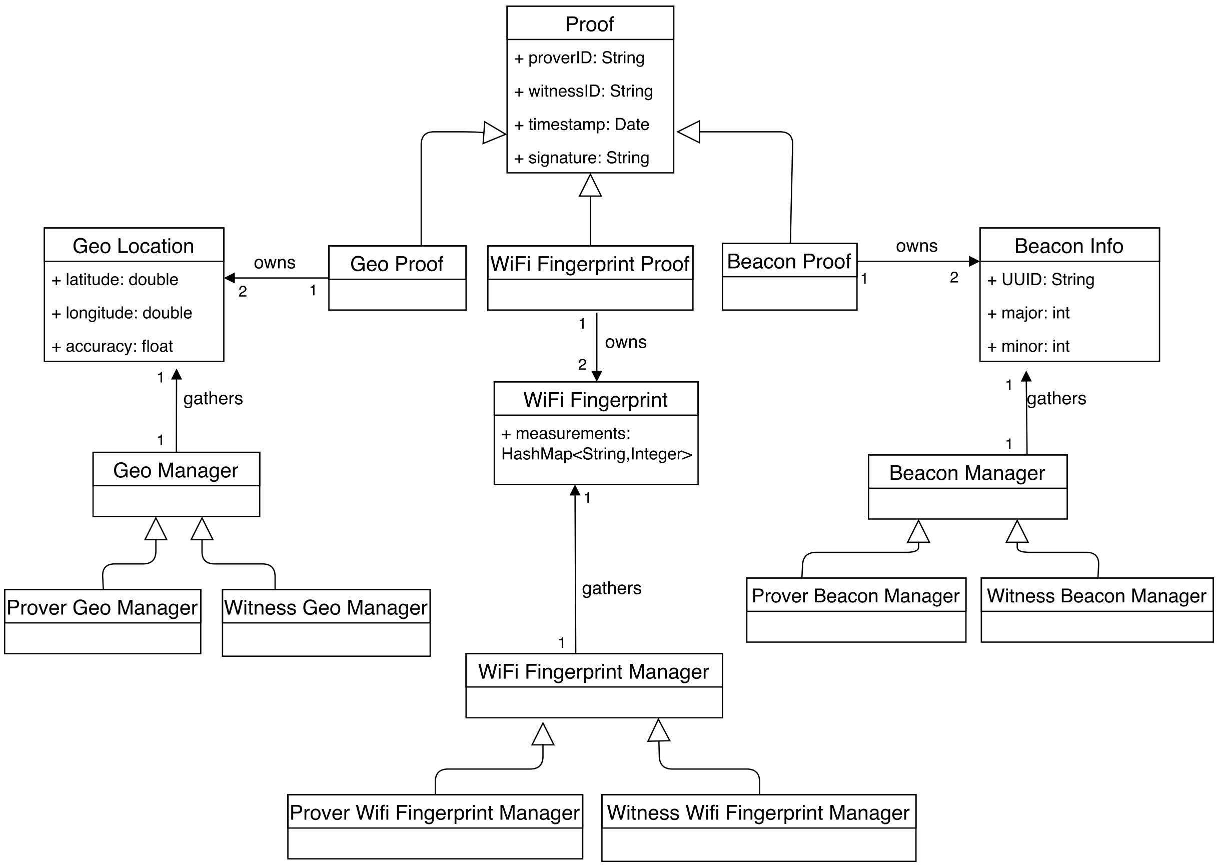


Figure 5.3: Class diagram with the main classes used by both Prover and Witness.

5.2.2 Proof Processing

Figure 5.3 presents the main classes that are used both by the Prover and the Witness [18]. *Proof* is an abstract class to represent the concept of LP. Each of its inherited classes represents a different proof technique (Section 4.2).

A *Geo Proof* always contains two *Geo Location* objects. One of these objects characterizes the location of the Prover and the other the location of the Witness. Each *Geo Location* is represented by its latitude, longitude and accuracy. The *WiFi Proof* and the *Beacon Proof* follow the same logic, but they own two *WiFi Fingerprint* and *Beacon Info* objects, respectively. A *WiFi Fingerprint* holds a field measurements within a map. The key of this map is the AP identifier whereas the value is the RSSI detected from that AP. Since we used a specific implementation of the iBeacon standard [19] as the Bluetooth beacons, the *Beacon Info* class holds the three main iBeacon identifiers: an UUID, a major value and a minor value (as detailed in Section 2.2.4).

The *Manager* classes presented in Figure 5.3 are the ones responsible for gathering each type of location data. For example, the *WiFi Fingerprint Manager* is responsible for gathering the fingerprints that will be used in the proofs. *Manager* classes are divided in two because Prover and Witness will gather location data for different goals. The *Prover Manager* goal is to inform the Witness about the Prover's location, so that the Witness can add it to the proof. The *Witness Manager* classes, after

[18]The *Proof* class and its inherited classes belong to the Shared Library since they are also used by the Verifier

they obtain the appropriate location data, will construct a proof and send it to the Prover. They share a common *Manager* class because the procedure to obtain the location does not change whether it is obtained from a Prover or a Witness. Only the use that each of them gives to the fingerprint is different.

5.3 Verification and Certification Services

Both the Verifier and the CA are RESTful web services, written in Java. The data transferred to these two entities and from them is in JavaScript Object Notation (JSON), a more lightweight data exchange format than Extensible Markup Language (XML). JSON is the most common choice when working with mobile devices as clients, due to its compactness, while still preserving the convenience of text based formats and ability to be mapped more easily in object-oriented programming languages, as in Java or JavaScript. We used the *Gson* [19] library, developed by Google, to transform the Java objects into JSON format and from this format to a Java object again.

We developed both entities with Eclipse [20] IDE. As for the mobile application library, we used Gradle as the dependency management tool for both the Verifier and Certification Authority. The two entities depend on configuration files so that they can operate as expected. Both of them need a configuration file where the host name and ports where their services should run are provided.

5.3.1 Verifier

The Verifier has further configuration files that are needed. These files are in JSON format and the service provider has to change their content depending on its needs. Each file contains information about a different feature from the following list:

- Proof Demands;
- Addresses of the Master Witnesses;
- Geo Data;
- Wi-Fi Fingerprint Data;
- Beacon Data;
- Thresholds (e.g. the maximum distance between Prover and Witness when the Prover is providing a Geo Proof)

Listing 5.2 presents a Proof Demand configuration file example. Upon the begin of its execution, the Verifier loads the content of this configuration file to know what are the Proof Demands that it should sent to the Prover. The demands in the configuration file do not contain the nonce, that will be later added by the Verifier, since each time time it sends a demand the nonce will be different. The *rank* parameter allows the Verifier to order the Proof Demands.

