Designing a Privacy-Aware Location Proof Architecture

by

Wanying Luo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Although location-based applications have existed for several years, verifying the correctness of a user's claimed location is a challenge that has only recently gained attention in the research community. Existing architectures for the generation and verification of such location proofs have limited flexibility. For example, they do not support the proactive gathering of location proofs, where, at the time of acquiring a location proof, a user does not yet know for which application or service she will use this proof. Supporting proactive location proofs is challenging because these proofs might enable proof issuers to track a user or they might violate a user's location privacy by revealing more information about a user's location than strictly necessary to an application. In addition, none of the existing architectures possesses an effective cheat detection mechanism to spot users who cheat about their location. We present seven essential design goals that a flexible location proof architecture should meet. Furthermore, we introduce a lightweight location proof architecture that realizes a subset of our design goals and that includes user anonymity and location privacy as key design components, as opposed to previous proposals. We then present a complete architecture that meets all of the design goals and demonstrate how some of the design goals can be achieved by adopting proper cryptographic techniques. Note that the reason of having a lightweight architecture that meets a subset of our design goals is explained in section 2.4.6. Finally, we provide an implementation, experimental results and a deployment strategy of our location proof architecture, and present three real-world location-proof-based applications to further demonstrate the practicality of our architecture.

Acknowledgements

There are many people to thank, and my family come first. I would like to thank my mother and my twin brother for their love and support. My mother is the bravest woman I have ever seen. She gave and continues to give me the strength and courage to fulfill my dream. My brother is the man I respect and trust the most. He understands and protects me all along. Without him, I will not be the person who I am today. I love and thank them from the bottom of my heart. I would also like to thank Lu Zhang, a wonderful woman with whom I am thrilled to spend the rest of my life. Computer science is a joyous career, but it is my family that really matter to me.

I would like to thank my supervisor Urs Hengartner. He guided me throughout my research and studies with great patience. I learnt so much from him. This thesis will not be possible without his great help.

I would like to thank Dr. Ian Goldberg and Dr. Srinivasan Keshav for reading my thesis. Their comments are extremely helpful in improving my thesis.

Last but not least, I would like to thank Nabeel Ahmed for providing me with a wireless testbed for conducting experiments.

Dedication

This is dedicated to my mother and my twin brother.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Mobile devices, such as smartphones and PDAs, are playing an increasingly important role in people's lives. Location-based applications take advantage of user location information and provide mobile users with a unique style of resource and service offerings. Some mobile application and service providers decide whether to grant people access to the service solely based on people's current locations. Many location-based service (LBS) providers grant access of service only to people present at a certain location. Today, it is typically a user's mobile device that determines the device's location (e.g., using GPS) and that sends the location to an application or service provider. This approach makes it possible for a user to cheat by having his device transmit a fake location. This way the user might be granted access to a resource erroneously. Therefore, an application or a service provider might ask a device to prove that the device really is or was at the claimed location. For example, a hospital might limit access to patient information to doctors and nurses who can prove that they are in (maybe a particular area of) the hospital. Content that is generated by mobile users might be geotagged, and people who download the content should be able to verify the location claim associated with the content [21]. A company might want proof from its repairmen that they really followed a prescribed route during the day. An insurance company might hand out discounts to drivers who can prove that they take non-accident-prone routes for their daily commutes. A person accused of committing a crime is very much interested in being able to prove to the police that he was somewhere other than the crime scene at the time the crime was committed [26]. The following is a list of more location proof based applications.

- **Store loyal customers**: Some stores offer discounts to nearby customer. In order to prove to be a frequent customer of the store, a person can collect location proofs whenever she is nearby the store and present them to the store after having accumulated enough proofs to get a discount.

- **Location-based access control**: Role-based access control is not always sufficient in restricting access to confidential information. Location proof is promising in providing an industrial solution for data leakage protection (DLP). Companies storing confidential information and crucial corporate secrets typically forbid their employees from accessing the company's database anywhere other than inside the company building, in fear that employees may deliberately or accidentally leak sensitive information. The company can set up a few access points (APs) around its premises, allowing employees to collect location proofs from an AP and submit proofs to the authentication server of the company in order to prove they are currently present in the working site.

- **Location-based content delivery**: Unlike location-based access control, some applications or service providers may only be concerned about users' location and not care about their identities. For example, a stadium owner decides only those inside the stadium are allowed to receive the live broadcast. Some Canadian location-based media provider may deliver media programs only to those within Canadian territory. In this case, a Canadian resident needs a way to prove her location to the provider and the provider also needs a way to verify the proof.

- **Location-based social networking**: Online social networks allow users to form social groups based on their geographical location. Yelp [36], Foursquare [13] and Gowalla [16] are all online social networks that allow users with mobile devices to "check-in" and learn their friends' location. Users could potentially use location proof technology as an alternative to "checking-in" through online social networks' portal. The most attractive benefit of this approach is the protection of their location privacy from online social networks, as demonstrated by an application we developed in section 4.4.3.

Saroiu et al. [26] and Lenders et al. [21] list more applications based on location proof in their papers. In a word, we need an architecture to allow legitimate users to prove their location, and to also enable applications and service providers to verify these location claims.

1.2 Location Proofs and Location Proof Architecture

Formally, a *location proof* is an electronic form of document that certifies someone's presence at a certain location at some point in time. A *location proof architecture* is a mechanism with which users can obtain location proofs from proof issuers and with which applications can verify validity of the proofs.

In order to be truly useful, a location proof architecture must be flexible in that users should be able to collect location proofs for both current and future needs, and the same location proof should be usable for interaction with different applications. For example, in some cases such as the insurance or police scenarios mentioned above, users might not know while being at a particular location that they will need a proof for having been at this location later on. Therefore, it must be flexible enough for users to gather location proofs proactively. This way, users with foresight are able to obtain and save location proofs before they even have a target application to interact with. However, the proactive gathering of location proofs must be done carefully; otherwise, proof issuers can track users and people's privacy will be in jeopardy. Besides, proactive location proofs are optional, and users can always request a proof when it is really needed. Different applications have different requirements for a location proof, such as the granularity of the location. For example, an insurance company might want to know only that a client drives around mainly in sedate Waterloo (as opposed to busy Toronto), but not where exactly in Waterloo. When a user does not know about the application that a location proof will be used for, she also does not know about the location granularity that will be required by the application. Users can always collection a location proof every time they want to interact with an application that requires a proof, but this is wasteful of users' time and computing resources, since there are so many LBSes and applications. Then the question is how to make a single location proof usable for different applications that have different location requirements. Including the most fine-grained location information in any location proof would solve this problem. However, a user might end up revealing more information than necessary to an application, and her privacy would get violated. Therefore, a useful location proof architecture must be designed in the way that users could use a small number of location proofs to interact with a large collection of mobile applications and services, at the same time, their location privacy is also preserved. Location privacy means the user should disclose her location information only sufficient enough to interact with mobile applications and she should not unnecessarily disclose more location information than strictly necessary.

Cheating detection is also a vital requirement of location proof technology in some cases. Mobile users typically have incentives to lie about their location. Therefore, a location proof architecture should adopt a cheating detection mechanism to pinpoint dishonest users who lied about their location information. This is different from the use of cryptography in location proof systems to prevent dishonest users from manipulating issued proofs. Even if the integrity of the issued location proofs is protected by careful use of cryptographic techniques, users may simply get around cryptographic protection by lying about their identity or location information when trying to obtain proofs from proof issuers.

Previous research has recognized the need for location proof architecture (e.g., [26, 21]).

However, none of the existing architectures is flexible enough to support all envisioned use cases. We make the following contributions:

- We put forward seven design goals for a flexible location proof architecture and introduce a lightweight location proof architecture that realizes a subset of our design goals. The architecture can be deployed on existing Wi-Fi access points (APs) and is usable by regular mobile users. Moreover, the architecture demonstrates how cryptography can improve user privacy and system security. After identifying limitations of the first architecture and drawing lessons from it, we present a more complete location proof architecture that meets all the design goals.

- We present a complete location proof architecture to address the design goals not covered by the lightweight architecture in addition to the design goals which are already met in the lightweight architecture. Different from most existing work that designs location proofs to be application-specific, retroactive and vulnerable to user cheating, our complete architecture constructs general location proofs so that they can be used for a variety of applications. Moreover, our complete architecture possesses a cheating detection mechanism capable of exposing users who lie about their location.

- We implement the complete location proof architecture and provide experimental results regarding the performance of the architecture. Our implementation is deployed on a WLAN testbed, which consists of 38 APs and spans two floors of the Davis Centre at the University of Waterloo.

- We build two real-world applications to demonstrate the practical value of our architecture.

Chapter 2

Lightweight Architecture

2.1 Introduction

In this chapter, we elaborate four of our seven design goals that are realized by a lightweight location proof architecture. This chapter is organized as follows. Four of our design goals are specified in section 2.2, which is followed by security assumptions and the threat model in section 2.3. We proceed to elaborate on a lightweight location proof architecture and security analysis in section 2.4.

2.2 Design Goals

In order for a location proof architecture to be deployable and useful in practice, the design should meet certain requirements. In this section, we elaborate four of our design goals which are realized in the first architecture we develop, and bring up another three design goals as we introduce a more complete architecture.

2.2.1 No Dedicated Hardware

One of the most significant limitations in many previous proposals on location proof architecture is reliance on dedicated hardware. A mobile user is often equipped with no more than a regular cellphone, so any special hardware requirement will prevent her from adopting location proof technology. Moreover, it is unrealistic to expect operators of existing APs to purchase and configure probably expensive hardware to provide location proof services.

2.2.2 Scalability

A public AP usually serves many users, so if a location proof protocol executed between the AP and the user demands heavy computation, the performance will decline dramatically and the architecture will not scale with increasing number of users. Furthermore, mobile devices in most users' hands have less processing power and memory than a desktop computer, and the functioning time is also limited by battery capacity. Even though mobile devices are becoming comparable to normal desktop computers in terms of speed and memory space, mobile users will be frustrated if requesting location proofs consumes too much time and interferes with their normal use of mobile devices. Therefore, a location proof protocol must not require intensive computation in order to be scalable on these kinds of devices.

2.2.3 User Anonymity

One important factor that many users are concerned with regarding new technologies is whether their privacy will be undermined by using the new technology [9, 24]. Therefore, in order to be widely adopted, location proof architectures must invest every effort to guarantee users' location privacy. This problem is somewhat challenging in that it poses a dilemma to the architecture design: in order to vouch for a person's presence at a certain place, a location proof issuer must know who it is vouching for; however, by knowing the identity of the person it is vouching for, the location proof issuer can track that person by her location, thus compromising her privacy. Location proof architectures should employ appropriate cryptographic techniques and design strategies to ensure users remain anonymous to proof issuers.

2.2.4 Application-agnostic Location Proofs

A number of mobile applications are available nowadays, and each location proof based application makes use of location proofs submitted by users in different ways. If a single location proof can only be used to interact with a specific application, when a user wants to prove her location to many applications, she has to try to obtain several location proofs, each of which is intended for a particular application. This approach is adopted by one previous location proof mechanism [35]. Another option is to have location proof issuers produce proofs in an application-agnostic manner such that the same proof can be repeatedly used by users for interaction with a variety of applications. In this case, the location proof issuer at a particular place only needs to hand out a proof to a user once, and the user is able to use the same proof for many applications. Our design follows the second approach, because it is obviously advantageous over the first one in many aspects.

First, a general location proof reduces storage space. Second, a general proof format relieves users of having to constantly interact with location proof issuers, and allows them to spend more battery power and time on other businesses. In order for a location proof to be application-agnostic, it must not contain application-specific data and has to be issued by a location proof issuer that applications trust.

2.3 Trust Assumptions and Threat Model

2.3.1 Trust Assumptions

If a mobile service provider writes its own application, the application can easily obtain users' location information from the service provider, which defeats our privacy protection mechanism. In addition, if an application is authorized by the mobile service provider to access users' location information, users' location privacy is also compromised. Therefore, our architecture targets those applications that are not written by mobile service providers and not authorized by mobile service providers to access users' location information.

We assume every mobile user has a public key and the public key is certified by a Certificate Authority (CA) in a PKI. Moreover, the user and the application she interacts with agree to trust a common CA which certified the user's public key. Although obtaining a public key certificate can be expensive, it is not unbearable. For example, for applications that only require e-mail addresses as identities, users can obtain a digital ID [18] from VeriSign for less than $20 per year. Belgium actually has decided to issue an electronic identity (eID) card [6] to all its citizens aged 12 years or older and the card contains a chip to be used for digital signatures. In this case, the PKI infrastructure is set up in the way that the entire nation has a common trusted CA and everyone's key is certified by this CA. We further assume no user will give away her private key to anyone else. This assumption is essential in preventing attacks discussed in section 2.4.4. If a user has her their public key certified, she not only can use the key for obtaining location proofs, but can also use the key for other purposes such as signing a document or proving her identity in a secure communication. If the user gives away her private key to other people, she essentially grants other people the ability to carry out these tasks on her behalf, which may result in unexpected consequences. For example, other people could use the user's private key to sign an illegal document and get the user into legal trouble. Furthermore, we assume the device carrier is the actual owner of the device. People nowadays store valuable personal information on their mobile devices. Such information includes contact information of friends and family members, private text messaging with spouses, appointments with doctors or lawyers, etc. With such important and private data stored on the device, users will be very reluctant to lend their mobile devices to others even for a short period of time. Finally, we assume users have their devices carried with them when location proofs are requested. Mobile services are becoming essential for people's daily communication and businesses. For example, a mobile user can look up a friend's phone number conveniently on her device, or receive live weather forecasts, or immediately call for help in case of emergency. Therefore, people are much better off by carrying their mobile devices all the time. Moreover, mobile devices are significantly easier for people to carry than a cumbersome desktop computer. Many university students bring laptops to class on a daily basis, not to mention carry a BlackBerry or iPhone which fits in a pocket.

We assume that applications are in the possession of the required public keys when carrying out proof verification. Furthermore, we assume users submit location proofs for verification when they need to use the application. Finally, we assume communication is secured against passive and active eavesdroppers with the help of TLS/SSL.

2.3.2 Threats That We Address

The following entities are considered threats in our architecture.

- **Dishonest users**. A dishonest user is someone who tries to obtain location proofs that certify her presence at some place at a particular time even if she was not there. Dishonest users may achieve this goal by colluding with malicious intruders and defecting APs.

- **Malicious intruders**. A malicious intruder is somebody who is not interested in obtaining location proofs for her own use but offers to collude with remote dishonest users to get location proofs on their behalf in exchange for other benefits like money. For example, a dishonest user may save the generated session data when interacting with an AP and give these data to a malicious intruder, who can use these data to launch replay attacks later on. This type of attack is analyzed in section 2.4.4. However, we assume malicious intruders will not collude with dishonest users to launch wormhole attacks, which we discuss in section 2.3.3.

- **Defecting APs**. A malicious or misconfigured AP may collude with dishonest users and issue them location proofs with fake information, to assist them in cheating. For example, a malicious AP in Vancouver, which is only authorized to issue proofs representing places in Vancouver, may issue location proofs representing places in Toronto.

- **Malicious application providers**. A malicious application provider obtains location proofs from users of her service and then tries to take advantage of these proofs to get unauthorized access to other applications.

- **Active and passive eavesdropper**. An eavesdropper records and potentially modifies communication between users, proof issuers, or applications.

2.3.3 Threats That We Do Not Address

We acknowledge the following threats which exist in real wireless networks. Since current researches lack effective counter-measures against these threats, we do not aim to provide complete solutions.

- **Wormhole attacks**. A wormhole attack [5] takes place when a malicious party records network traffic in a particular region of the wireless network and replays it in another region. For example, suppose two wireless devices A and B are invisible to each other, a malicious device C within the transmission range of device A tunnels traffic from A to B, which makes device B appear visible to A. This kind of attack poses severe threat to wireless networks and is extremely hard to detect. In a location proof architecture, by launching wormhole attacks a user may collude with several remote malicious intruders to simultaneously obtain location proofs from APs at different places. In this paper, we do not aim to completely solve this problem due to its notorious challenge, but we provide a partial solution that defends against a special type of wormhole attacks as discussed in section 3.4.3. In fact, an effective way of defeating wormhole attacks is to rely on dedicated hardware and distance bounding techniques suggested by Drimer and Murdoch [11], who proposed and implemented a defense mechanism using a distance bounding protocol against smart card wormhole attacks. However, their mechanism requires alteration to current hardware and software. Since this kind of approaches violates our first design principle, we do not consider it here.

- **Weak identities**. A device carrier is not always the actual owner of the device. For example, a device may be stolen by a thief or lent to a friend by the owner. Therefore, our use of users' public keys as their identities may not be reliable. This form of user identity is deemed a weak identity. Saroiu and Wolman [26] propose several alternatives to achieve stronger identities, but none of them is entirely foolproof.

Both wormhole attacks and attacks exploiting weak identities can be instantiated in real wireless networks. Unfortunately there are no easily deployable solutions against either of them. Applications that need to protect access to highly valued resources should therefore not rely on our location proof architecture alone for security. For other applications, these threats can be ignored since the attacks tend to be expensive to instantiate, and their cost might be higher than the value of the protected resource.

2.4 Architecture for Obtaining and Verifying Location Proofs

In order to prove to the application that she was nearby some location at a particular time, a user must collect location proofs from the APs that are trusted by the application. Applications can make decisions on which APs to trust on an organizational basis. For example, applications can choose to trust APs deployed by AT&T which has APs running all over the US, or applications may trust APs run by other companies according to their own standards. Different applications have different requirements on APs. Low-valued applications may accept many APs operated by various companies, whereas high-valued applications are likely to only accept APs run by high-profile organizations. Since Wi-Fi signals typically have a short range of transmission, only nearby users are able to communicate with the APs directly. In addition, to protect users' privacy, cryptographic signatures and hashes are used to hide users' real identities from the APs.

The location proof architecture that we will present in this section is lightweight compared to the one in section 3.4 in that it incurs less communication overhead and involves fewer parties. It is specifically targeted at scenarios where protection of users' location privacy against applications is not crucial, and the benefit of lying about their location is not significant enough to give users incentives to cheat.

2.4.1 Notations

The following protocol and the one introduced in section 3.4 both extensively use cryptographic signatures. From this point on, we use $S_T(m)$ to represent message m signed with T's private key. As usual, we use a hybrid signature scheme, where a message is first cryptographically hashed and the signature is computed for the hash value.

The architecture in section 3.4 also makes use of cryptographic encryption, so we define encryption operations as follows:

$$\mathrm{Enc}\left(K,\ \mathrm{data}\right)$$

which represents data is encrypted with some party's public key K. Moreover, the encryption scheme we will use here is probabilistic [15], i.e. encrypting the same plaintext several times generally yield different ciphertexts.

2.4.2 Obtaining a Location Proof

When a user is nearby an AP, she can execute the following protocol to obtain a location proof from the AP. Note we use the user's public key as her identity, i.e. $\mathrm{ID}_{\mathrm{user}} = P_{\mathrm{user}}$, where P_{user} stands for the user's public key.

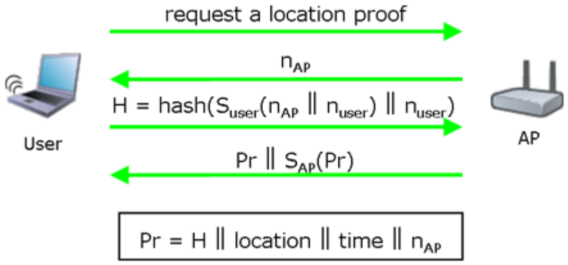

request a location proof
n_{AP}
H = hash(S$_{user}$(n$_{AP}$ ‖ n$_{user}$) ‖ n$_{user}$)
Pr ‖ S$_{AP}$(Pr)
User
AP
Pr = H ‖ location ‖ time ‖ n$_{AP}$

Figure 2.1: Request location proofs

1. (Optional) The user sends a location proof request to the AP.
2. The AP generates and sends the user a random nonce n_{AP}.
3. The user generates a random nonce n_{user} and sends the following hash value to the AP:
$$H = \mathrm{hash}\left(S_{\mathrm{user}}\left(n_{\mathrm{AP}} \parallel n_{\mathrm{user}}\right) \parallel n_{\mathrm{user}}\right)$$
Note that instead of computing the hash of an appended nonce, a MAC would also suffice. In that case, the user generates a MAC key, k, and computes $H = \mathrm{MAC}_k\left(S_{\mathrm{user}}\left(n_{\mathrm{AP}}\right)\right)$. Later, the user delivers k, instead of n_{user}, to the application for proof verification.
4. Finally, the AP sends to the user a location proof of the following format.
$$\text{location proof} = Pr \parallel S_{\mathrm{AP}}\left(Pr\right)$$
where $Pr = H \parallel \text{location} \parallel \text{time} \parallel n_{\mathrm{AP}}$

This process is demonstrated in figure 2.1. Note that the first step is optional. If the AP does not receive any user request within a certain period of time, it will start to broadcast random nonces so that users can capture broadcasted n_{AP} and begin to interact with the AP starting from step three.

2.4.3 Verifying a Location Proof

The user submits n_{AP}, n_{user}, $S_{\mathrm{user}}\left(n_{\mathrm{AP}} \parallel n_{\mathrm{user}}\right)$, $\mathrm{ID}_{\mathrm{user}}$ and $Pr \parallel S_{\mathrm{AP}}\left(Pr\right)$ to the application. The application needs to verify whether the user is in the possession of the private key corresponding to $\mathrm{ID}_{\mathrm{user}}$, for example, by setting up a SSL/TLS connection with client authentication. The application then makes sure the n_{AP} submitted by the user equals the n_{AP} contained in the location proof. Then it computes the following hash value:

$$H' = \text{hash}\left(S_{\text{user}}\left(n_{\text{AP}} \parallel n_{\text{user}}\right) \parallel n_{\text{user}}\right)$$

and makes sure H' equals the hash value H in the location proof. The application continues to verify $S_{\text{user}}\left(n_{\text{AP}} \parallel n_{\text{user}}\right)$ is a valid signature using the user's public key. Finally, the application checks the AP's signature on the location proof. As explained in section 3.6, our architecture is suitable for public APs, so home APs are not considered here. Only if all checks succeed, the proof will be accepted.

2.4.4 Security Analysis

By including a hash of the user's signature, instead of the actual signature, a location proof does not reveal the user's identity to the AP. Nevertheless, the proof may still be abused by a malicious application, which tries to find a public key / nonce combination that will verify the signature, $S_{\text{user}}\left(n_{\text{AP}} \parallel n_{\text{user}}\right)$, correctly. This attack is infeasible in our protocol, however, for the following reason. n_{AP} is not only covered by the signature, but also included in the proof. Similarly, n_{user} is covered by both the signature and H. In other words, $n_{\text{AP}} \parallel n_{\text{user}}$ is a message committed in the proof, so the security of the digital signature will prevent a forgery.

Replay attacks take place when someone records network traffic and replays it later on. A dishonest user may collude with a malicious intruder to launch replay attacks, in an attempt to acquire location proofs from a place where the dishonest user is not currently located. Suppose a dishonest user was at place P_1 at time T_1. By requesting a location proof from a nearby access point A_1, the user successfully obtained n_{AP}, a random nonce generated by A_1. She then can give n_{AP} and $S_{\text{user}}\left(n_{\text{AP}} \parallel n_{\text{user}}\right)$ to a malicious intruder in the same area. Later at time T_2 when this dishonest user was at another place P_2 which is far away from P_1, she wants to get a location proof from A_1 again with time stamp T_2. The malicious intruder then tries to request a location proof on behalf of the user. However, since the AP generates a new random nonce for each request, it is extremely unlikely for the new random nonce to be the same as the one in the malicious intruder's hand. Therefore, this attack is next to impossible in our architecture. The malicious intruder may keep trying until she receives a random nonce from the AP matching the random nonce in her hand so that she could use $S_{\text{user}}\left(n_{\text{AP}} \parallel n_{\text{user}}\right)$. But this requires generating a huge amount of requests, and such abnormal behavior will likely be detected and blocked by the AP. Therefore, our architecture resists collusion between dishonest users and malicious intruders.

In the above attack, it is also impossible for the malicious intruder to sign a fresh nonce n_{AP} on behalf of the dishonest user, because under the trust assumptions in section

2.3.1, she does not have the private key of the dishonest user. Note that the malicious intruder may also try to set up a communication channel through which she can send the new random nonce obtained from the AP to the remote dishonest user to get the signed random number in real time. But this is a wormhole attack, which we do not aim to solve in this paper.

Dishonest users may try to collude with defecting APs. An AP colluding with the user can issue location proofs with fake information. For example, a malicious AP in Los Angeles may be bribed by a user to include "New York" instead of "Los Angeles" as the location field in the proof. This kind of cheating may or may not be detected by applications depending on whether applications have knowledge of this malicious AP's geographical location. Since the proof is signed by the AP, if the application knows the exact location of this AP or takes time to actually investigate, the discrepancy between the actual location of the AP and the location claimed by the AP in the proof will be detected by the application. Otherwise, if the application blindly trusts the location information specified in the proof, cheating will go undetected. This drawback of the architecture will be rectified in the new architecture presented in section 3.4.

It is possible for several APs to collude and compromise a user's privacy by tracking her location. Since a user who contacts an AP must be nearby that AP due to the short-range transmission of Wi-Fi signals, if several APs learn the user's identity, they can jointly track the user's location. Besides, user devices may also be remotely fingerprinted [20] by the APs they are connected to without cooperation of these devices. With this technique, malicious APs can track, with a certain probability, a device as it connects to the Internet from APs. These threats are hazardous to users' location privacy, which our architecture can not defend against and are left as future work. But our architecture prevents malicious APs from learning users' identities. Therefore, as long as malicious APs do not learn the identity of the owners of the tracked devices, users' location privacy is not completely compromised.

2.4.5 Improvements Over Previous Work

Saroiu et al. [26] proposed a location proof mechanism that shares many commonalities with ours and is the most recent and relevant work. We compare their mechanism with our architecture in this section.

In Saroiu et al.'s mechanism, APs broadcast beacon frames periodically. A user who wants to request location proofs from an AP must first capture a frame and extract the sequence number from it, then she returns the signed sequence number along with her public key (which is used as the user's identity) to the AP. After checking the validity of the signature, the AP issues a location proof to the user. We identify two advantages that our architecture has over this mechanism, with the first one being a major improvement.

First, Saroiu et al. claimed in order to reduce the chance of being tracked, users are free to choose when to request location proofs and when to present them to an application. But requiring a user to keep an eye on her own privacy is not only unrealistic but also error-prone. This is a major drawback of Saroiu et al.'s approach. Most mobile users are simply regular people from a variety of backgrounds other than computer security. These users are unlikely to have even bare amount of knowledge or experience in privacy protection, not to mention making appropriate privacy-oriented decisions. Even people with computer science background and security technicians sometimes make incorrect judgments about what privacy-protection measures to take. In comparison, our architecture incorporates privacy protection as a built-in component. By exploiting cryptographic signatures and hashes, we disguise user identities from APs, thus making user identities untrackable. The burden of ensuring user privacy is entirely rested with the protocol, so users do not have to be wise about when to request location proofs and when to not.

Second, our architecture keeps workload distributed among applications and APs more evenly than Saroiu et al.'s mechanism. In their mechanism, a user's signature is verified at the AP from which the user is requesting a location proofs. However, in our architecture a user's signature is hashed to conceal the user's identity from the AP, thus it is not verified by the AP. Instead, applications are entities responsible for verifying signatures; moreover, only when the user presents her location proof to the application is her signature verified. There are two advantages of shifting this workload from the AP to the application. For one thing, APs are weaker devices compared to applications in terms of computing power. Verifying cryptographic signatures is an expensive operation, so performance of APs will decline drastically if this operation is executed very often. But application servers are typically more powerful devices and capable of handling a large volume of costly operations. For another, signature verification is sometimes unnecessary. For instance, if a user collects location proofs precautionarily as evidence for future police investigations that she may get involved in, those location proofs may not get used during the user's lifetime. Therefore, it will be misallocation of resources for APs to verify user signatures when issuing these unlikely-to-be-used location proofs. But when it is time for users to present location proofs to interact with applications, validity of the signatures does need to be checked without any doubt; in other words, verification at this stage is necessary and more appropriate. In our architecture, by shifting workload of user signature verification from APs to applications, we ensure this expensive operation is carried out by appropriate parties with enough computing power and is executed only when it is necessary.

In the example of police investigation mentioned above, it may appear desirable for users to always request location proofs proactively, so that when they are involved in legal issues related to proving or disproving their location, they will be able to do so by revealing their proofs. But location proofs are obtained under the assumption that users are carrying their devices when proofs are requested (see section 2.3.1). Although this assumption is very likely to hold in practice, people that intend to commit crime are likely to break this assumption deliberately by leaving their devices at a place different from the crime scene and collecting location proofs which could help prove their innocence. Location proofs provide additional security and evidence when it comes to proving or disproving people's location, but they are unsuitable to serve as hard evidence for people's locations. Therefore, having people always request proofs proactively does not help here. In fact, this could even damage user privacy in some circumstances such as users being forced to reveal their proactively requested proofs.

2.4.6 Scenarios Where the Lightweight Architecture Applies

The architecture introduced in this section is a lightweight architecture in that the protocol requires relatively few round-trip communications and only three parties are involved: user, AP, application. In contrast, the complete architecture that will be presented in section 3.4 not only possesses all the functionalities of this lightweight architecture, but is also more feature-rich at the cost of more computational overhead and more complicated working model with more parties involved. Although the complete architecture proves more powerful than this simple one, the lightweight architecture still has its value and is a better option in some scenarios. For example, Saroiu et al. provide an interesting application of location proof: green commuting. Microsoft gives rewards to its employees who use other ways of commuting, such as taking buses, than driving their own cars. This green commuting program requests a location proof architecture to allow employees to collect location proofs issued by APs near bus lines. However, the reward is not significant enough to give employees initiatives to bother the inconvenience of cheating. This is an ideal scenario where the lightweight architecture outshines the more complicated one due to its low computational overhead and cost of setup. However, in some other scenarios such as presenting location proofs in police investigation, it is critical to detect cheating and protect user privacy, which requires the more heavyweight architecture in spite of its overhead and cost. Therefore, we next introduce a more powerful architecture that addresses more stringent location proof requirements.

Chapter 3

Complete Architecture

3.1 Introduction

In this chapter, we present the last three of our seven design goals and present a complete location proof architecture that meets all the design goals. The chapter is organized as follows. We state the last three of our seven design goals in section 3.2, which is followed by the trust assumption in section 3.3 and the presentation of a complete architecture in section 3.4 that incorporates these three goals. In section 3.5, we provide several observations of the complete architecture. Finally, we discuss deployment strategies in section 3.6.

3.2 Additional Design Goals

The architecture introduced in section 2.4 has three missing features that could be beneficial in practice. We formalize these three features as additional design goals below.

Before diving into the details, we must point out the necessity of splitting our seven design goals (with the first four in section 2.2 and the last three in this section). As argued in section 2.4.6, a full-blown architecture with a complete set of features is not necessarily suitable for all scenarios and can actually prove cumbersome. Therefore, we identify the four design goals in section 2.2 as the most crucial requirements for both lightweight and heavyweight architectures, and present a lightweight architecture that realizes those design goals. In this section, we will present three additional design goals that will be realized by a more heavyweight and powerful location proof architecture.

Granularity level	1	2	3	4	5	* (wildcard)
Location information	country	province	city	street	building	all

Table 3.1: Location granularity

3.2.1 Location Granularity

Although a location proof generated by the protocol in section 2.4.2 contains location information, the application can learn more location information by investigating the location of the issuing AP. For example, if a certain service is offered to residents in Canada, users presenting a location proof issued by an AP in Montreal not only proves their qualification, but also reveals overly detailed location information, thus allowing for applications to track their location.

As the first step of providing location granularity control, we define 6 location granularity levels as shown in table 3.1. Next, we make changes to the architecture and the protocol to allow users to specify desired location granularities in the proof. Last but not least, we hide the issuing APs from the application, because otherwise the application can find out the detailed location of the issuing AP, which makes location granularity control evaporate.

3.2.2 Proactive Location Proof Requests

In the architecture introduced in section 2.4, users are aware of which APs the application trusts as well as the desirable location granularity before requesting a proof. But more often than not, users may just want to collect location proofs for future purposes. For example, if a user would like to be able to provide evidence for her presence at some location in possible police investigation, she may tend to collect location proofs whenever she is nearby APs. Another example is that users foresee upcoming LBSes and applications may require proofs of their location, thus, they will have incentives to collect location proofs before hand.

We define two terms to be used in later discussion.

- **Retroactive location proof** is a location proof that is requested by a user to interact with a target application.
- **Proactive location proof** is a location proof that is collected by a user for future purposes, without having a target application in mind at the time of request.

To allow for proactive location proofs, we must make sure location proofs do not contain application-specific data; in other words, they should be crafted to follow a general format.