[19]Gson is a library developed by Google that can be used to convert Java Objects into their JSON representation and vice-versa: `https://github.com/google/gson`

[20]Eclipse is an IDE that is mainly used for developing projects in Java: `https://www.eclipse.org/`

Listing 5.2: Proof Demand Configuration File.

```
[
    {
        "witModel":"WITNESS_MASTER",
        "proofType":"PROOF_AMBIENT_BEACON_SENSING",
        "rank":1
    },
    {
        "witModel":"WITNESS_MOBILE",
        "proofType":"PROOF_GEO",
        "numberOfWitnesses":4,
        "rank":2
    },
]
```

Listing 5.3 presents an example configuration file to be used in the evaluation of Geo Proofs.

Listing 5.3: Geo Data Configuration File.

```
[
  {
    "latitude": 38.737964,
    "longitude": -9.303260,
    "radius": 23.0,
    "place": "Cinema"
  },
  {
    "latitude": 38.737527,
    "longitude": -9.303152,
    "radius": 23.0,
    "place": "Food"
  },
  {
    "latitude": 38.737184,
    "longitude": -9.302925,
    "radius": 23.0,
    "place": "Clothing"
  }
]
```

The Wi-Fi Fingerprint configuration file is the only one that can be populated in an automatic way. SureThing's mobile library has a functionality that allows the construction of a map where each AP is associated with the corresponding RSSI value. This map is labeled with the place that it corresponds to e.g. Food Court. A JSON is sent to the Verifier with this data and the Verifier will save it in the configuration file.

The Verifier is a REST web service that offers the following services to the Prover:

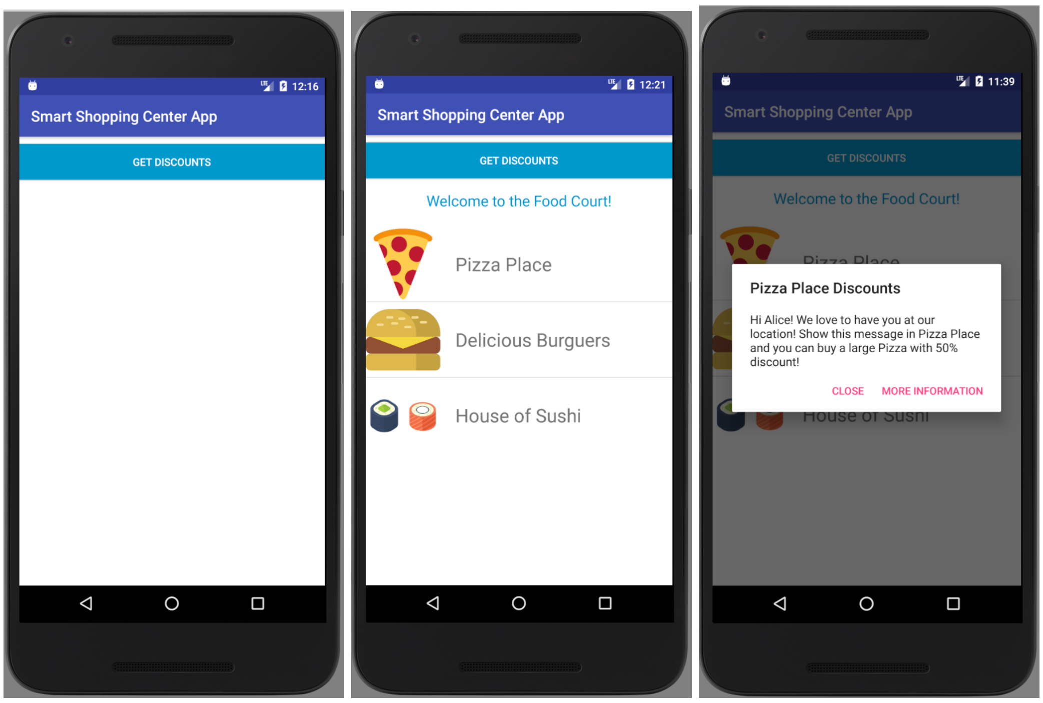
- *GetProofDemands()*: called by the Prover to know what are the proof demands of the Verifier;
- *PostProofsGeo (List<GeoProof> proofs)*: the Prover sends a list of collected Geo proofs;
- *PostProofsWifiFingerprint (List<WiFiProof> proofs)*: the Prover sends a list of collected Wi-Fi proofs;
- *PostProofsBeaconSensing (List<BeaconProof> proofs)*: the Prover sends a list of collected Beacon proofs;

5.3.2 Certification Authority

The CA offers different services to the Prover and to the Verifier. To the first, the CA offers a post method to register in the system. When the mobile user contacts the CA to register in SureThing, it gives to the CA a *Certificate Signing Request* object. This object contains the identifier of the user, the password and its public key. The private key and the public key are generated by the user's smartphone with the RSA algorithm. We assume that the communication with the CA is done through a secure channel. The CA generates a public key certificate for the user. These certificates use the X509 standard. To the Verifier, there is a get method to get the public key certificate from a given user of the system. The identifier of the user has to be passed as the argument of the method.

5.4 Example Application

Consider the following example where SureThing is used to develop an application to create the shopping center scenario introduced in Section 1.2. Figure 5.4 shows a simple interface for the shopping center's application. Consider that Alice, a user of the application that is in the shopping center's food court, wants to know what are the available discounts. She will click in the button depicted in Figure 5.4(a). The application will now try to prove Alice's location. It will contact the Verifier to know what are the current proof demands that it is requesting. Suppose that the Verifier sends only one proof demand stating that the Prover should find a Master Witness and use the Wi-Fi fingerprint technique to obtain its location data. When Alice's smartphone receives these demands, it will try to find a nearby master witness. At the same time, it will get its location data using Wi-Fi fingerprinting. Bob is a shopping center's employee and is recognized by the Verifier as a Master Witness. Alice will contact Bob to receive a proof of her location. Bob's smartphone will calculate his location data with the same technique that Alice's smartphone as used. When it finishes, it will sent back to Alice a signed location proof. Alice can now



Hi Alice! We love to have you at our location! Show this message in Pizza Place and you can buy a large Pizza with 50% discount!

(a) Button to begin location proofing process. (b) Screen after proof acceptance. (c) Screen after selecting one restaurant.

Figure 5.4: Screens of the Shopping center's application to give discounts to certified users.

send this proof to the Verifier, that will evaluate it. The Verifier checks that the signature and nonce are valid and that both location data within the proof indicate the same place, the food court. Alice's location proof is accepted and Alice can now see on her smartphone what are the restaurants where she can use a discount, as depicted in Figure 5.4(b) and 5.4(c).