3.2.3 Cheating Detection

The architecture in section 2.4 allows applications to verify a location proof is legitimate in the sense that the application can check whether the proof has been tampered with by the user, and whether it is issued by a valid AP. However, if a dishonest user manages to obtain location proofs from multiple far-apart APs simultaneously by colluding with multiple parties, cheating can not be detected. For example, a dishonest user may launch wormhole attacks by colluding with two malicious intruders in Los Angeles and New York respectively, to obtain two location proofs that certify this user's presence in both places at the same time. Later on, the user can present the proof obtained in Los Angeles to application A, and present the other proof to another application B to prove her location in both places. Since application A and B are probably run by different service providers, they have no share knowledge of the dishonest user's location and thus will both believe the user's proof is legitimate. Note that this is a special type of wormhole attack in that it involves a dishonest user launching the attack with multiple remote parties at different places. We will elaborate on how the complete architecture can detect this attack in section 3.4.3, but we do not aim to solve general form of wormhole attacks.

Users are not the only entities that may lie. As stated in section 2.3.2, a malicious or misconfigured AP may also generate location proofs with fake information. For example, the operator of an AP in Los Angeles may be bribed by a local user to configure the AP in such a way that it generates location proofs representing places in New York.

These examples suggest we need a mechanism to detect cheating.

3.3 Trust Assumptions and Threat Model

3.3.1 Working Model and Roles of Trusted Third Parties

The basic working model of the revised architecture is that we let an AP issue an *informal* location proof that certifies the user's presence near the AP. Later when the user wants to provide the application with a *formal* proof, she must first presents her informal proof to one of the TTPLs (introduced next) that the application trusts to obtain a formal location proof.

In order to achieve these goals, we need three types of trusted entities and they need to be run by different parties.

- TTPLs (Trusted Third Party for managing Location information) are responsible for issuing formal location proofs by creating a new proof that includes all the information contained in informal location proofs, with some information transformed to another representation. In particular, A TTPL should replace the AP identity (i.e. $\mathrm{ID}_{\mathrm{AP}}$) in an informal proof with the actual location information and an encrypted $\mathrm{ID}_{\mathrm{AP}}$ when copying it to a formal proof. Furthermore, A TTPL should make its mapping from AP identities to geographical locations publicly available. A TTPL can create the mapping from APs to geographical locations in two ways: (1) a TTPL can use war-driving to collect AP information, or (2) a TTPL can receive reports of APs' location from companies that operate the APs and verify the correctness of that information. The disclosure of AP-to-location mapping serves two purposes: (1) Applications can verify the accuracy of these mappings and decide which TTPLs to trust. (2) The CDA (introduced later in this section) can directly use TTPLs' mappings in its cheating detection procedure, thus dispensing with the overhead of maintaining an AP-to-location database itself. Several TTPLs exist in our architecture, each of which is trusted by different applications. For example, TTPLs in charge of issuing proofs for all Starbucks APs are trusted by Starbucks stores but probably not by Tim Hortons.

- TTPU (Trusted Third Party for managing User information) is in charge of storing encrypted location information associated with users. More specifically, the TTPU stores triples of the form $(\mathrm{ID}_{\mathrm{user}},\ T,\ E)$, where T represents the time when the user (with identity $\mathrm{ID}_{\mathrm{user}}$) requested a proof and E represents encrypted identity of the AP that issued the proof to the user. In a word, each record shows when and where (to be more precise, encrypted "where") a particular user requested a location proof. When verifying the formal location proof submitted by the user, the application submits $\mathrm{ID}_{\mathrm{user}}$ and T to the TTPU, which searches in its database for any record matching the same $\mathrm{ID}_{\mathrm{user}}$ and roughly the same T value. The TTPU extracts corresponding E values and submits them to the CDA (introduced next) for cheating detection. Note that location information is encrypted with the CDA's public key.

- CDA (Cheating Detection Authority) is the entity that carries out the actual cheating detection action. After receiving encrypted location information from the TTPU, the CDA decrypts them and checks whether any two locations are far apart, which is a sign of cheating (because the same user can not request location proofs at two far-apart places simultaneously). The CDA notifies the TTPU in case cheating is detected. As mentioned above, the CDA does not manage its own AP-to-location database; instead, it makes use of the mapping published by the TTPL that issued the formal location proof.

3.3.2 Revised Trust Assumptions and Threat Model

Like the lightweight architecture, our complete architecture targets those applications that are not written by mobile service providers and not authorized by mobile service providers to access users' location information.

Neither users nor APs are trusted by applications, because users have incentives to lie about their location and APs may defect to collude with users. We list trust assumptions on three kinds of trusted third parties (TTPs) as follows.

- **TTPLs** are trusted by applications to replace AP identity in informal location proofs with correct location information when producing formal proofs. In addition, TTPLs are trusted to encrypt the correct AP identity from an informal location proof and include the resulting ciphertext in the formal proof. A particular TTPL may be trusted by some applications and distrusted by the others. Therefore, when a user wants to convert her informal location proofs into formal proofs, she should consult the application first to get a list of TTPLs trusted by that application. TTPLs are trusted by users not to collude with applications. Otherwise TTPLs and applications can collectively track users' location. Last but not least, TTPLs are trusted not to collude with users or APs to provide fake location information in formal location proofs, because otherwise the cheating detection mechanism in our architecture will fail.

- **TTPU** is trusted by applications to assist in cheating detection. More specifically, when an application submits a user's ID and time T to the TTPU, the TTPU should retrieve from its database all the records of the target user with time value close to T, and submit corresponding E value to the CDA for cheating detection. Furthermore, the TTPU is trusted by applications to forward cheating detection results from the CDA to applications without altering the results. The TTPU is also trusted by users to not disclose their identities to the CDA, because otherwise users may be tracked by the CDA. If the TTPU defects, it may collude with users to make their cheating undetected by returning fake cheating detection results to applications. For example, when receiving $\mathrm{ID}_{\mathrm{user}}$ from an application, the TTPU only extracts a subset of all the records of the user such that the CDA does not have sufficient information to detect cheating. Even simpler, the TTPU can skip the step of contacting the CDA and simply notify the application that no cheating behavior was detected. Therefore, if the TTPU violates these trust assumptions, cheating detection mechanism in our architecture will fail.

- **CDA** is trusted by applications to conduct cheating detection and report results honestly. The CDA is also trusted by users to keep their location information concealed from the TTPU. If the CDA becomes malicious, it may be bribed by users to not report cheating. Moreover, a malicious CDA may also collude with the TTPU to track users' location, since the TTPU knows users' identities and associated E which can be decrypted by the CDA.

Finally, we assume all the TTPs are run by different organizations. This is an approach that has also been taken by other researchers [7, 17]. However, we did study approaches without such TTPs in section 3.5.3, but they are not as effective as our TTP-based approach (see section 3.5.3 for explanation). Overall, TTPs play essential roles in the complete architecture. How much the TTP-based infrastructure can be addressed by using cryptography is discussed in section 3.5.3.

Finally, we assume communication is secured against passive and active eavesdroppers with the help of TLS/SSL.

3.4 Architecture for Obtaining and Verifying Location Proofs

In this section, we design a more heavyweight architecture to fulfill the additional design goals discussed in section 3.2.

3.4.1 Obtaining an Informal Location Proof

Before the user can request a formal location proof from a TTPL, she must request an informal proof from an AP by executing the following protocol. As usual, we use the user's and APs' public keys as their identities, i.e. $\mathrm{ID}_{\text{user}} = P_{\text{user}}$ and $\mathrm{ID}_{\mathrm{AP}} = P_{\mathrm{AP}}$, where P_{user} and P_{AP} stand for public keys of the user and the AP respectively.

1. The user sends $\mathrm{ID}_{\text{user}}$ and $e_1 \parallel S_{\text{user}}(e_1)$ to the TTPU, where $e_1 = \mathrm{Enc}(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}})$ is the encrypted $\mathrm{ID}_{\mathrm{AP}}$ with the CDA's public key and $\mathrm{ID}_{\mathrm{AP}}$ is the identity of the AP from which the user attempts to request an informal location proof.

2. The TTPU sends the user a random nonce n_{TTPU}.

3. The user sends
$$n_{\text{user}} \parallel S_{\text{user}}(n_{\mathrm{TTPU}} \parallel n_{\text{user}})$$
to the TTPU, where n_{user} is a random nonce generated by the user.

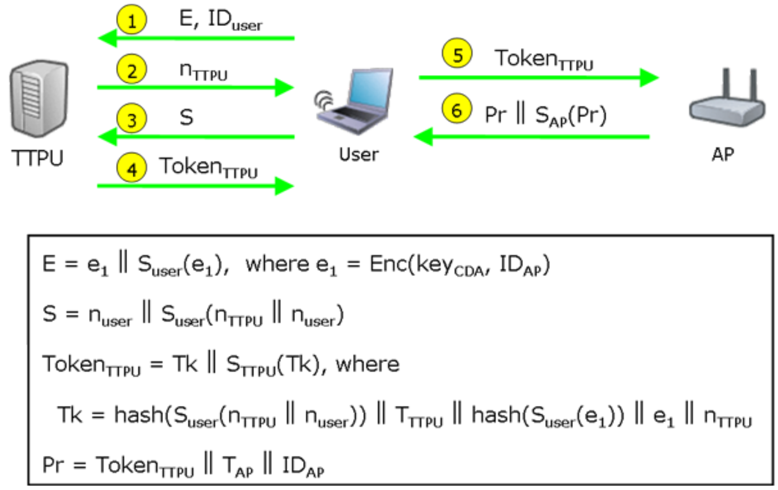

1 E, ID_user
2 n_TTPU
3 S
4 Token_TTPU
5 Token_TTPU
6 Pr ǁ S_AP(Pr)
TTPU
User
AP
E = e1 ǁ S_user(e1), where e1 = Enc(key_CDA, ID_AP)
S = n_user ǁ S_user(n_TTPU ǁ n_user)
Token_TTPU = Tk ǁ S_TTPU(Tk), where
Tk = hash(S_user(n_TTPU ǁ n_user)) ǁ T_TTPU ǁ hash(S_user(e1)) ǁ e1 ǁ n_TTPU
Pr = Token_TTPU ǁ T_AP ǁ ID_AP

Figure 3.1: Request an informal location proof

4. The TTPU verifies the user's signature $S_{\text{user}}\left(n_{\text{TTPU}} \parallel n_{\text{user}}\right)$ (the scalability issue of the TTPU is discussed in section 3.5.1). The TTPU then stores $(\text{ID}_{\text{user}},\ T_{\text{TTPU}},\ e_1)$ in its database, where T_{TTPU} is the time when the TTPU receives the request from the user. A new record of the user is added to the TTPU's database every time the user registers with the TTPU before requesting a location proof. Finally, the TTPU sends the following token to the user.

$$\text{Token}_{\text{TTPU}} = Tk \parallel S_{\text{TTPU}}\left(Tk\right)$$

   where

$$Tk = \text{hash}\left(S_{\text{user}}\left(n_{\text{TTPU}} \parallel n_{\text{user}}\right)\right) \parallel T_{\text{TTPU}} \parallel \text{hash}\left(S_{\text{user}}\left(e_1\right)\right) \parallel e_1 \parallel n_{\text{TTPU}}$$

5. The user sends $\text{Token}_{\text{TTPU}}$ to the AP.

6. The AP sends to the user an informal location proof of the following format.

$$Pr \parallel S_{\text{AP}}\left(Pr\right)$$

   where

$$Pr = \text{Token}_{\text{TTPU}} \parallel T_{\text{AP}} \parallel \text{ID}_{\text{AP}}$$

   and T_{AP} is the time when the AP receives the request.

This procedure is demonstrated in figure 3.1.

3.4.2 Obtaining a Formal Location Proof

The user executes the following protocol to obtain a formal location proof.

1. The user sends the informal location proof obtained from an AP and the desired location granularity g to the TTPL, where $g = 1, \ldots, 5$.

2. The TTPL examines whether the proof contains a TTPU token. If it does not, the request is rejected. Otherwise, the TTPL replaces $\mathrm{ID}_{\mathrm{AP}}$ field with L_g which is actual location information of granularity level g. Moreover, the TTPL includes encrypted $\mathrm{ID}_{\mathrm{AP}}$ with the CDA's public key in the proof for applications to check whether the user has cheated when registering with the TTPU. Finally, the TTPL sends to the user a formal location proof of the following format.

$$Pr' \parallel S_{\mathrm{TTPL}}\left(Pr'\right)$$

   where

$$Pr' = \mathrm{Token}_{\mathrm{TTPU}} \parallel T_{\mathrm{AP}} \parallel e_2 \parallel L_g$$

   and $e_2 = \mathrm{Enc}\left(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}}\right)$

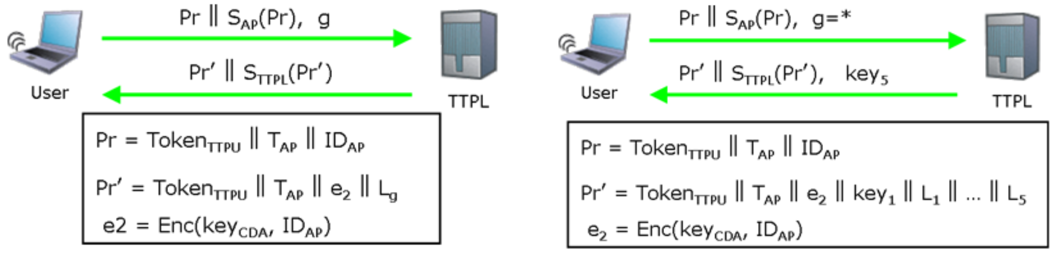
This procedure is demonstrated in figure 3.2a. Note that although $\mathrm{Enc}\left(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}}\right)$ is already included in $\mathrm{Token}_{\mathrm{TTPU}}$, APs may cheat. Therefore, the TTPL must include it again in the proof for the CDA to check for cheating later.

The above protocol applies to the case where the user has a desired location granularity in mind. But as we discussed in section 3.2.2, the user is unlikely to know what granularity to ask for when she requests in a proactive manner. The strategy to deal with this case is to include in the proof encrypted location information of every granularity level and allow the user to selectively reveal location information of the desired granularity. The modified protocol is as follows.

1. The user sends the informal proof obtained from an AP and the wildcard location granularity $g = *$ to the TTPL.

2. The TTPL generates a root key key_5, which is used to encrypt the most granular location information, that is, $\mathrm{L}_5 = \mathrm{Enc}\left(\mathrm{key}_5\ ,\ \text{location of granularity level 5}\right)$. For granularity level $g = 4, \ldots, 1$, the TTPL computes

$$\begin{aligned} \mathrm{key}_g &= \mathrm{hash}\left(\mathrm{key}_{g+1}\right) \\ \mathrm{L}_g &= \mathrm{Enc}\left(\mathrm{key}_g\ ,\ \text{location of granularity level } g\right) \end{aligned}$$


Pr || S_AP(Pr), g
Pr′ || S_TTPL(Pr′)
User
TTPL
Pr = Token_TTPU || T_AP || ID_AP
Pr′ = Token_TTPU || T_AP || e_2 || L_g
e2 = Enc(key_CDA, ID_AP)
Pr || S_AP(Pr), g=*
Pr′ || S_TTPL(Pr′), key_5
User
TTPL
Pr = Token_TTPU || T_AP || ID_AP
Pr′ = Token_TTPU || T_AP || e_2 || key_1 || L_1 || ... || L_5
e_2 = Enc(key_CDA, ID_AP)

(a) Request a formal location proof retroactively

(b) Request a formal location proof proactively

Figure 3.2:

The TTPL sends to the user the root key key$_5$ and a formal location proof of the following format.

$$Pr' \parallel S_{\text{TTPL}}\left(Pr'\right)$$

where

$$Pr' = \text{Token}_{\text{TTPU}} \parallel T_{\text{AP}} \parallel e_2 \parallel \text{key}_1 \parallel \text{L}_1 \parallel \ldots \parallel \text{L}_5$$

and $e_2 = \text{Enc}\left(\text{key}_{\text{CDA}},\ \text{ID}_{\text{AP}}\right)$

This procedure is shown in figure 3.2b.

Direct communication with the TTPL reveals the user's identity such as her MAC address, so communication should be anonymous. Users can use tools such as Tor to achieve anonymity, which is discussed in section 3.5.5.

3.4.3 Verifying a Formal Location Proof

The user submits n_{TTPU}, n_{user}, $S_{\text{user}}\left(n_{\text{TTPU}} \parallel n_{\text{user}}\right)$, $S_{\text{user}}\left(e_1\right)$, ID$_{\text{user}}$ and $Pr' \parallel S_{\text{TTPL}}\left(Pr'\right)$ to the application. In order to verify the location proof, the application carries out the following procedure:

1. The application first verifies $S_{\text{user}}\left(e_1\right)$ submitted by the user is a valid signature and its hash value equals the second hash value contained in the TTPU token. Furthermore the application verifies $S_{\text{user}}\left(n_{\text{TTPU}} \parallel n_{\text{user}}\right)$ submitted by the user is a valid signature and the n_{TTPU} submitted by the user matches the n_{TTPU} in the TTPU token. The application then computes the hash value of this signature, i.e. hash$\left(S_{\text{user}}\left(n_{\text{TTPU}} \parallel n_{\text{user}}\right)\right)$, to make sure it equals the first hash value in the TTPU token.

2. If the above step is passed, the application verifies whether T_{AP} (extracted from the location proof) and T_{TTPU} (extracted from the TTPU token) represent approximately the same time. The meaning of “approximately” varies depending on the specific application which is discussed in section 4.2.7.

3. If the above step is passed, the application sends $\mathrm{ID}_{\mathrm{user}}$, T_{TTPU}, e_1 and e_2 to the TTPU for cheating detection. Moreover, the application notifies the TTPU which TTPL issued the formal location proof.

4. With T_{TTPU} and $\mathrm{ID}_{\mathrm{user}}$, the TTPU is able to pull out all the records of the user with approximate T_{TTPU} value from its database. The TTPU then extracts from all these records the encrypted identities of APs which issued the user location proofs around time T_{TTPU}, and send the ciphertexts to the CDA. Moreover, the TTPU also sends e_1 and e_2 to the CDA and informs the CDA which TTPL issued the formal location proof.

5. The CDA decrypts all the ciphertexts to reveal AP identities. If any decrypted ciphertext does not reveal a valid AP identity, the CDA aborts the checking procedure and reports to the TTPU that the user has cheated. If all the ciphertexts are decrypted to valid AP identities, the CDA compares whether $\mathrm{ID}_{\mathrm{AP}_1} = \mathrm{ID}_{\mathrm{AP}_2}$, where $\mathrm{ID}_{\mathrm{AP}_1}$ is decrypted from e_1 and $\mathrm{ID}_{\mathrm{AP}_2}$ is decrypted from e_2. If these two decrypted AP identities are not identical, the CDA aborts the checking procedure and informs the TTPU that the user has cheated. Otherwise, the CDA proceeds with the rest of the procedure by examining whether any two $\mathrm{ID}_{\mathrm{AP}}$'s are far apart, which is an indicator of cheating, since it is impossible for the same user to simultaneously obtain multiple location proofs from different places at roughly the same time unless the user cheats by launching wormhole attacks. Note that when executing this step, the CDA should obtain and use the AP-to-location mapping published by the TTPL that issued the formal proof. Finally, the CDA reports to the TTPU whether the user has cheated, which is forwarded to the application by the TTPU.

If all the steps above are passed successfully and no cheating is detected, the application accepts the proof. If the proof is a “retroactive” location proof, the application can use the L_g part in the proof directly. However, in the case the user requested the proof proactively, the user needs to decide a desired granularity g and to send the application the appropriate decryption key key_g for decrypting L_g. For instance, if the user intends to reveal location information of granularity level 3, she should first compute $\mathrm{key}_4 = \mathrm{hash}\,(\mathrm{key}_5)$ and $\mathrm{key}_3 = \mathrm{hash}\,(\mathrm{key}_4)$, and then send key_3 to the application. On receiving the decryption key, the application must make sure the user submitted a proper key by doing the following computation: for $i = g, \ldots, 1$, $\mathrm{key}_i = \mathrm{hash}\left(\mathrm{key}_{i+1}\right)$. If the computed key_1 matches the key_1 in the location proof, the application can be certain the decryption key submitted by the user is valid.

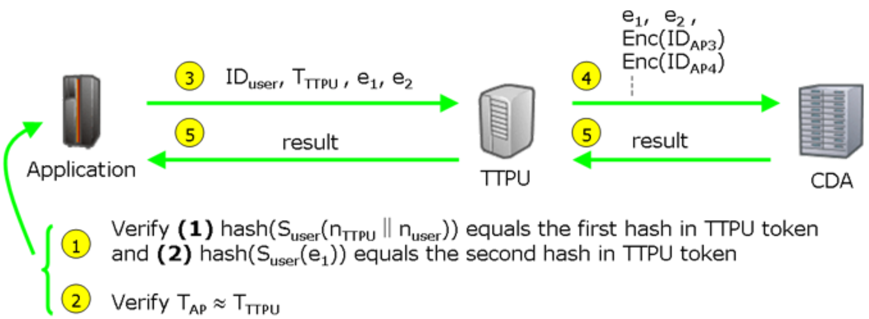

3
$\mathrm{ID}_{\mathrm{user}}$, T_{TTPU}, e_1, e_2
5
result
Application
TTPU
4
e_1, e_2,
$\mathrm{Enc}(\mathrm{ID}_{\mathrm{AP3}})$
$\mathrm{Enc}(\mathrm{ID}_{\mathrm{AP4}})$
5
result
CDA
1
Verify **(1)** hash(S_{user}($n_{\mathrm{TTPU}} \| n_{\mathrm{user}}$)) equals the first hash in TTPU token
and **(2)** hash(S_{user}(e_1)) equals the second hash in TTPU token
2
Verify $T_{\mathrm{AP}} \approx T_{\mathrm{TTPU}}$

Figure 3.3: Verification of location proofs

This procedure is sketched in figure 3.3.

3.4.4 Security Analysis

The architecture presented above defeats collusion between users and APs. Assuming information in the TTPU token is correct, if users and APs collude to produce informal location proofs with fake information, it will be detected during the verification phase, because the information that an AP places in an informal proof can be verified by comparing corresponding fields in the TTPU token. For example, an incorrect T_{AP} or $\mathrm{ID}_{\mathrm{AP}}$ will be detected in step 2 and 5 of the verification phase respectively.

However, the TTPU token may contain incorrect information, since the user may cheat by submitting a wrong $\mathrm{ID}_{\mathrm{AP}}$ when registering with the TTPU. For example, a dishonest user may submit $S_{\mathrm{user}}\left(\mathrm{Enc}\left(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}_1}\right)\right)$ to the TTPU to obtain a TTPU token, while she is actually attempting to request a location proof from another AP with identity $\mathrm{ID}_{\mathrm{AP}_2}$. No matter whether she further colludes with AP_2, her cheating will not succeed. Assume AP_2 behaves honestly and places its real identity $\mathrm{ID}_{\mathrm{AP}_2}$ in the informal proof. In this case, the dishonest user has no way of preventing the TTPL from copying the correct $\mathrm{ID}_{\mathrm{AP}_2}$ committed by AP_2 to the formal location proof, which allows the application to detect the cheating in step 5 of the verification phase. If AP_2 does collude with the user and puts the incorrect identity $\mathrm{ID}_{\mathrm{AP}_1}$ in the informal proof, it is not able to sign the proof with AP_1's private key. Therefore, the user's attempt to cheat by submitting incorrect information to the TTPU fails regardless of whether the AP colludes with her.

A TTPU token is valuable to the user, since a valid location proof must contain a valid TTPU token. Therefore, attackers have incentives to steal legitimate users' TTPU tokens by eavesdropping on their communication with the TTPU. Once the attacker succeeds in obtaining a TTPU token, she will use it to request informal location proofs from APs without having to register with the TTPU first. Moreover, the attacker may also try to sell it to other users who also intend to cheat by using other people's TTPU tokens. But since each TTPU token contains a signature of its intended recipient, nobody can exploit other people's TTPU tokens.

Replay attacks take place when a user reuses the same TTPU token she previously obtained for requesting more than one location proof. However, this type of attack is also defeated in our architecture. A TTPU token produced by the TTPU contains 5 fields as illustrated in section 3.4.1. The second field (the time when the user is about to send a location proof request to the AP) and the fourth field (the AP from which the user attempts to request a proof) specify when and where the user tries to request an informal location proof. The AP that issues an informal location proof to the user has to provide the "when" and "where" information once again in the informal proof, and include the TTPU token as the first field of the proof, which is required by TTPLs. Therefore, if the user reuses the same TTPU token with the same AP, the AP has no choice but to enter correct "when" and "where" information in order to produce a valid proof. On the other hand, if the AP colludes with the user and provides incorrect information in the proof, inconsistency will be resulted from comparing the incorrect information with the valid information in the TTPU token, thus the verification procedure will fail and the proof will be rejected. If the user reuses the same TTPU token with a different AP, that AP has no way of proving itself to be the AP indicated in the TTPU token, since it does not possess the private key of the AP in the token and thus unable to produce informal proofs with correct signature. Therefore, even if the user manages to collude with another AP to reuse previously obtained TTPU token, informal location proofs produced by the colluding AP can never be used to obtain a formal location proof from TTPLs because they lack valid signature.

Another form of replay attack also exists. An attacker eavesdrops on communication between the user and the TTPU to intercept data E in the first step of TTPU registration (see section 3.4.1). Then the attacker stands down. Note that E contains the encrypted identity of the AP from which the user plans to request a location proof. Later when the same user tries to register with the TTPU again but in order to request location proofs from different APs, the attacker eavesdrops on the user's communication again and replaces the E in the first step with the E intercepted early on. This way, the user unintentionally commits to a wrong AP and will be accused of cheating while totally unaware of being framed by the attacker. This attack is defeated by communicating over TLS/SSL. As stated in section 3.4.1, communication between the user and the TTPU should always be protected by TLS/SSL.

Applications and the TTPU should be prevented from knowing $\mathrm{ID}_{\mathrm{AP}}$, i.e. the identity of the AP that issued location proofs to the user, since the knowledge of $\mathrm{ID}_{\mathrm{AP}}$ will allow them to deduce the user's accurate location by investigating the AP's geographical location. Instead, both the application and the TTPU are given the encrypted $\mathrm{ID}_{\mathrm{AP}}$ when the user presents her formal location proof. Applications and the TTPU may use brute-force attack to try to break the protection of encryption: they encrypt identities of all the APs in the world using the CDA's public key and compare ciphertexts to the ciphertext of encrypted $\mathrm{ID}_{\mathrm{AP}}$ in the user's proof. If two ciphertexts are identical, they will learn which AP issued the proof. However, we use probabilistic encryption as specified in section 2.4.1, which produces different ciphertexts each time the same message gets encrypted. Therefore, using brute-force attack to break the security of our architecture is not possible.

Even though we state in section 3.3.2 that a TTPL is trusted to encrypt the correct $\mathrm{ID}_{\mathrm{AP}}$ from an informal proof and include the resulting ciphertext in the formal proof, this assumption is not required for the application to detect and reject invalid location proofs. If a TTPL encrypts an incorrect $\mathrm{ID}_{\mathrm{AP}}$ in the formal proof, the CDA will capture this anomaly during cheating detection phase, so the proof will not be accepted by the application. However, the user will be wrongly accused of cheating whereas the TTPL is the real culprit to blame in this case. Therefore, this trust assumption is still required in our architecture to ensure honest users are not framed by TTPLs.

3.5 Observations On the Complete Architecture

3.5.1 Scalable TTPU

The TTPU is shown as a single server in figure 3.1, but this is unrealistic in real-world deployment due to the high performance demand of the TTPU. Because the TTPU must act as a global entity that stores user information for the purpose of cheating detection, there cannot be multiple independent TTPUs as in the case of TTPLs, since independent TTPUs will each have a subset of global user information, which is insufficient for cheating detection. A high performance data center is a suitable choice for the TTPU. Moreover, servers of the data center should use a centralized database to store user information for the purpose of cheating detection. Since whenever a user tries to request an informal location proof from an AP the TTPU must be contacted first to get a TTPU token, this raises substantial scalability requirement of the TTPU.

A memcached system [22], which is widely used by many websites such as Wikipedia and Facebook that serve an enormous amount of users, is a viable option for making the TTPU data center scalable. A memcached system is a distributed and high performance memory caching system that reduces the number of database access by caching data on several machines. Memcached servers cache data locally. When a program on a memcached client inserts a chunk of data to the database or updates the database, it also caches the data on a memcached server. Later on, when the program on the memcached client needs to read the data, before it queries the database, it queries all the memcached servers and uses data from the cache if available. Since the memcached client sends the query to all the memcached servers in parallel, and the communication protocol used by memcached system is very efficient, performance can be boosted dramatically. To benefit from memcached, the TTPU data center needs to spare several machines to run as memcached servers. The rest of the data center are run as memcached clients and rely on memcached servers to cache user data during TTPU registration, so that the need of database access to retrieve user data to complete the second step of TTPU registration could be eliminated. If the data center fails to scale up as the number of user requests increases above a certain threshold, we can split the data center into several data centers, each of which handles a particular range of users. For example, based on the value of the first byte of users' IDs, user requests can be routed to different data centers.

3.5.2 How to Choose Between Retroactive and Proactive Location Proofs

A user can request formal location proofs from TTPLs in either a retroactive or a proactive manner as discussed in section 3.2.2. There are good reasons to use either approach, so which option to choose depends on users' preferences and various considerations.

There are two advantages of using proactive approach. First, since location proofs obtained using proactive approach can also be used in the retroactive case, a proactive proof is more general. Second, even if the user knows the desired location granularity for the time being, she might have to change it when encountering new applications in future. For example, suppose Alice obtains a location proof of granularity level 3 and uses it to interact with application A. Some time later, she needs to interact with another application B which requires location proofs for the same location but with granularity level 4. In this case, Alice has to request a new proof from TTPLs again. It might have been wiser for her to obtain a proactive proof in the first place. In addition, TTPLs may go down temporarily for maintenance. With a proactive proof, the user does not need to constantly go back to TTPLs whenever she wants to change the location granularity, thus suffering less from unavailability of TTPLs. According to our measurements, a proactive proof is about 22% larger in size than a corresponding retroactive proof (but both are smaller than 4 KB), and it takes a TTPL roughly 127% more time to produce a proactive proof (but both cost less than 0.3 millisecond). Therefore, proactive proofs do not have a significant disadvantage in contrast to retroactive proofs in terms of storage and processing time required, especially when storage is getting cheaper and mobile devices are getting faster. How often a user should request proactive proofs depends on the user's behavior pattern. The more frequent the user travels, the more frequent she should request proactive proofs.

There are valid justifications of using a retroactive approach as well. For example, a formal location proof obtained in a retroactive manner is smaller in size than a proactive proof which contains location information in every granularity level. Thus, for devices with small space, retroactive approach is more storage-friendly. Users can typically carry informal proofs most of the time and convert them to formal proofs of appropriate granularities right before they need formal proofs to interact with applications.

3.5.3 Two Alternatives to Trusted Third Parties

One of the major motivations of introducing trusted third parties to our location proof architecture is to hide AP identities from applications. This step is essential for protecting users' location privacy and supporting proactive location proofs. However, during the development of the architecture, we were inspired by another two approaches that could potentially allow us to achieve the same goal. We elaborate on these two alternatives below and explain why they are abandoned in favor of our current approach.

Group Signature Schemes

A group signature scheme [4] is a cryptographic mechanism for allowing members of a group to anonymously sign a message on behalf of the group. The central element in a group is the *group manager* who is in charge of group management such as adding a new group member and revoking a group member's ability to sign. Anyone can verify the correctness of the group signature but is unable to learn which member in the group produced it. By leveraging a group signature scheme, a number of APs can form a group to issue location proofs without revealing their identities, and the only thing learnt by the proof verifier is the group identity instead of the identity of individual APs. Despite the elegance of this approach and lower overhead (since trusted third parties can now be eliminated), group signature schemes are not appropriate for three reasons. Firstly, there are no sensible means to elect a group manager in the setting of an AP environment. APs in a particular region are run by different organizations, and those APs that belong to a certain organization function independently of those operated by another organization. Hence, there lack plausible motivation and mechanism for all these organizations to elect a group manager trusted by all other organizations. Secondly, it is difficult to set up a group such that location privacy and manageability are achieved at the same time. Large companies such as AT&T may choose to deploy group signature schemes on their APs. However, this kind of setup leaks user location information. For example, if an organization deploys APs only in a specific city or province, a location proof issued by any AP belonging to this organization reveals that the user was/is in that particular city or province, regardless of group signature schemes being used. Some small organizations may form an alliance and deploy group signature schemes to allow users to achieve better location privacy; however, the management of such a group will become troublesome if the size of the group is too large, whereas trying to keep the group size small does not provide satisfactory location privacy. Thirdly, an AP can be a member of only one group in a group signature scheme. If it were a member of multiple groups, it would have to sign a proof multiple times, once for each group, which is expensive. But if the AP belongs to only one group, it could result in problematic situation where applications do not trust the group manager but might trust some of the APs that are in the group. In this case, it is hard for the application to decide whether to continue its trust in those APs. Therefore, this approach has several problems in practice and not adopted in our architecture. Our architecture does not suffer from these problems, because in our architecture APs do not need to be managed by a central group manager; instead, companies and organizations set up and manage their own APs.