SureThing was used to manage all the location estimation and proofing used in this application and all entities communicated as proposed in SureThing's design (Figure 4.1 in Chapter 4). The application developer does not have any contact with the location estimation and proofing features when developing his application. He only has to specify the moment when the proof demands request is made to the Verifier and what to show when proofs are accepted.

5.5 Summary

We presented a full working prototype for SureThing that allowed to develop a location-based application to transform our motivating use case (Section 1.2) into a working system.We divided our implementation in three different main projects: a Mobile Application library, that mobile applications should use in order to develop location-proofing systems based on SureThing; a Verifier and a Certification Authority. We developed our Mobile Application library for an Android environment what made Java the chosen programming language. Both Prover and Witness have to use this library. The communication between these two entities is supported by Bluetooth. We chose this technology due to its short range of communication. The Verifier and the CA are RESTful web services also written in Java. The data transfered to these entities and from them is in JSON format. The necessary configuration files to make both the Verifier and the CA behave as expected are also written in JSON.

In the next Chapter we will evaluate our prototype to make sure that our system is well designed and meets the necessary technical requirements.

Chapter 6

Evaluation

In this Chapter we will present and discuss the evaluation results of the SureThing prototype. We will explain in detail what were the experiments conducted to validate our proposal. We focused the evaluation on three different aspects:

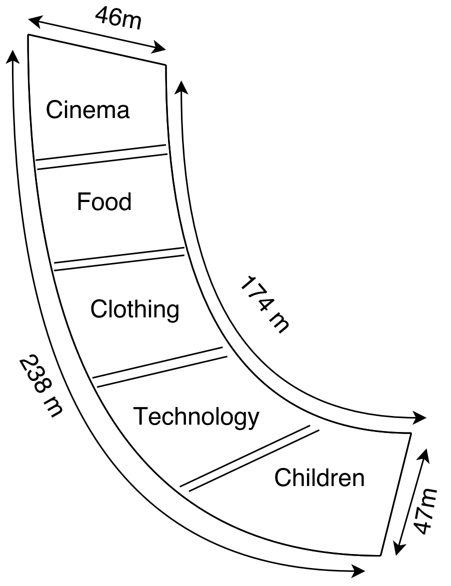
- How accurate are the location estimation techniques?
- How long does it take to issue a location proof?
- Are the collusion avoidance mechanisms suitable to real world smart spaces?

6.1 Location Estimation Accuracy

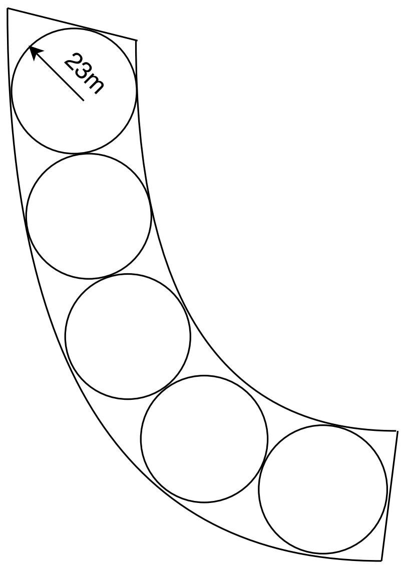
As a scenario for evaluation, we used a real world building to stand in as the shopping center. The location map is represented in Figure 6.1(a). We divided the area in five main zones: Cinema, Food, Clothing, Technology and Children. We tested the location estimation accuracy for each technique proposed, starting with Geo and Wi-Fi Fingerprinting.

6.1.1 Geo and Wi-Fi Fingerprinting

Since Geo proofs were used in an indoor environment, we chose to obtain geographic coordinates from the ANLP. Each of the shopping zones has its own Wi-Fi fingerprint that was collected after 100 signal strength readings, to ensure that poor values did not have a strong impact in the final fingerprint. For Geo and Wi-Fi proofs, our goal was to identify in which area from Figure 6.1(a) the user is in. Figure 6.1(b) represents the areas that were defined for use in Geo proofs, one for each zone. We tested Wi-Fi fingerprinting to determine the current shopping zone where the user is with different quantities of Wi-Fi readings. In each reading, all the access points found were saved with an associated RSSI value. We wanted to compare if by doing multiple readings we can diminish the positioning error. For both techniques, we did 30 estimations per area. The readings were collected in different spots of each area to diversify the obtained sample. Our results are presented in Figure 6.2. We can observe that,



(a) Different areas in the Shopping Center.



(b) Geo Proof Setup.

Figure 6.1: Shopping Center building.

for the majority of places inside the shopping, Wi-Fi fingerprinting with 10 readings was the technique with the highest score. By doing just 1 reading, is was not possible to correct errors, since not enough access points to calculate a good fingerprint were found. When we increased the number of readings to 5, we had an increase in location accuracy, since it is possible to detect more access points and if some reading was poor, the others can correct it. With 10 readings, a poor reading will not affect as much the final result, leading to better results. The areas of *Cinema* and *Clothing* prove the importance of doing multiple Wi-Fi readings, since they yield a difference close to 20% between Wi-Fi fingerprinting with 5 and 10 readings. We did not increase any further the number of readings because each reading took about 550ms. If we continued to increase this number, the location proof will take too long to be generated. Geo has, most of the time, a lower accuracy than Wi-Fi fingerprint (with 5 and 10 readings).

We also tested if we could detect if a user is not in any of the building areas. As we can conclude from Figure 6.2, Geo was able to identify half of the situations. Wi-Fi fingerprint had a success rate over 80% when the user was outside of the building. When evaluating a fingerprint provided by a user, we check what is the percentage of access points that the user should not have identified, because they are not registered in the saved fingerprint for that location. This threshold allows the user to see some unrecognized access points, but it will make the Verifier reject proofs where the percentage of unidentified access points becomes too high in comparison to the legitimate ones. We tested this scenario with a maximum threshold of 10%, meaning that if the fingerprint provided by the user has, for example, 10 access points and 2 of them are not in the fingerprint saved in the Verifier, the latter will assume that the user is not inside of the building.

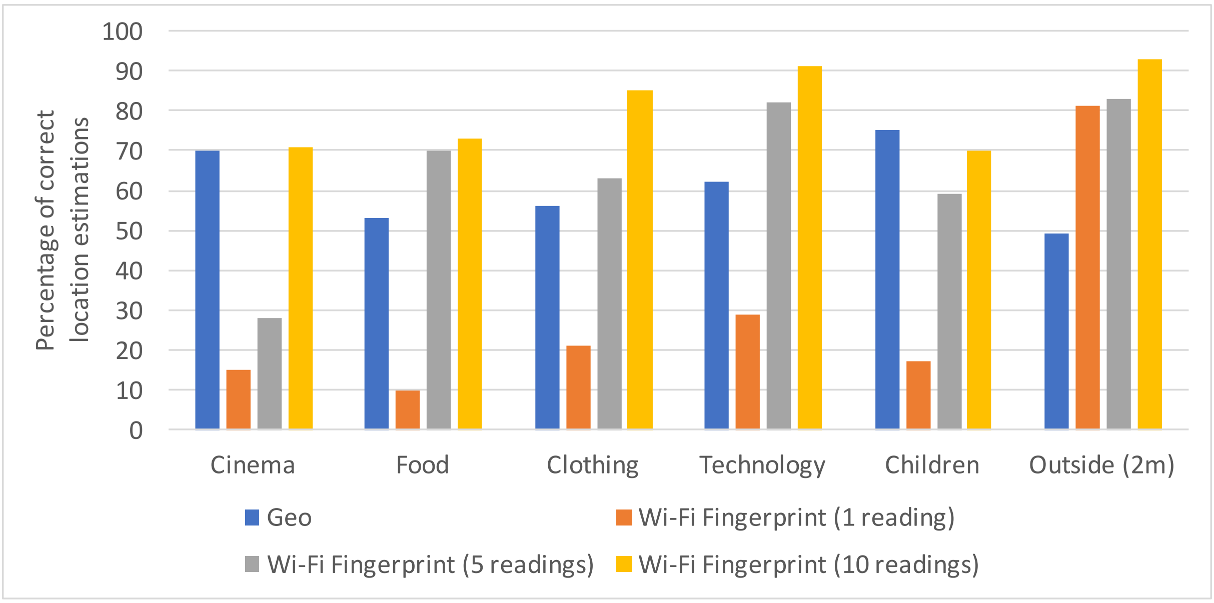


Figure 6.2: Percentage of successful readings for Geo and Wi-Fi Fingerprinting.

	Sushi Beacon	Vegan Beacon	Pizza Beacon
Correct Claims	85%	72%	81%
Wrong Claims	15%	28%	19%

Table 6.1: Location estimation accuracy with beacons.

6.1.2 Beacon

For Beacon Sensing Proofs, we had three *Estimote* beacons available. We wanted to evaluate if the beacons would have enough location precision to identify smaller areas inside the different zones presented in Figure 6.1(a). To obtain the location estimation accuracy with beacons, we prepared a scenario as in Figure 6.3. Each beacon is 5 meters away from the previous and in a different room. Tests were done with the user standing at 1 meter from the beacon. Table 6.1 summarizes the results. We made 50 location estimations near each beacon. The beacon in the Vegan restaurant has the lowest accuracy since it was in the middle of the two other beacons. We got a success rate of approximately 80% after the analysis of the three beacons. This means that beacons will provide a correct location 80% of the times when separated by 5 meters.

6.2 Time Evaluation

We performed two different studies regarding time evaluation, to answer to the following questions:

- How much time did each location estimation technique took to obtain the necessary location data?
- How much time did it take from the moment when the Prover requests a service to the moment when it receives information about the acceptance or rejection of the Proof?

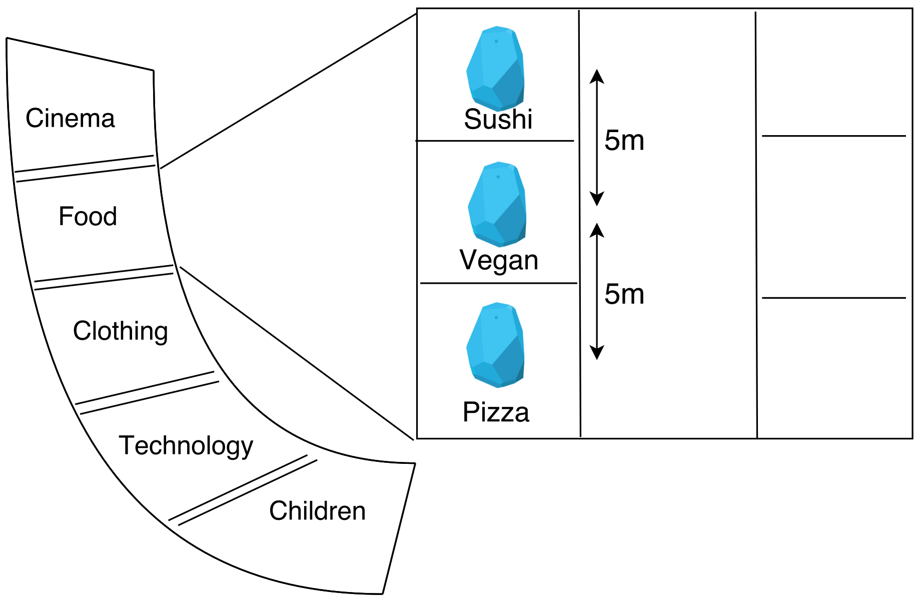


Figure 6.3: Test scenario with beacons.

6.2.1 Location Estimation Time Evaluation

We compared the three proof techniques regarding the time that each one took to complete. Figure 6.4 represents the time that was needed to obtain the location of the user. We can observe that geographic coordinates were the fastest to be collected. However, the mean value represented for Geo proofs is far from the median value. Half of the measurements were done below 1.61 seconds. The average and median are apart because we defined that an obtained geographic location [22] must have an accuracy lower than a threshold defined to be 20 meters, given the shape of the building in the experiment. This threshold should be adapted for the specific application. In very large areas, a larger threshold would be acceptable, because the user is probably still inside the area. In smaller areas, smaller thresholds have to be used. It may sometimes take longer to achieve this accuracy threshold, which increases the average time of this proof technique, but not the median.

The Wi-Fi Fingerprint location technique is the one with the most regular readings regarding time spent. Mean and median values were practically the same. In the Wi-Fi fingerprint proof, we did 10 Wi-Fi readings to obtain a fingerprint, since it was the Wi-Fi fingerprinting measure with higher location estimation accuracy (as seen in Section 6.1.1).

In Beacon proofs, the *Beacon Manager* [23] can take sometimes longer to connect or to discover the closest beacon, which helps to increase the time needed to obtain the location data. For the beacon value we used the first value read by the *Beacon Manager* as recommended by the technical guidelines of *Estimote* Beacons.