Delegatable Credential Schemes

Belenkiy et al.[2] proposed a delegatable credential system where users can obtain credentials from an authority and in turn delegate credentials to others. With such a system, we can build a hierarchical architecture for delegating location proofs. More specifically, a trusted root delegator is first established to issue location proofs representing the most coarse-grained location. Lower delegators obtain proofs from higher delegators and delegate new proofs, which are concatenation of proofs from higher delegators and more granular information at the current level, to other delegators or end users. An end user with a location proof can choose to reveal a prefix of the delegation path as her location proof. For instance, the root delegator for Canada delegates to provincial-level delegators location proofs labeled "Canada". Provincial-level delegators extend the proof with the location of the province name. For instance, delegators of Ontario create location proofs labeled "Canada, Ontario" based on the proof issued by the delegator for Canada. Similarly, delegators of Waterloo extends location proofs issued by delegators of Ontario with location "Waterloo", and again these proofs can be further extended by the delegator of the University of Waterloo with location "University of Waterloo". More granular location proofs can be produced in this fashion. An end user in the possession of such a location proof can choose to reveal a prefix, like "Canada", "Canada, Ontario", or "Canada, Ontario, Waterloo", etc, of the delegation path to an application that requires location proofs.

Two key features of Belenkiy et al.'s credential delegation system are non-malleablility of the delegation path and anonymity of delegators, which are crucial to the location proof architecture. The property of non-malleablility states that a particular delegator is unable to change the delegation path accumulated so far. Delegators may cheat, but they may cheat only in the sense that they are able to extend higher delegators' credentials with an incorrect credential, instead of replacing the delegation path with a fake path entirely. For example, malicious delegators of Waterloo may extend the proof "Canada, Ontario" with city name “Toronto” instead of “Waterloo”, but they are not capable of producing location proofs like “Canada, Alberta, Edmonton” or “USA, Maryland, Annapolis”. This property prevents malicious delegators and users from forging fake location proofs on their own. The second property protects users’ location privacy by hiding identities of delegators. An end user should not reveal the signature of the location proof since that will disclose the identity of delegators, which in turn will enable applications to obtain the most granular location information by looking up the location of the delegators. Instead, the user should execute a zero-knowledge proof protocol with the application to prove her possession of a location proof with a valid signature, instead of having to present the proof along with the signature.

Although the hierarchical structure of this credential delegation system neatly mimics the hierarchical relationship of geographical location, the system is impractical. A major hurdle of adopting a credential delegation system in practice is that it is difficult to design a mechanism where the location hierarchy formed by the location credentials can efficiently interact with organizational hierarchies. For example, a company may have several branches all over Canada, but the delegator of Waterloo should delegate only to APs of that company that are located in Waterloo region, which makes credential management more difficult.

Advantages and Disadvantages of Our Approach

Our approach relies on two trusted entities, the TTPU and TTPLs, to prevent the situation where a single entity simultaneously learns user identity and location information. In our approach, a user and an application need to trust a TTPL; but since there can be multiple TTPLs, the user and the application can try to find a TTPL that is trusted by both of them. Each TTPL has its own rules of deciding whether an AP should be trusted. For instance, a TTPL may choose to trust APs belonging to some high profile organizations or companies with outstanding public reputation, or it could place trust in APs whose user population is above a certain threshold, or it can decide to trust those APs that have their public keys certified by a certain CA, etc. Many more options for TTPLs to filter out untrusted APs are possible. Therefore, there is more flexibility in our approach than in the schemes mentioned above.

Nonetheless, our approach is not without shortcomings. Obviously, overhead associated with setting up and maintaining trusted third parties is inevitable. This is especially true for the TTPU; as already elaborated in section 3.5.1, a memcached data center is the most suitable choice for a scalable TTPU. Building a trusted third party of this scale is rather expensive.

3.5.4 Necessity of the CDA

We have investigated possibilities of replacing the CDA in the location proof architecture with appropriate algorithmic or cryptographic techniques. However, the fundamental difficulty in achieving this goal lies in the fact that whether two APs are nearby is not determined at proof issuing time but at proof verification time. Therefore, it is extremely difficult to find an appropriate encoding of APs' locations that will allow us to determine whether two APs are nearby at the verification time without revealing geographical location of the APs. More specifically, if we dispense with the CDA, then we must record location information of the APs in some form at proof issuing time, so that applications can investigate whether two far apart APs issued location proofs to the same user at roughly the same time. Since applications must not know the identities and exact location of the APs, the recorded information must not leak AP information but still enables the application to compute the closeness of the two APs. This task is rather challenging, if not entirely impossible, because it is decided by the application that how close is deemed “nearby” at proof verification time, but encoding of the APs' location is conducted by the APs or other parties at proof issuing time.

3.5.5 Achieve Extra User Anonymity Through Third Party Tools

Even though user privacy is an essential design consideration, privacy leakage is still possible in the our architecture. When a user communicates with a TTPL to obtain formal location proofs, although the TTPL is unable to learn the real identity of the user, nothing prevents it from tracking the user's IP addresses. Users can take advantage of anonymity tools such as Tor [32] or proxies to protect their privacy. The reason why we do not incorporate these components in the architecture is that users typically have different privacy requirements depending on their needs and preferences. For example, if the user demands high performance but cares less about her privacy, using Tor or proxies would be an overkill. Therefore, our architecture does not force users to make such strong privacy commitment, but users should adopt additional anonymity tools whenever appropriate.

3.6 Deployment Strategies

Our location proof architecture is suitable for enterprise networks where public APs are run by organizations or companies. For example, AT&T, which runs public APs all over the US, could deploy our architecture.

It is common for a large enterprise to run dozens of APs at the same time. For instance, chain stores may deploy an AP around each branch to allow customers to collect location

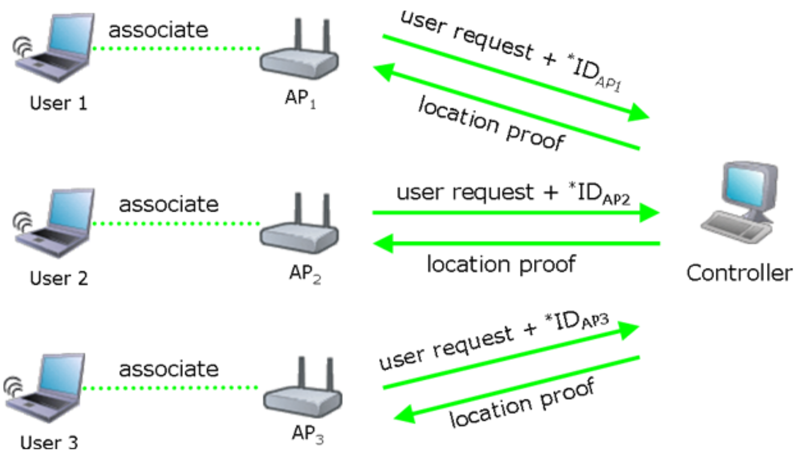


Figure 3.4: Deployment of location proof architecture for an enterprise network

proofs from these APs in order to participate in the stores' loyal customer program. If every AP belonging to the same company is locally installed with a location proof issuing system, any upgrade or patch on the location proof system will require changes on all the APs. This working model not only is error-prone, but also creates huge maintenance overhead for companies that operator these APs. For the sake of flexible management, a centralized controller provides a viable solution.

Figure 3.4 displays our proposed deployment configuration of the location proof architecture for an enterprise network. A central controller, which all the APs are connected to, is responsible for interacting with users and issuing location proofs. A user can associate with one of the APs and send a location proof request to the central controller through the AP. The AP, which the user associates with, sends to the central controller an association message informing the controller of the IP address of the user. Subsequently, the controller adds an entry to its association table to record the user's IP address along with that particular AP's ID, and the location proof issued to this user will be signed using that AP's private key. In some cases, the controller might have a specific IP address allocation method that allows it to automatically learn which AP the user is associated with based on the user's IP address. For instance, the controller might configure AP_1 to allocate IP addresses of range 192.168.1.1 to 192.168.1.100 and AP_2 to allocate IP addresses of range 192.168.1.101 to 192.168.1.200. Thus, if the request comes from a user with IP address 192.168.1.3, then the controller will know the user must be associated with AP_1. In this case, there is no need of managing an association table on the controller. In order to issue a location proof on behalf of APs, the controller must have the private keys of all the APs. Therefore, the private keys of the APs under the control of the controller are all stored on the controller.

This enterprise-level deployment strategy is used in our implementation and actual deployment of the complete architecture, which is detailed in section 4.2.

Chapter 4

Implementation and Experiments

4.1 Introduction

This chapter presents the implementation and evaluation of the complete location proof architecture. In section 4.2, we provide implementation details of the TTPL, the TTPU and the CDA, as well as the testbed where we deployed our architecture. In section 4.3, we summarize experimental results regarding the performance of our architecture. Finally, we present three real world applications that we developed that are based on our location proof architecture in section 4.4.

4.2 Implementation

In this section, we elaborate on our wireless testbed and implementation of three trusted third parties, as well as experimental results of the performance of the complete architecture. In our implementation, RSA (with key size 2048 bits) and AES [8] (with key size 256 bits) are chosen as the algorithms for all public-key and symmetric-key encryption respectively. In addition, SHA-256 is the hash function used for all the hashing operations. Since the current implementation is only meant for internal testing and experiments, we created self-signed certificates using OpenSSL [23] for the user, APs and the three trusted third parties, instead of trying to obtain real certificates from CAs. All these certificates are self-signed, and they are not signed by a common CA. Real-world applications should certainly rely on a formal PKI, however.

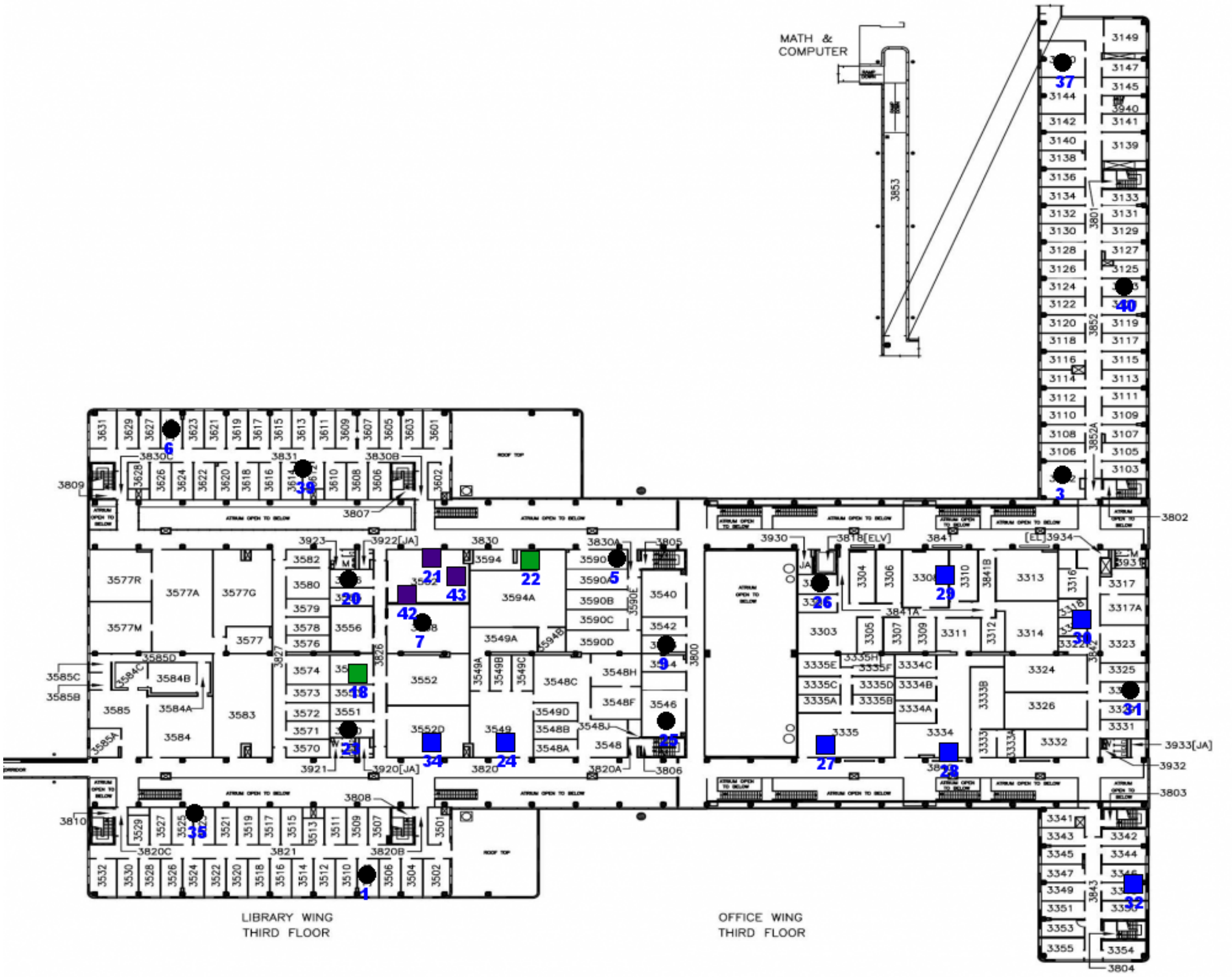


Figure 4.1: WLAN testbed on the 3rd floor (circles represent clients and squares represent APs. However, this is just annotation and the actual configuration can be changed by the person configuring the testbed)

4.2.1 Wireless Testbed

In order to deploy and test our location proof architecture, we adopt the WLAN testbed developed by Ahmed et al. [1]. A total of 38 AP nodes, all connected to and managed by a central controller, are deployed on the second and third floors of the David Center (DC) building at the University of Waterloo. The layout of these nodes on the third floor is shown in figure 4.1. This testbed consists of two major components which we detail next.

Central controller: The central controller is a desktop computer with a dual core 2.66 GHz and Gigabit connection with APs. It manages the system settings and configuration of all the APs. The controller is installed with Ubuntu 7.04 and runs the 2.6.20-15 Linux kernel. In addition, it supports NFS which allows the AP nodes to mount and share the same file system. The controller also stores the MAC to IP address mapping of each AP.

APs: APs are wireless nodes connected to the central controller. Each AP consists of a VIA EPIA EN12000EG mainboard with 1.2 GHz C7 nanoBGA2 processor. Each AP node also has two network interfaces, a Gigabit Ethernet interface whose IP address is drawn from the subnet 192.168.1.0/24 and a wireless interface whose IP address is drawn from the subnet 192.168.2.0/24 by the controller. Each AP is a NFS client which mounts the file system served by the central controller after booting from the network.

A user who wants to request a location proof from the central controller must first associate with one of the APs. After the association step is finished, the AP sends an association message to the central controller to inform it of the user's MAC address so that the controller can add an User-MAC-to-AP entry to its association table. The user can then interact with the controller through the AP. The AP will route the user's traffic to the central controller through its Ethernet interface. After the controller constructs a location proof, it signs the proof on behalf of the AP that the user is associated with.

4.2.2 Implementation of the TTPL

The TTPL web interface is hosted at `https://olten.cs.uwaterloo.ca/ttpl/index.py`, allowing users to use a web browser to interact with the TTPL and obtain formal location proofs. The TTPL web interface uses SSL to secure the communication with users. As mentioned before, the TTPL uses a self-signed certificate.

As shown in the figure 4.2, a dropdown menu lists available location granularities from 1 to 5, with 1 being the most coarse granularity and 5 being the most fine-grained granularity. * means a wildcard granularity. A user must choose a desired granularity from the menu and upload her informal location proof, based on which the TTPL will create a formal location proof. A formal location proof is returned to the user in a zipped archive. Behind

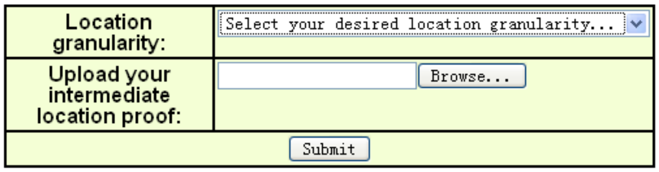

Location granularity:
Select your desired location granularity...
Upload your intermediate location proof:
Browse...
Submit

Figure 4.2: Web interface for the TTPL

the scene, the TTPL runs a MySQL database to manage AP information. The TTPL database contains location information of the recognized APs and their public keys.