[22]Location Object in Android: `https://developer.android.com/reference/android/location/Location.html`
[23]The *Beacon Manager* is responsible for the interaction with the beacons: `https://estimote.github.io/Android-SDK/JavaDocs/com/estimote/sdk/BeaconManager.html`

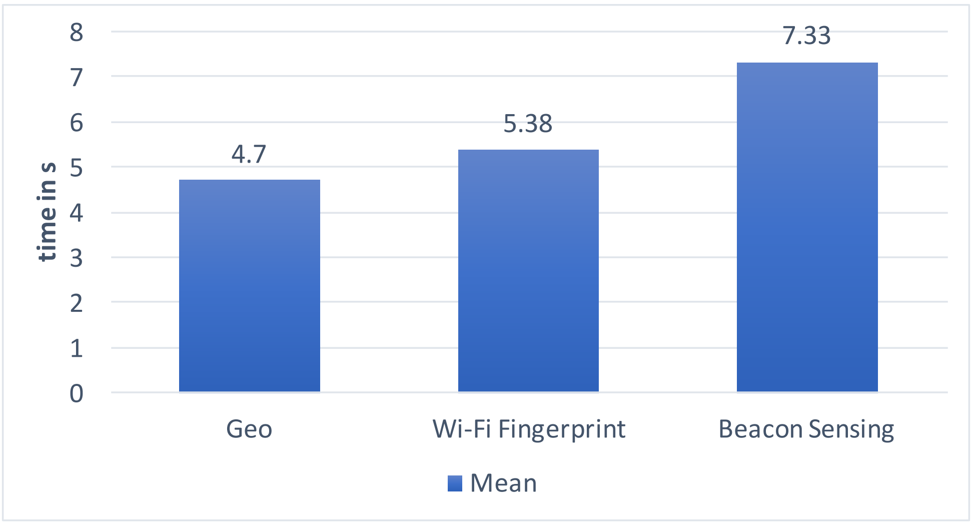


Figure 6.4: Time to get Location Data.

6.2.2 Time Evaluation of Witness Models

We conducted an experiment to compare the different witness models regarding the time that was needed to receive a proof acceptance or rejection state from the Verifier. This time evaluation begins at the moment that the Prover sends a Proof Demand request to the Verifier. We will present the different measures obtained depending on the Witness model that is used. These measures were obtained after 30 proof generations, to obtain a significant number of samples.

Master and Mobile Witness

The Master and the Mobile Witness model share many similarities regarding time evaluation, since in our setup we only have one Witness nearby the Prover. Nevertheless, the main difference between the two models is that in the Master Witness model, the Prover has to find a witness with a Bluetooth address within the list received by the Verifier. In the Mobile Witness model, the Prover only has to find a given number of random witnesses. We only have one witness in our setup when testing each model, which will result in similar time measurements for both models. However, in real world scenarios is expected that a Master Witness can take a little bit longer to be found, since the Prover is looking for a specific subset of addresses.

Figure 6.5 demonstrates the total time, from the moment when the Prover asks for a proof demand to the Verifier, to the moment when the Verifier informs the Prover about the acceptance or rejection of the proof. We can see that higher the location accuracy, the more time will be needed to generate a proof. Most of the time is spent on obtaining location data and this is what differentiates the total time between techniques. For example, in the Wi-Fi fingerprint, 10.76ms are spent in obtaining location information (5.38ms in each device). The lookup for witnesses occurs in parallel with the gathering of location data. Each technique spends approximately between 4.5 and 5 seconds in processes other than obtaining location data. These other processes are: the signing of the proof (112ms), the verification of the proof by the Verifier (85ms, approximately the same time regardless of the used location technique) and the

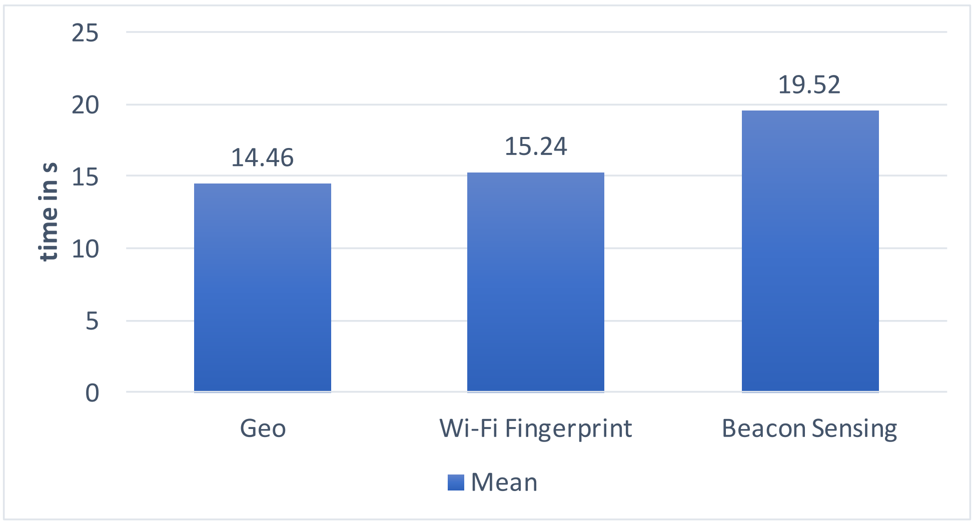


Figure 6.5: Time spent from requesting proof demand until the Verifier informing the Prover about the proof's acceptance or rejection (Master and Mobile Witness Model).

establishment of Bluetooth connections between devices and communication between entities. The time overhead in the Beacon Proof is also caused due to the initialization of the *Beacon Manager*, the object responsible for the interaction with the beacons. This connection took, in average, from 500ms to 1000ms in each device.

Self Witness

The Self Witness model is a last resort model that the Verifier can use when is offering services with less value and when there are scenarios where is probable that no witnesses are found by the Prover. In this model there is no communication between Prover and another Witness, since the first will act as its own Witness. This reduces the time needed to generate a proof, since there will be only one location estimation and there are no communications via Bluetooth. The mean time spent for each proof technique was close to half of the time spent in the Master and Mobile Witness model.

6.3 Discussion

We evaluated three different types of location estimation: geographic coordinates, by using the ANLP; a grid of Wi-Fi fingerprints constructed by the application developer; and the use of Bluetooth beacons spread over the area. As expected, the beacons gave the highest accuracy, even if measuring smaller areas. Wi-Fi fingerprint with 10 readings showed the ability to distinguish between areas more precisely than geographic coordinates, but it requires a more careful setup phase, where a grid of Wi-Fi fingerprints has to be constructed. Our Wi-Fi fingerprint map can be divided as the application developer and the service provider want. We showed an evaluation test in which we try to divide the shopping center in five areas to compare its results against the geographic coordinates provided by the ANLP. Our Wi-Fi fingerprint location estimation accuracy is above 70% for our evaluation setup, but it can be used in larger or smaller areas. However, its accuracy should decrease when the areas to identify are smaller.

Regarding proof time, the more accurate the location technique is, the more time is needed to generate a proof. One time constraint that can affect the feasibility of the system is the time that Prover and Witness are in the Bluetooth range of each other. If the Witness takes too long to generate a proof, the Prover can now be too far to receive it. We think that in the shopping center example, our time measurements are enough to claim the feasibility of our system. For example, in a food court users tend to walk slowly or be sitting, which increases the probability of Prover and Witness staying in range of each other. In our tests, we only used one Witness. However, the total time should not increase linearly with the number of Witnesses that received a proof request since they work in parallel to one another.

Given the accuracy that our location estimation techniques achieve, we propose that the three of them can be used simultaneously. In our example, the ANLP (Geo Proofs) can be used to detect when the user enters the shopping center. It does not indicate an exact zone of the shopping but by knowing that the user is in the shopping, the other two location estimation techniques can be activated. If the Wi-Fi fingerprint indicates that the user is in a zone with Bluetooth beacons, the user's smartphone can try to detect the closer beacon, so that the system can infer the store where the user is. The location estimation techniques are complementary to one another and can be used together.

6.4 Collusion Avoidance Simulation

We wanted to know if in a real environment our collusion avoidance mechanisms would not compromise the normal behavior of our system. We simulated the shopping center using *Netlogo*[24], a multi-agent programmable simulating environment. Figure 6.6 presents a screen capture of our simulation environment. Grey areas represent the stores, white areas represent the corridors, black represents walls and the blue dots are the users of SureThing. This simulation has the following main goals:

- Identify if our collusion avoidance mechanisms would not create a barrier for SureThing's normal output. The collusion avoidance mechanisms should not make the system deny proofs from legal Provers and Witnesses;
- Detect if the system can discover dishonest Provers and Witnesses colluding with each other and how much time it takes to detect it;
- Evaluate how many Witnesses a user finds each time it requests for a proof;
- Study the probability of a Prover finding the same Witness multiple times, which will cause a decay in the value of the proof.

6.4.1 Simulation Setup

Our environment is represented as a grid with two coordinates. The horizontal coordinate goes from 0 to 255 and the vertical goes from 0 to 127. Each cell can only be occupied by one user. There are

[24]Netlogo: `https://ccl.northwestern.edu/netlogo/`

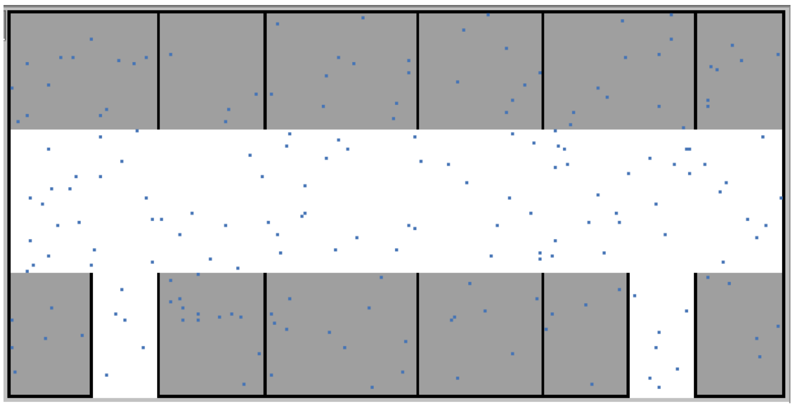


Figure 6.6: Shopping center simulated in *Netlogo*.

250 users in our simulation. *Netlogo* provides a *tick* counter, that is equivalent to a time counter. Users may only move one cell per tick. Users move randomly around the shopping center. However, while in a corridor, they have a 70% probability of moving forward in each tick and only a 15% probability of changing direction. Inside a store, the moving probability decreases to 50% and the rotating one increases to 40%. Although it is not an exact model of how people walk in a shopping center, these probabilities help to decrease the randomness in the movement of the users.

At each tick of the simulation, each user has a 1% probability of requesting a proof. The Witnesses can be at a maximum distance of 10 cells from the requesting user.After 10 ticks, the Witnesses that remained nearby the Prover will respond to him. This simulates the time that Prover and Witness take when changing data about the proof via Bluetooth. We used Formula 4.1 defined in Section 4.4 to determine the value of proofs given by the Witnesses. The parameter *k* was defined as 3 so we can have a fast decay. We defined the maximum value of each proof as 1. In this simulation, the Verifier accepts batches of proofs with at least value of 1, which means that if a Prover receives a proof from a Witness never before seen, the Verifier will accept the claim. A proof batch is a group of proofs provided by the Witnesses that the Prover contacted.

6.4.2 Simulation Results

We ran the experiment with three breakpoints: after the request of 50, 100 and 500 proof batches. Each proof batch is generated when the Prover receives a group of proofs from Witnesses. We analyzed these different breakpoints so that we can observe how the system behaves in different situations. For example, after 500 proof batches are generated, it is expected that the provers are starting to find more repeated Witnesses than after the generation of only 100 proof batches. The number of cases where no Witnesses are found remains constant and near to 20%. In the other cases, proof batches are accepted or rejected. Results are summarized in Figure 6.7. The number of proofs accepted decreases when there are more requests. This is caused by the Witness Decay since after a while Witnesses start to be repeated and their value decreases as defined in Equation 4.1. There are also no changes in the population, since the 250 users are always the same. In a real-world environment we can find more dynamic systems, which will benefit our solution.