When preparing a proactive location proof, the TTPU uses AES as the encryption algorithm for encrypting location information. The key size used is 256 bits and encryption executes in cipher-block chaining (CBC) mode.

4.2.3 The TTPL Database

In section 3.2.1 we introduced the idea of location granularity. The illustrative example in that section adopts a semantic form of location representation (for example, "Waterloo, ON, Canada") as opposed to GPS form of location representation (for example, 43°30'N 80°32'W, which is the GPS coordinate of "Waterloo, ON, Canada"). Semantic form of location is intuitive for human users, but in terms of facilitating cheating detection, it is not as useful and convenient as the GPS form representation. For example, it is difficult for the CDA to figure out whether the two locations "50 Columbia Street West, Waterloo, ON, Canada" and "43 University Avenue, Waterloo, ON, Canada" are close to each other. Therefore, we use the Universal Transverse Mercator (UTM) coordinate system [34] to convert each AP's location to the form of (x, y), and the TTPL stores (x, y) in the database as the location of the AP. However, in our implementation we hardcode the coordinates for simplicity and leave it as future work to implement conversion between semantic location and UTM coordinates.

However, this coordinate form of location representation gives rise to the issue of representing location granularity. With semantic, we can decrease the granularity of a particular location, say "43 University Avenue, Waterloo, ON, Canada", by simply stripping off its prefix. Fore example, we can remove "43 University Avenue" from the string "43 University Avenue, Waterloo, ON, Canada" if we want to decrease the granularity level by 1, and we strip off "43 University Avenue, Waterloo" if lower granularity is desired. With coordinate form of location information, we use the following idea to achieve location granularity.

We store the coordinate (x, y) of an AP as its most granular location. When a user specifies $g = 5$ in the formal location proof request, the TTPL simply returns (x, y). But

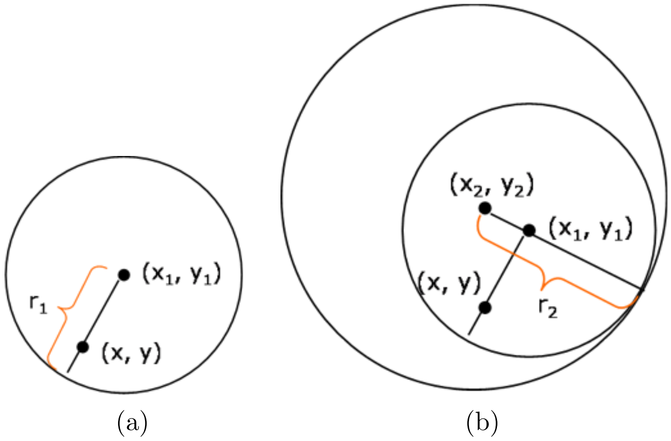
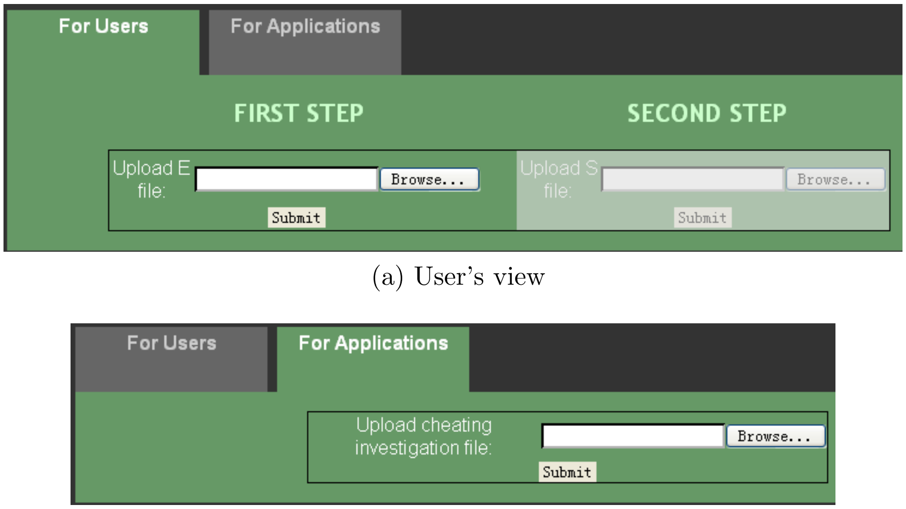


Figure 4.3: Granularity representation in coordinate system

for requests with lower granularity, say $g = 4$, we select a nearby point of (x, y), say (x_1, y_1), as the center of a circle and $r_1 = \text{distance}\left((x, y), (x_1, y_1)\right) + \triangle r$ to be the radius of the circle, where $\triangle r$ is a small value randomly generated by the TTPL. Then, the location with granularity $g = 4$ is represented by this circle. Note that we should not simply use (x, y) as the center of the circle, because that knowledge will enable the attacker to obtain the precise location of the AP fairly easily. Similarly, if we need to decrease the granularity level to $g = 3$, we select a random point, (x_2, y_2), in the circle represented by $g = 4$. Then, we compute $r_2 = \text{distance}\left((x_1, y_1), (x_2, y_2)\right) + r_1$. (Note that $\triangle r$ is not needed here, because the center of the old circle is already disguised by choosing the center of the new circle randomly. However, in computing r_1, if we eliminate $\triangle r$, then the user's most granular location is not disguised, because it is simply the center of the new circle. $\triangle r$ serves as a "start-up" disguising factor.) As a consequence, the new center (x_2, y_2) and the new radius r_2 now represent the location with granularity level $g = 3$. Similarly, $g = 1$ or 2 can be achieved in the same way. Figure 4.3 illustrates this idea.

The above algorithm for adjusting location granularity does not apply to semantic location, i.e. textual location information. For example, if a user stands on the Canadian side of Niagara Falls, then to decrease location granularity we may create a circle that covers both Canadian and US territory; in other words, the user is no longer able to prove she is in Canada. Since we are solely dealing with the coordinate form of location here, the algorithm still applies. We leave it as a future work to design an algorithm suitable for semantic location.



(a) User's view

(b) Application's view

Figure 4.4: Web interface for the TTPU

4.2.4 Implementation of the TTPU

The TTPU web interface for users and applications is displayed in figure 4.4a and 4.4b respectively. The TTPU server is hosted at `https://olten.cs.uwaterloo.ca/ttpu/index.py`. As mentioned before, the TTPU uses a self-signed certificate. In addition, the TTPU manages a MySQL database to store user registration information following the protocol specification in section 3.4.1.

A user needs to accomplish two steps before she is issued a TTPU token. When the user first clicks the "For Users" tab, she is presented two forms that correspond to the two steps of registration with the TTPU, and the second form is initially greyed out. The user needs to upload a file containing items required by the location proof protocol (in section 3.4.1) to the TTPU server. After checking the validity of the user's signature, the TTPU server inserts a temporary record about the user to the database and marks a special field of the record to indicate the registration is incomplete. The TTPU then returns the same page but with the second form enabled, so that the user can proceed to the second step of the registration. A generated random number is display on the top of the second form and highlighted in red. To complete the second step, the user must upload another file which contains her signature on this random number. Moreover, to limit wormhole attacks, the user must complete this step within 5 minutes after finishing the first step. Otherwise, the TTPU removes the temporary record of the user from its database and the user has to restart the registration again. If the second step is completed successfully, the user is returned a zipped archive that contains a TTPU token file. The TTPU then updates the user record in the database by changing the value of the special marker field to indicate the registration process is now complete.

An application can use the “For Applications” tab to upload a file that contains related information of the user who the application wants to conduct cheating detection against. Based on $\mathrm{ID}_{\mathrm{user}}$ and T_{TTPU}, the TTPU extracts a list of related encrypted $\mathrm{ID}_{\mathrm{AP}}$ from the database, and utilizes the web service provided by the CDA to investigate whether the user has cheated. The result of investigation is displayed on the page return to the application.

4.2.5 SOAP Versions of the TTPL and TTPU

For better script automation and facilitating the experiments, we also implement the SOAP [29] server version of the TTPL and TTPU. The TTPU exposes two public functions, **register_step_one** and **register_step_two**, under the name space `https://olten.cs.uwaterloo.ca/TTPU_Service`, and these two functions correspond to the two TTPU registration steps. The TTPL SOAP server offers a public function, **issue_proof**, under the name space `https://olten.cs.uwaterloo.ca/TTPL_Service`, through which users can use a SOAP client to request formal location proofs.

4.2.6 Implementation of the CDA

The only role that the CDA plays is to offer cheating detection service to the TTPU, and it does not interact with users or applications directly. Therefore, we provide no user interface for the CDA, since there is no such need. Instead, we implement the CDA as a SOAP server that offers cheating detection web service using SOAP protocol. The SOAP server handles requests arriving on port 4543. Moreover, it uses SSL to secure communication with the TTPU server. The same as the TTPU and TTPL server, our CDA server is currently using a self-signed certificate.

The CDA SOAP server offers a public function called **detect_cheating** under the name space `https://olten.cs.uwaterloo.ca/CDA_Service`. This function accepts a list of $\mathrm{ID}_{\mathrm{AP}}$ encrypted with the CDA’s public key and returns a 3-tuple, whose first field indicates the exit code of the function, the second field indicates the result of cheating detection and third field provides optional extra information. The exit code has two possible values, 0 being success and -1 being failure. If the exit code is -1, the second field is the error message that provides detailed information about anomaly. The third field is an empty string in this case. On the other hand, the exit code 0 indicates successful execution of the function. In this case, the second filed is a Boolean value, indicating whether any two APs in the list are geographically far apart. If the second field is “true”, the third field further specifies the two adjacent APs. Otherwise, the third field is an empty string. It is worth mentioning once again that the CDA does not maintain an AP database on its own; instead, it makes use of the AP databases published by TTPLs.

In addition, the TTPU uses a SOAP client to leverage the service provided by the CDA server in order to conduct cheating detection. Typically, the TTPU assembles a list of $\mathrm{ID}_{\mathrm{AP}}$ and invokes the **detect_cheating** function with this list as the parameter using SOAP protocol.

4.2.7 Computing Proximity of Two Locations

When the CDA receives a list of encrypted AP IDs from the TTPU, the CDA carries out cheating detection by examining whether any two APs in this list are far apart. Since TTPLs make their AP location databases publicly available, the CDA first consults the appropriate TTPL database and converts the list of AP IDs into a list of corresponding coordinates. The CDA then computes the distance of every two coordinate points and checks whether the result falls within a threshold D. If the distance between any two points is greater than D, those two APs are considered too far apart. Moreover, since this list of APs are contacted by the user within a short period of time, any two APs in the list should be relatively close to each other. Therefore, existence of two far apart APs in the list allows the CDA to conclude the user has cheated. How far is considered far apart depends on the user's moving condition, but does not depend on the location granularity at which the proof was revealed. The cheating detection procedure concerns whether a user can obtain proofs from two different APs without violating basic facts. For example, if a user obtains a proof in the city of Toronto at time t_1 and another proof in the city of Ottawa at time t_2, and the difference between t_1 and t_2 is only 5 minutes, then the user definitely has cheated (such as by launching a wormhole attack).

The CDA has a default value for the threshold D, which assumes the user moves at the speed of an airplane. More specifically, $D = \triangle t * V$, where V is the speed of a regular commercial airplane and $\triangle t$ is specified by the application. Thus, this threshold value should cover almost all possible moving conditions of the user, unless the user is an astronaut and frequently travels by space shuttle. However, the application can also provide a more appropriate threshold value for the CDA to use. As a matter of fact, the application is encouraged to do so, since it is more likely to have knowledge of user's condition, i.e. whether the user frequently drives in a car or walks on foot while requesting location proofs. Accordingly, the application can submit a more reasonable threshold value for computing whether two locations are considered far apart. For example, passing two APs that are 5000 meters away from each other within 5 minutes is quite normal for a person driving in a car, but is entirely impossible for a person walking on foot.

4.2.8 Programming Libraries

All three trusted third parties are hosted on the olten server at the University of Waterloo. The olten server has Ubuntu 8.04 with Linux kernel 2.6.24-24-server installed and runs on two 2.40GHz Intel Xeon(R) CPUs with 3.9 GB memory. Moreover, all three trusted third parties are implemented in Python programming language. We use SOAPpy library [30] to implement SOAP service offered by the TTPU, TTPL and CDA. This python library has SSL support for SOAP server and client. As to all the cryptography algorithms used in our programs, we adopt a popular cryptography toolkit called PyCrypto [25]. This library supports various symmetric key (including AES) and public key (including RSA) schemes. Furthermore, it provides various popular hashing algorithms (including SHA-256, which is extensively used in our location proof protocols).

4.3 Experiments

4.3.1 Motivation

In practice, APs often have to interact with users on a regular basis and typically are weak devices that could potentially throttle the performance of the location proof architecture. Therefore, we provide experimental results of performance measurement and analysis in this section. We first measure the throughput of the controller and pinpoint the performance bottleneck. Then, we compare our implementation with popular OpenSSL implementation of cryptographic functions to evaluate the quality of the programming library we rely on.

However, we do not conduct TTPL and TTPU-related performance measurement, because results of such measurement will not be informative based on our current setup. In practice, neither the TTPU nor the TTPL will be run on a single server. As we discussed in section 3.5.1, the TTPU will typically be deployed on a memcached data center to ensure scalability. Moreover, there will also be multiple TTPLs to allow applications and users to choose the one they both trust. Despite lack of measurement of TTPLs and the TTPU, performance of these two entities is less of a concern, since they are more powerful compared to APs in practice. We leave it as a future work to investigate how large of a server farm we would need for the TTPU and TTPLs.

4.3.2 Measurement of Controller Throughput

The controller is the master of all the APs and responsible for issuing location proofs. Therefore, it is the potential performance bottleneck of the architecture. We conduct

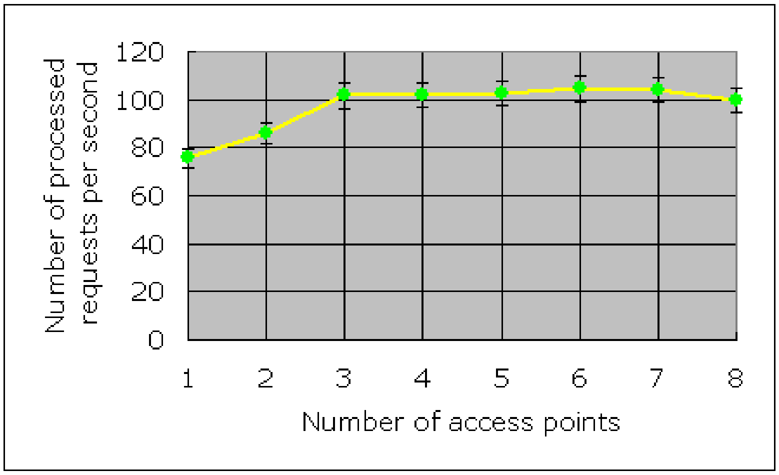


Figure 4.5: Controller throughput

a series of experiments to measure the throughput of the controller and determine the scalability of the controller as the number of location proof requests increases. We first need to eliminate the step of user registration with the TTPU since it has no impact on the controller throughput. We manually prepare TTPU tokens and save them to files for the user-side program to read locally instead of having to obtain from the TTPU in real time. Another factor that could potentially skew the result is the delay between APs and users. We eliminate this delay by running the user-side program on the APs, so that location proof requests are directly issued from APs. The delay between the APs and the users does exist but vary depending on the signal strength, power, noise, connectivity of the users' network cards, and so on. Therefore, including this delay in the measurement will make the result unreliable and less useful. We intend to investigate the controller's full capability of issuing proofs when the delay between the APs and the users is 0. In practice the real throughput will be below this value because of delay, so the result obtained here serves as an upper bound.

We conduct six rounds of experiments with increasing number of APs. For every round of the experiment, we run the controller for 15 minutes, during which we instruct a certain number of APs to send location proof requests persistently and as fast as possible. At the same time the controller keeps track of the number of processed requests so far. We repeat each round 10 times to compute the average number of requests that the controller is able to handle per second. The experimental results are displayed in figure 4.5. As shown in the figure, the performance of the controller reaches its peak after we run three APs and more. Moreover, during the experiment we monitored the CPU usage which was kept at 100%. Therefore, we can be certain that network latency is not the bottleneck here.

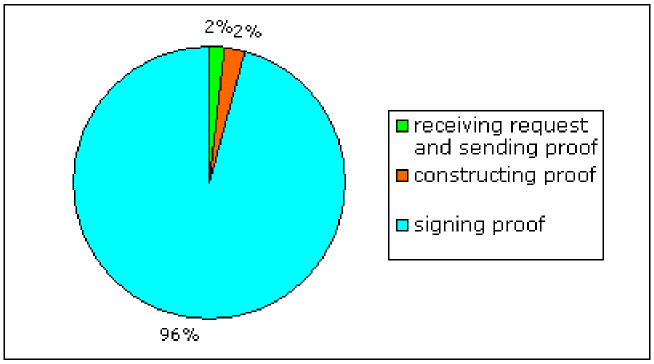


Figure 4.6: Time contributed by each part of serving a request

4.3.3 Micro-Benchmarking Performance Bottleneck

Now that we have determined it is our program on the controller, instead of network latency, that is the bottleneck, we need to find out which part of the program significantly affects the performance. Generally speaking, creating cryptographic signatures is an expensive operation and is the most likely culprit. In order to determine whether it is the performance bottleneck of the architecture, we conduct the following experiment.

We divide the time that the controller spends serving a single location proof request into three parts: (1) receiving the request and sending the proof. This part measures the cost associated with network activities. "Receiving the request" refers to the time between accepting the user connection and receiving the last bit of the request message. "Sending the proof" refers to the time between transmitting the first and the last bit of the location proof along with the signature. (2) constructing the location proof and (3) signing the proof. We control an AP to send 100 location proof requests to the controller and measure the time contributed by each part. Since all the APs use the same NFS provided by the controller and are configured with the exact same hardware and network connectivity, the choice of AP is irrelevant. Figure 4.6 summarizes the results. It is clear from the figure that the most time-consuming part is proof signature creation, which is 23 times more expensive than the other two parts combined. In terms of the numerical value of the cost, the following table summarizes the results.

	Part 1	Part 2	Part 3
time (in ms)	0.317 ± 0.003	0.387 ± 0.004	15.02 ± 0.09

Part 1 stands for receiving the request and sending the proof, part 2 stands for constructing the proof and part 3 stands for signing the proof.

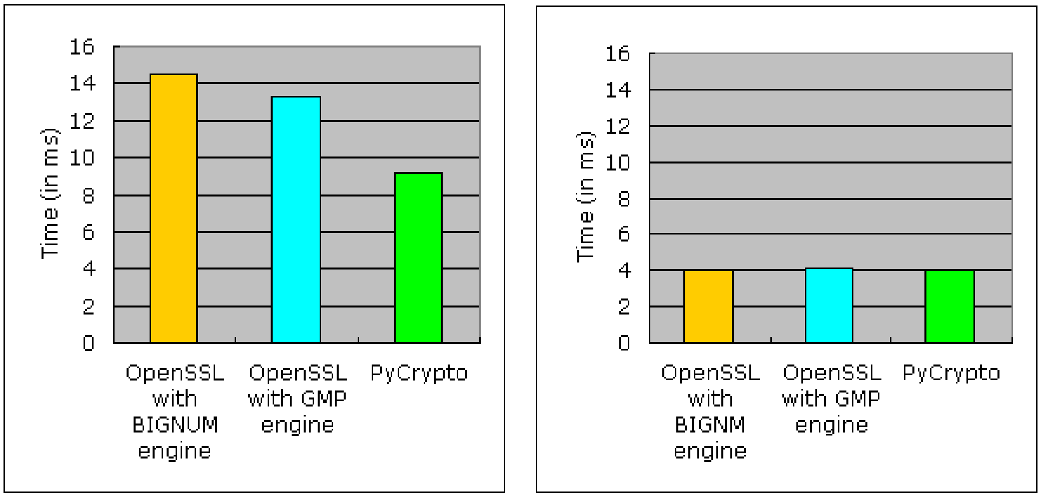
4.3.4 Benchmarking PyCrypto and OpenSSL

Now that we have pinpointed the exact spot of performance bottleneck, we further investigate whether our programming choice is inappropriate, given that interpreted languages like Python are considered significantly slower than compiled languages such as C. Since we rely on PyCrypto library's implementation of RSA for creating cryptographic signatures, we compare it to the popular OpenSSL implementation of RSA. OpenSSL already provides an easy-to-use command to benchmark its implementation of RSA signing, so we simply invoke it and record statistics reported by OpenSSL. To benchmark PyCrypto's implementation of RSA signing, we use the standard Python benchmarking module cProfile and timeit. We conduct benchmarking tests on two computers, one of which is the controller and the other of which is the olten server. Both computers are installed with PyCrypto 2.0.1, and OpenSSL 0.9.8k. The default OpenSSL engine, i.e. the BIGNUM engine, is slower than the GMP [14] engine due to a number of countermeasures implemented against various PKI vulnerabilities. Thus, we run our benchmark tests using the GMP engine as well after the default BIGNUM engine is benchmarked. By measuring the speed of creating RSA signature using a 2048-bit key, we find that PyCrypto outperforms OpenSSL on both machines according to figure 4.7. More specifically, PyCrypto runs faster than OpenSSL on the controller by 36.75% when the default BIGNUM engine is used and by 31.25% when the GMP engine is enabled. However, on the more powerful olten server, PyCrypto performs the same as OpenSSL. This result is somewhat unexpected since Python is believe to be slower than C programming language which is used to implement OpenSSL. In fact, PyCrypto library implements its speed-critical operations in C and relies on the high performance GMP library for fast arbitrary precision arithmetic. Therefore, the performance disadvantage of the python library compared to C implementations diminishes in this particular case. In summary, our choice of PyCrypto library is no worse than the popular OpenSSL implementation. To improve performance of the implementation, we could choose other faster signature schemes such as ECDSA [19]. However, the crypto library our implementation relies on does not yet support ECDSA, so we leave it as future work to measure how the performance improves after switching to a faster signature scheme.

4.4 Real World Applications

Our architecture is suitable for two kinds of applications: (1) applications that are low-value and where executing a wormhole attack or borrowing-a-device attack would be more expensive, and (2) applications that already have some sort of location check and where our architecture gives better security.

In the rest of this section, we present two real world applications we built using the complete location proof architecture. Note that the location proof daemon below is not a



(a) Benchmarking on the Controller

(b) Benchmarking on the olten server

Figure 4.7: Benchmarking of PyCrypto and OpenSSL

location proof based application; instead, it is a client-side program on users' computers that automates the task of requesting location proofs.

4.4.1 Location Proof Daemon

We implemented a client-side daemon program that automates the procedure of requesting location proofs. More specifically, the daemon reads a local configuration file on startup, where a few parameters such as the address of the TTPL and TTPU SOAP servers, desired location granularity, how often to request a location proof, etc., are specified. The daemon runs in the background after initialization. It sends location proof requests periodically and saves location proofs to the user's computer. Moreover, the daemon allows the user to dynamically change these settings through a frontend command shell. The user is also able to instruct the daemon to send a location proof request at once instead of waiting until the next scheduled request. With the help of this daemon program, we are able to develop three real world applications that make use of our location proof architecture. Since the next three applications rely on this daemon program to request location proofs, we assume this daemon program is installed on the client's computer.

4.4.2 Class Attendance Reporter

School teachers often take students' class attendance into account when assigning final grades. However, managing attendance records could be annoying and troublesome, especially for a large class with hundreds of students. Traditional procedures for teachers to take class attendance include calling out loud students' names at the beginning of the class and marking the name after hearing a response, or passing around an attendance sheet for students to sign and collect the sheet at the end of the class. These traditional methods are not reliable since students could answer the teacher's calling or sign attendance sheet for their friends and it is hard for teachers to notice. Some schools try to use the *Clicker* system as a new way of taking class attendance. At a school where the Clicker system is used, students who have registered for a course are also required to register their clickers. Students must bring and use their clickers when taking a quiz, signaling attempt to answer a question in class, etc. When a student uses her clicker in class, the system will automatically record her attendance. Students typically have to buy and register a clicker in this case. Instead, our location proof architecture provides an alternative where students can conveniently use their mobile devices in hand to prove their attendance.

We developed a class attendance reporter program that allows students to prove their attendance to the teacher conveniently. The program has two components: a student module that is installed on the student's laptop or mobile devices. When the student brings her laptop or mobile devices (such as a blackberry) to the class, the program instructs the location proof daemon to request a location proof during the class period and sends the proof to the teacher module of the program, which is installed on the teacher's computer. In addition, the teacher module generates formatted web pages for the teacher to browse which students have reported their attendance. Students can always choose to leave the class after having proved their locations, and our application does not aim to solve this problem.

4.4.3 Location Signature For Yelp Web Review

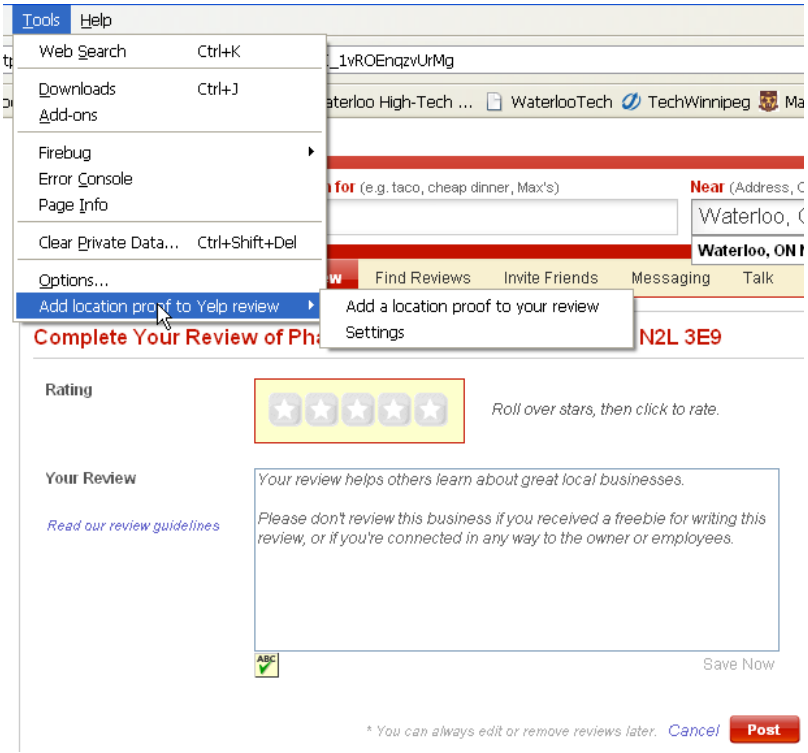
Yelp [36] is a social networking and user review website with over 25 million visits per month. Users can provide rating and reviews for various places on the Yelp website. In early 2010, Yelp released a collection of location features for its iPhone application that allows users' reviews to be supported by their location at the time of composing the review. More specifically, a Yelp user can "check-in" through Yelp's iPhone application before leaving a review on the Yelp website, then after submitting the review the user's current location will be shown alongside with her review. This feature dramatically increases the creditability of reviews, since the users who can prove they actually made several visits to the reviewed place are more likely to provide insightful comments regarding the place. This feature is currently only available to iPhone users.

Our architecture is a good fit for enabling this feature for all the users that access Yelp's website, not only because we can integrate location proof to the Yelp website with the help of a browser plug-in, but also because our architecture improves security. By jailbreaking an iPhone, an attacker could tamper with Yelp. In contrast, our architecture does not learn users' locations simply based on GPS, which is used by Yelp; instead, the architecture requires users to prove their physical presence at the location using sophisticated protocols to guard against various attacks and cheating.

We developed a Firefox extension that lets users to attach a location proof to their review on the Yelp website. Any user that wants to use this feature must download our extension. A user composes a review on the Yelp website as usual. Before submitting the review, she attaches a location proof to her review using the extension as shown in figures 4.8 and 4.9. The extension generates a review identifier which is used to locate this specific review in the page later on. The extension then submits the name of the place under the review, the location proof and the review identifier to a remote server, which is currently hosted at the University of Waterloo. The remote server is responsible for storing location information associated with Yelp reviews. Upon receiving the location proof, the server first verifies it. If the proof is invalid, the server rejects the request. Otherwise, the server extracts location information from the proof and investigates the location of the reviewed place using the Google map service. If the location represented by the location proof is indeed near the location of the reviewed place, the server stores the extracted location information along with the corresponding review identifier. Otherwise, the server stores the extracted location information with an appended string which warns that the location proof represents a non-nearby location to the reviewed place (therefore, the user's review is questionable). Once this process is complete, the server notifies the extension. The user now submits her review and the review identifier to the Yelp website. Note that the review identifier is automatically added to the user's review by the extension. Later when a user tries to view the page, assuming she has the extension installed, the extension will automatically retrieve from the server location information associated with each review on the page and display it along with the review, as shown in figure 4.10.

The Yelp review-location server consists of three parts: a web server component, a SOAP server component and a database, as shown in figure 4.11. The SOAP server component is responsible for receiving users' requests of associating a location proof with a review. Typically, the SOAP server component verifies the uploaded location proof and stores verified location information in the database. The duty of the web server component is to retrieve from the database location information corresponding to review identifiers submitted by users. The Yelp DB component stores location information associated with Yelp review identifiers. The motivation of having two separate components, instead of one, to serve user requests of upload and download is to improve performance. Location proof verification requires massive computation and thus is CPU-bound, but downloading location information mainly involves database access which is IO-bound. Therefore, we build two separate components to serve upload and download requests.

Finally, the functionality of using Google map service to convert the name of the re-



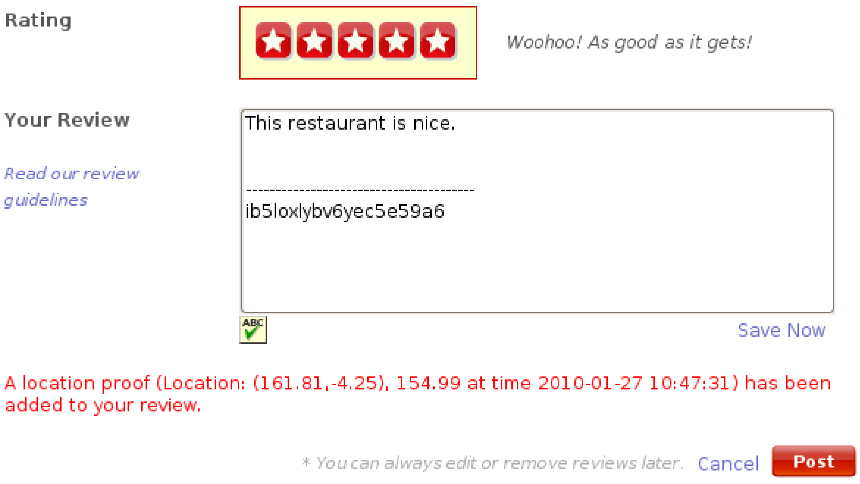
Roll over stars, then click to rate.

Your review helps others learn about great local businesses.

Please don't review this business if you received a freebie for writing this review, or if you're connected in any way to the owner or employees.

* You can always edit or remove reviews later.

Figure 4.8: Attach a location proof to a Yelp review



Woohoo! As good as it gets!

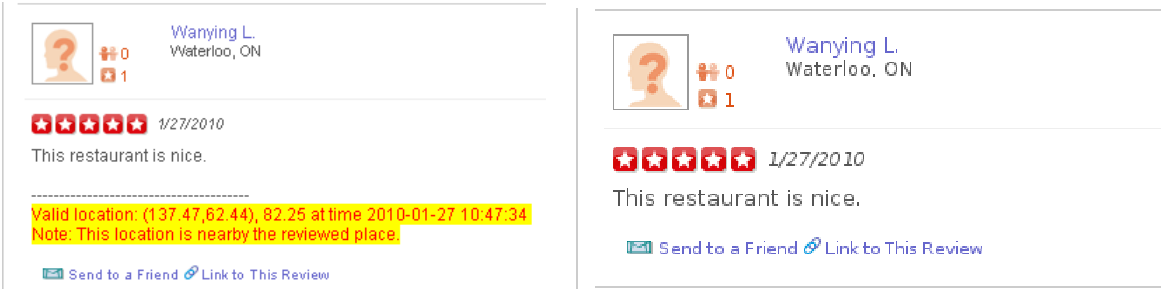
This restaurant is nice.

----
ib5loxlybv6yec5e59a6

A location proof (Location: (161.81,-4.25), 154.99 at time 2010-01-27 10:47:31) has been added to your review.

* You can always edit or remove reviews later.

Figure 4.9: Notify the user that location proof has been attached to the review



Wanying L.
Waterloo, ON

1/27/2010

This restaurant is nice.

Valid location: (137.47,62.44), 82.25 at time 2010-01-27 10:47:34
Note: This location is nearby the reviewed place.