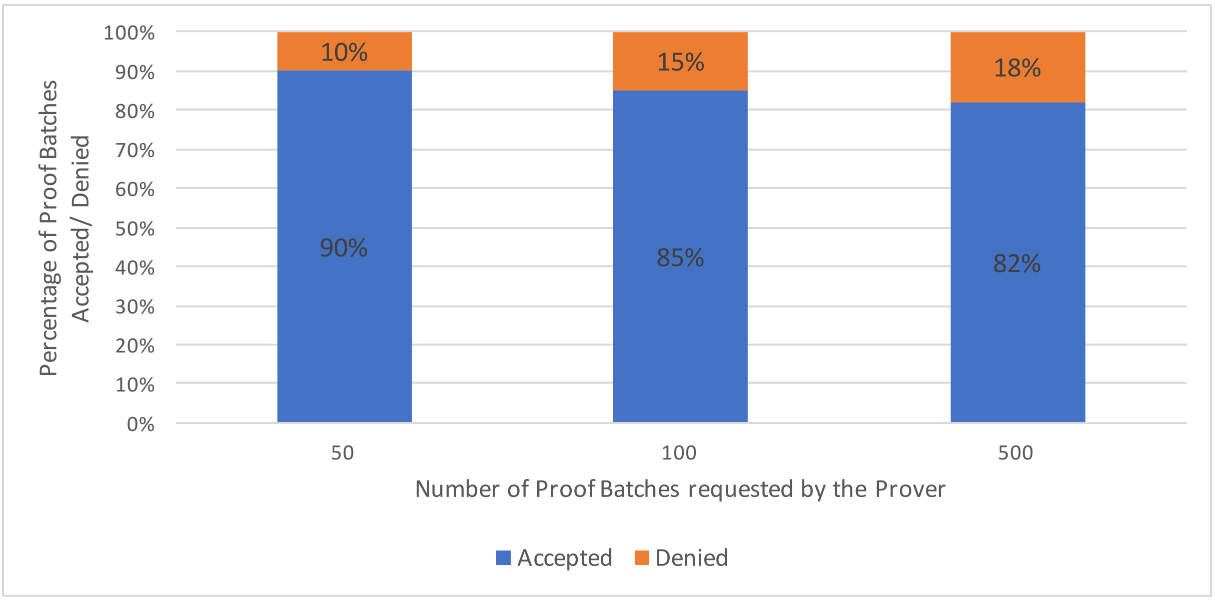


Figure 6.7: Acceptance and Denial rate of the Proof Batches.

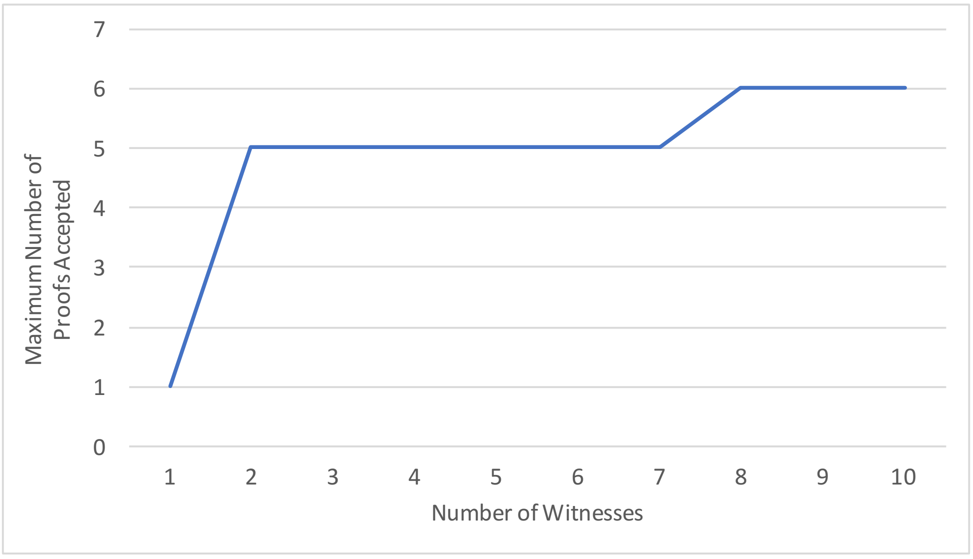


Figure 6.8: Maximum quantity of Proof Batches accepted when the Prover is always repeating the same Witnesses.

We also analyzed the average number of Witnesses found per request. With our described setup, with 250 users, there was an average of only 2 Witnesses found. If we increased the population to 1000 users, the number of Witnesses increases to 7. This demonstrates that depending on the number of users in the space, the Verifier can request proofs from a different number of Witnesses.

If a user wants to deceive the system, he will have to collude with false Witnesses. If the Verifier asks for proofs from N different Witnesses (*Witness Redundancy*), the user will have to gather proofs from N dishonest Witnesses. As we showed before, in a setup with 250 users, provers usually only find 2 Witnesses, which may not be too difficult for the malicious user to gather.

Our *Witness Decay* mechanism guarantees that the dishonest Prover cannot re-use the same Witnesses too much. Figure 6.8 shows the quantity of Proof Batches that are accepted when a malicious

Prover is always getting proofs from the same Witnesses. We can see that, for example, when a Prover is getting proofs from the two same witnesses over time, the Verifier will start rejecting the proofs after the fifth batch is sent. For these calculations we used the same setup as in Section 6.4.1. We show that if a Prover always sends a batch of proofs containing the same Witnesses, the Verifier will not accept this batch more than 6 times. From the seventh time that a Witness generates a proof, its value is 0, so the Verifier will never accept the batch. The Prover is obliged to renew its Witnesses since the old ones no longer have any value.

6.4.3 Simulation Summary

Following the results of the simulation, we conclude that SureThing will be able to prevent collusions in a real world scenario by taking advantage of the diversity of Witnesses. The decay of a proof given by the same Witness to the same Prover will deny the access to the service. SureThing can also adapt to the number of users in the place and take advantage of it. It uses the total number of users to calculate the probability of a Prover encountering the same Witness and adjusts the proof decay accordingly.

Chapter 7

Conclusion

We presented SureThing, a solution that allows service providers to have more certainty about the location claims made by users. Users have to provide location proofs in order to gain access to valuable services provided by other entities. With the growth in the development of location-based applications, we envision that location proofing will be crucial.

After analyzing different works related with location *estimation* and *proofing*, we decided to create a flexible solution regarding these two features. SureThing allows the application developers to determine the location of users by using geographical coordinates (with GPS or the ANLP), Wi-Fi fingerprinting or Bluetooth Beacons. Although we are using one location estimation technique at a time, we have shown that these techniques can work together e.g. GPS detects when the user enters in the shopping center, Wi-Fi fingerprinting detects the zone of the center and the Bluetooth Beacon indicates the store where the user is. We have prepared the technical work that will allow this combination of techniques.

SureThing uses a witness-based approach to prove the location of a user. We proposed three different Witness models: Master, Mobile and Self Witness. We wanted to construct a solution that can adapt and fulfill the needs to the maximum number of scenarios possible and we think that this separation is an important feature of SureThing. It gives the opportunity to the application developer to choose the best model for his application, since each model has its advantages and disadvantages. When using random mobile witnesses, SureThing also offers two different collusion avoidance mechanisms, based on the number of witnesses and on the value of each proof.

We implemented a prototype of SureThing, by developing a mobile application library, a service provider (Verifier) and a Certification Authority. The mobile library can be used in any Android application while the other two entities act as RESTful web services that can also be reused. We performed an evaluation of our prototype, regarding location estimation accuracy, response times and feasibility of proof exchanges between users. We conducted a simulation to analyze how the collusion avoidance mechanisms behave and we had promising results. We conclude that our solution can be a valid option for developing location-based mobile application with stronger assurances about the real location of a user.

Regarding future work, SureThing can be extended in many ways. We think that the next most important step is to increase the level of privacy and security. Since users often share their locations, it is important to protect the privacy of the users. The use of random changing pseudonyms is being used in similar systems and can be added to SureThing, to provide stronger privacy protection. The communication between entities is also not protected from eavesdropping. The attacker cannot change the content of a proof, since it is signed by the Witness, but he can listen to its content. If applications using SureThing want to ensure confidentiality in the exchange of messages, our system will have to adopt cryptographic methods to protect communication.

SureThing would also take advantage of the implementation of proximity verification methods, to guarantee that Prover and Witness are really near each other [30]. The renewal of digital certificates is not incorporated in the current prototype of SureThing and it is also a future need.

We defined a base implementation to SureThing but we envision that our solution can be extended, for example, to allow new location estimation techniques such as Ambient Fingerprinting [21]. Our base for SureThing allows that new developed location estimation techniques can be easily included in a location proof. Currently SureThing is useful for mobile applications. We envision that it will also be useful in proofs done by devices other than smartphones, in IoT use cases.

By using a witness-based approach where Witnesses are fundamental to the success of SureThing, it is important to reward them when they help other users. While this feeling of community already exists in SureThing, we envision that an incentive scheme engine (e.g. gamification) will help developing the community.

The proofs that the Verifier receives are not being stored and this will be a need for SureThing in the future. The Verifier may want to revisit a past proof or it may want to analyze the proofs that were previously sent, for example, to detect possible collusions. In future developments of SureThing, the persistence of proofs is a key feature to allow the inspection of past proofs.

We proposed a first approach to deal with collusions and evaluated it with simulations. Actual deployments will further validate the effectiveness of the mechanism.

Bibliography

[1] D. Uckelmann, M. Harrison, and F. Michahelles. An architectural approach towards the future internet of things. In *Architecting the internet of things*, pages 1–24. Springer, 2011.

[2] M.-O. Pahl, G. Carle, and G. Klinker. Distributed smart space orchestration. In *Network Operations and Management Symposium 2016 (NOMS 2016) - Dissertation Digest*, May 2016.

[3] S. M. Coelho and M. L. Pardal. Smart Places: A framework to develop proximity-based mobile applications. In *INForum*, Lisboa, Portugal, September 2016.

[4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[5] D. Guinard and V. Trifa. *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co., 2016.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[7] L. Fridman, S. Weber, R. Greenstadt, and M. Kam. Active authentication on mobile devices via stylometry, application usage, web browsing, and gps location. *IEEE Systems Journal*, 11(2):513–521, 2017.

[8] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, 2001.

[9] H. Koyuncu and S. H. Yang. A survey of indoor positioning and object locating systems. *IJCSNS International Journal of Computer Science and Network Security*, 10(5):121–128, 2010.

[10] S. Nirjon, J. Liu, G. DeJean, B. Priyantha, Y. Jin, and T. Hart. Coin-gps: indoor localization from direct gps receiving. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 301–314. ACM, 2014.

[11] R. Hansen and B. Thomsen. Efficient and accurate wlan positioning with weighted graphs. In *International Conference on Mobile Lightweight Wireless Systems*, pages 372–386. Springer, 2009.

[12] S. Xia, Y. Liu, G. Yuan, M. Zhu, and Z. Wang. Indoor fingerprint positioning based on wi-fi: An overview. *ISPRS International Journal of Geo-Information*, 6(5):135, 2017.

[13] A. T. Mariakakis, S. Sen, J. Lee, and K.-H. Kim. Sail: Single access point-based indoor localization. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 315–328. ACM, 2014.

[14] A. R. Pratama, R. Hidayat, et al. Smartphone-based pedestrian dead reckoning as an indoor positioning system. In *System Engineering and Technology (ICSET), 2012 International Conference on*, pages 1–6. IEEE, 2012.

[15] M. Ibrahim and M. Youssef. Cellsense: A probabilistic rssi-based gsm positioning system. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[16] A. Baniukevic, C. S. Jensen, and H. Lu. Hybrid indoor positioning with wi-fi and bluetooth: Architecture and performance. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, volume 1, pages 207–216. IEEE, 2013.

[17] G. Anastasi, R. Bandelloni, M. Conti, F. Delmastro, E. Gregori, and G. Mainetto. Experimenting an indoor bluetooth-based positioning service. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 480–483. IEEE, 2003.

[18] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[19] M. Koühne and J. Sieck. Location-based services with ibeacon technology. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2014 2nd International Conference on*, pages 315–321. IEEE, 2014.

[20] E. Costa-Montenegro, F. J. González-Castaño, D. Conde-Lagoa, A. B. Barragáns-Martínez, P. S. Rodríguez-Hernández, and F. Gil-Castiñeira. Qr-maps: An efficient tool for indoor user location based on qr-codes and google maps. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 928–932. IEEE, 2011.

[21] M. Azizyan, I. Constandache, and R. Roy Choudhury. Surroundsense: Mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pages 261–272, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-702-8.

[22] S. Saroiu and A. Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, page 3. ACM, 2009.

[23] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *Proceedings of the 2nd ACM workshop on Wireless security*, pages 1–10. ACM, 2003.

[24] R. Khan, S. Zawoad, M. M. Haque, and R. Hasan. Who, when, and where? location proof assertion for mobile devices. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 146–162. Springer, 2014.

[25] Z. Zhu and G. Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *INFOCOM, 2011 Proceedings IEEE*, pages 1889–1897. IEEE, 2011.

[26] E. S. Canlar, M. Conti, B. Crispo, and R. Di Pietro. Crepuscolo: A collusion resistant privacy preserving location verification system. In *Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on*, pages 1–9. IEEE, 2013.

[27] X. Wang, A. Pande, J. Zhu, and P. Mohapatra. Stamp: enabling privacy-preserving location proofs for mobile users. *IEEE/ACM Transactions on Networking*, 24(6):3276–3289, 2016.

[28] B. Kannhavong, H. Nakayama, and A. Jamalipour. Nis01-2: A collusion attack against olsr-based mobile ad hoc networks. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–5. IEEE, 2006.

[29] R. Khan, S. Zawoad, M. M. Haque, and R. Hasan. Otit: Towards secure provenance modeling for location proofs. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 87–98. ACM, 2014.

[30] A. Ranganathan and S. Capkun. Are we really close? verifying proximity in wireless systems. *IEEE Security Privacy*, 15(3):52–58, 2017. ISSN 1540-7993.