(a)

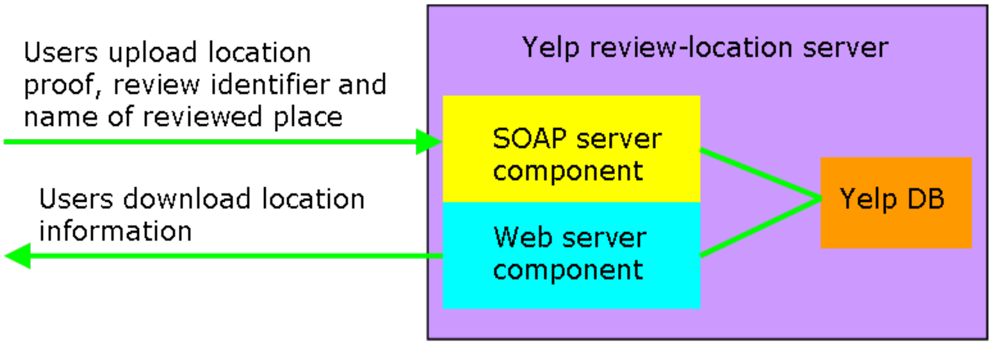
Wanying L.
Waterloo, ON

1/27/2010

This restaurant is nice.

(b)

Figure 4.10: Yelp page with and without location proof



Users upload location proof, review identifier and name of reviewed place
Users download location information
Yelp review-location server
SOAP server component
Web server component
Yelp DB

Figure 4.11: Yelp server architecture

viewed place to its GPS coordinates is in fact undone and left as a future work. For the time being, we hardcode the coordinates of the city of Waterloo in the server and only experiment with places in Waterloo.

Chapter 5

Related Work

5.1 Related Work

Denning and MacDoran [10] present a location-based authentication system where a location signature sensor (LSS) creates location signatures from satellite signals that describe the physical location of the LSS at a particular time. A user carrying an LSS can hand a location signature to an application, which compares it to the signature that the application derived from the satellite signals. This system lacks a strong binding between the location signature and the identity of the user. Therefore, a user can abuse location signatures by selling them to anyone. Moreover, the system relies on dedicated hardware and supports only retroactive location proofs. In comparison, our architecture ties location proofs to specific users, does not require dedicated hardware, and also supports proactive location proofs.

Brands and Chaum [3] elaborate a distance bounding protocol in which a verifier V computes an upper bound on the distance to a prover P by measuring the round-trip delay of a challenge-response protocol. The high level idea of the protocol is that V and P both generate k bits randomly at the beginning of the protocol. V sends her bits to P one at a time, to which P responds with her bits one by one as well. V measures and records the round-trip delay for each round of bit exchange. After having received all k bits from V, P sends V a message encrypted with a secret key which is concatenation of V's and P's bits. If the message is valid, V computes the upper-bound on the distance to P using the measured k delays. However, the protocol does not let P and V exchange identity information. In other words, a malicious prover can claim whoever she wants to be, which is problematic for many applications that require user identity information. In contrast, our architecture requires users to sign and submit a random nonce generated by the location proof issuer and this signed nonce will be verified by the application when the user presents the location proof to the application. Based on the assumption that no user will share her private key, it is impossible for a user to impersonate others. Thus, our architecture ensures users must be physically present at the target location when requesting location proofs.

Walters and Felten [35] present a system that allows a device to obtain location proofs from a location manager (LM) and to submit proofs to a verifier. A device starts to request location proofs by sending out a message containing its device ID encrypted with the public key of the verifier. Upon receiving the request, the LM sends the device a nonce, and measures the round-trip delay between sending the nonce and getting the response from the device. It then encloses the measured delay and the encrypted device ID into a location proof and delivers the proof to the device. Finally, the device presents the proof to a verifier. The use of device IDs as identities of the recipients of location proofs in this system is problematic, because a malicious user can craft fake device IDs such that she will be able to impersonate anyone else. In comparison, our architecture mandates the recipient of the proof to commit to some value that can be verified by the application later on. Therefore, no one can impersonate others or be impersonated. Another drawback of this system requires the device to know target verifiers in advance in order to encrypt device IDs with appropriate public keys. Location proof issuers in our architecture send out location proofs which are designed to follow a general format, so that they can be used for various applications and users do not have to know target verifiers before hand.

Sastry et al. [27] develop a secure location verification protocol, named Echo, which is used in location-based access control. The authors concentrated on the in-region verification problem where a verifier V tries to verify a prover P is in a region R near V. The basic idea of the protocol is that: (1) a prover P sends its claimed location L to a verifier V using radio frequency (RF), (2) V sends a nonce to P using RF as well, (3) P immediately echoes the nonce back using ultrasound. By timing this process, V either accepts or rejects the location claim based on the computed upper-bound of the round-trip delay. By exploiting the property of speed of light and sound, the protocol avoids relying on complicated cryptography. However, the identity of the prover is not known by V, so the system can only be used by applications that take user location as the sole consideration. As argued above, our architecture provides much stronger guarantee on the identity of the recipients of location proofs. Furthermore, for this system to work, P has to have the capability of using RF and ultrasound, which rules out many regular users whose devices are only equipped with basic wireless cards. By contrast, our architecture does not require any dedicated hardware and is usable by users with simple mobile devices.

Faria and Cheriton [12] design a location-based authentication architecture for wireless LANs. A centralized wireless appliance (WA) controls a group of APs to provide location-based access control for a LAN. In order for a client to authenticate to the LAN, the client must prove its proximity to one of the APs by executing a handshake protocol with the WA. More specifically, the WA generates a set of random nonces and broadcasts them through one of the APs. To prove its closeness, a client must send all the received nonces back to the WA. A far-away attacker has a much greater chance of receiving corrupted nonces, so she is more likely to send incorrect nonces thus unable to prove her proximity. Since no identity information of the client is transmitted during the process of handshake, this architecture does not guarantee the identities of clients. In addition, the architecture depends on the property of wireless LANs: the further away the user, the greater the chances of receiving corrupted packets, which is not a reliable approach of proving location. In comparison, our architecture takes both user identities and location into account. Besides, we use cryptographic techniques to make the protocol more reliable instead of succeeding by chance.

Traynor et al. [33] design a two-phase localization protocol which allows a system to determine a user's location in both coarse and fine granularity. During the first phase, the regionalization phase, multiple APs broadcast a set of unique tokens. A user close to an AP can capture the token and send it back to the AP controller (APC), which is able to determine which AP the user is in the proximity of based on the unique location "fingerprint" of the token. If finer-grained location information is demanded, the protocol enters the second phase, the localization phase, during which the system uses multiple Bluetooth devices to transmit a new set of tokens. Similarly, by capturing a token and sending it back to the APC, the user can prove her location of a higher granularity. The primary drawback of this protocol is lack of user authentication and time stamp recording, that is, the protocol does not associate a particular proof with appropriate user identity and time information. This shortcoming allows a malicious user to lie about her location by capturing a token when residing around a certain AP but submitting it to the APC after moving to a different location, or she may also give away the token to a remote user who is not entitled to the proof. Our location proof protocol eliminates these problems by including both user identity and a time stamp in location proofs. Moreover, not all devices have Bluetooth, which means their approach may not always apply. In contrast, our architecture does not rely on Bluetooth, although devices with Bluetooth can also benefit.

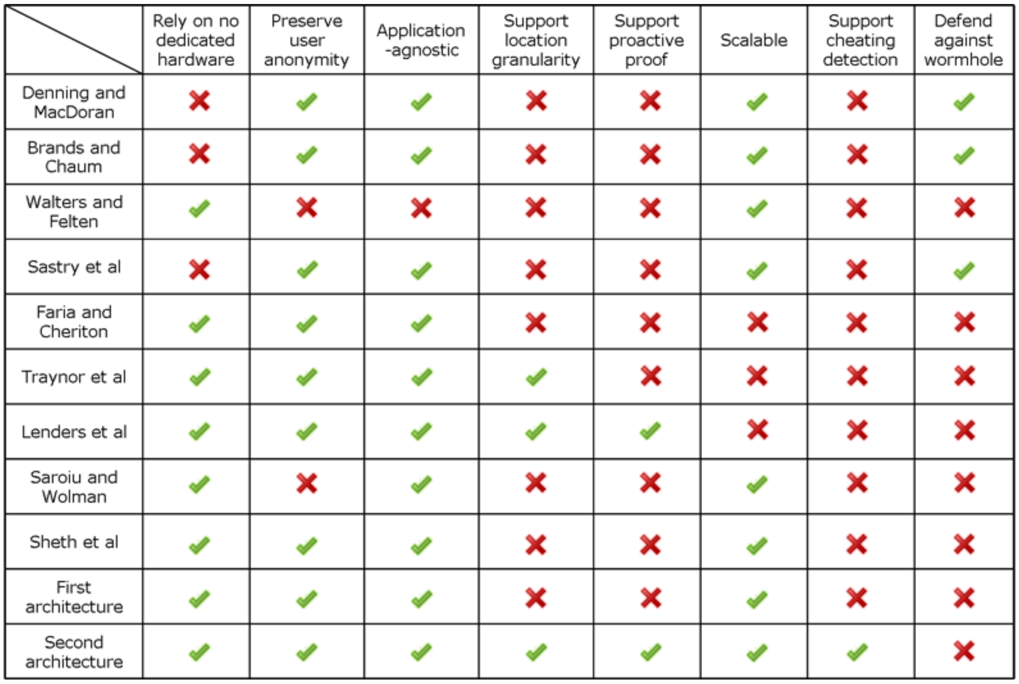
Lenders et al. [21] describe a geotagging service that allows the publisher of mobile user-generated content to obtain a location-and-time-stamp certificate for the content. Such a certificate proves where and when the content was generated. Consequently, those who downloaded the content along with the certificate can verify the source location and time stamp associated with the content. There is a drawback of this system compared to ours. The location/time certificates in their system do not bind the content and the certificate to the content originator; in other words, anyone can claim or disclaim her ownership of the content and its certificate. In contrast, our architecture achieves stronger binding between the location proof and its owner, while still keeping the anonymity of the owner to the proof issuer.

Saroiu and Wolman [26] propose a mechanism for mobile devices to acquire location proofs from APs and to use them for interaction with location-based applications. In their mechanism, Wi-Fi APs broadcast beacons to announce their presence. A client wishing to obtain location proofs must enclose its own identity and the sequence number extracted from the beacon in a signed request and send it to the AP. After checking the validity of the client's signature, the AP sends to the client a location proof containing the identities of the client and the AP itself, a time stamp, and the location of the AP. A drawback of this mechanism is that APs get to decide what granularity of location information to include in the proof. When the user only needs a proof for proving her rough location, a fine-grained location proof reveals more information than necessary, and is thus harmful to the user's location privacy. Another problem is the lack of privacy protection for users who directly expose their IDs to the AP, which may track clients by their location. Our design effectively addresses these two problems. In our architecture, a user can request a proof that includes encrypted location information of different granularities. When presenting the proof to the application, the user can selectively reveal location information of appropriate granularities. Moreover, in our complete architecture the identity of the user is disguised from the AP, so the location privacy is not compromised while the usability of location proofs is preserved. Our lightweight architecture suffers from the same problem of disclosing user identity directly to APs.

Sheth et al. [28] present geo-fencing, a technique to confine Wi-Fi coverage to a certain physical boundary. To make Wi-Fi usable only inside the geo-fenced region, packets are coded across all the APs in the region. Users outside the region are unlikely to receive signals from all the APs, thus unable to re-construct valid packets, while users inside the region are able to receive valid packets. This technique is aimed to ensure the availability of some service is not leaked to outside a certain geographical region, instead of providing a location proof mechanism. The guarantee on users' presence within the geographical confinement is achieved by exploiting the limitation of the range of Wi-Fi signal transmission. Our architecture facilitates a user to obtain location proofs at one location and use them at another place which might be far away from the place where the proof was issued. Therefore, location proofs do not necessarily have to be used only within the issuing location in our architecture.

Ole Tippenhauer et al. [31] analyze the security of a public WLAN positioning system. In particular, they demonstrated how location spoofing can be leveraged to impersonate a legitimate AP and convince a device of a fake location where it is not currently located. Furthermore, the authors proposed approaches to detect this kind of attacks to acquire localization signals of the APs securely. Although this work is not about inventing new location proof systems, it provides indispensable insights and suggestions for achieving security of location proof architectures.

Figure 5.1 shows the comparison of our location proof architectures with previous work.



	Rely on no dedicated hardware	Preserve user anonymity	Application -agnostic	Support location granularity	Support proactive proof	Scalable	Support cheating detection	Defend against wormhole
Denning and MacDoran	✗	✓	✓	✗	✗	✓	✗	✓
Brands and Chaum	✗	✓	✓	✗	✗	✓	✗	✓
Walters and Felten	✓	✗	✗	✗	✗	✓	✗	✗
Sastry et al	✗	✓	✓	✗	✗	✓	✗	✓
Faria and Cheriton	✓	✓	✓	✗	✗	✗	✗	✗
Traynor et al	✓	✓	✓	✓	✗	✗	✗	✗
Lenders et al	✓	✓	✓	✓	✓	✗	✗	✗
Saroiu and Wolman	✓	✗	✓	✗	✗	✓	✗	✗
Sheth et al	✓	✓	✓	✗	✗	✓	✗	✗
First architecture	✓	✓	✓	✗	✗	✓	✗	✗
Second architecture	✓	✓	✓	✓	✓	✓	✓	✗

Figure 5.1: Comparison to previous work

The first seven columns list our design goals formalized in section 2.2, and the last column is an important feature that we believe a useful location proof architecture should also strive to support.

Chapter 6

Conclusion and Future Work

We have formalized seven design goals that should govern the construction of a location proof architecture. We elaborate on the importance of these design goals with regard to their roles in system functionality and their implications on user privacy. Among these design goals we formally explore user privacy protection, location granularity control, proactive location proof and cheating detection, which lack sufficient attention in previous work. We provide our insights on how these design goals can be realized by putting forward a location proof architecture that meets a partial list of the goals and another one that meets all of the design goals. In both architectures, we illustrate how cryptographic techniques can aid in location proof architecture design.

Our work leaves several directions for future work. First, wormhole attacks are not fully defended against. Our architecture addresses defense against a special case of wormhole attacks but does not provide a general solution to the threat. As mentioned in previous sections, the only effective technique against wormhole attack so far relies on dedicated hardware. Therefore, further research of exploring solutions that do not rely on dedicated hardware is demanded. Second, conversion between semantic location and UTM coordinates needs to be implemented. For the time being, we hardcode the coordinates in the implementation for simplicity. Third, as mentioned in section 4.2.3, an algorithm for adjusting granularity of semantic location is needed, since our research here deals with location information in the form of coordinates. Lastly, the RSA signature scheme used in our implementation should be replaced by other faster signature schemes to obtain improved performance.

References

[1] Nabeel Ahmed and Usman Ismail. Designing a high performance WLAN testbed for centralized control. *Testbeds and Research Infrastructures for the Development of Networks & Communities, International Conference on*, 0:1–6, 2009. 38

[2] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 108–125, Berlin, Heidelberg, 2009. Springer-Verlag. 31

[3] Stefan Brands and David Chaum. Distance-bounding protocols. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 344–359, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc. 54

[4] D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT '91: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 257–265, April 1991. 30

[5] Yih chun Hu, Adrian Perrig, and David B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, 2001. 9

[6] Danny De Cock, Christopher Wolf, and Bart Preneel. The Belgian Electronic Identity Card (Overview). In *Sicherheit*, pages 298–301, 2006. 7

[7] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 211–224, New York, NY, USA, 2008. ACM. 21

[8] J. Daemen and V. Rijmen. AES Proposal: Rijndael. 1999. 36

[9] George Danezis, Stephen Lewis, and Ross Anderson. How Much is Location Privacy Worth? In *Online Proceedings of the Workshop on the Economics of Information Security Series (WEIS 2005)*, 2005. 6

[10] Dorothy E. Denning and Peter F. MacDoran. Location-based authentication: grounding cyberspace for better security. pages 167–174, 1998. 54

[11] Saar Drimer and Steven J. Murdoch. Keep your enemies close: distance bounding against smartcard relay attacks. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, Berkeley, CA, USA, 2007. USENIX Association. 9

[12] Daniel Faria and David Cheriton. No Long term Secrets: Location Based Security in Over Provisioned Wireless LANs. In *HotNets-III: Proceedings of the Third ACM Workshop on Hot Topics in Networks*, November 2004. 55

[13] Foursquare. `http://foursquare.com/`. 2

[14] GMP. `http://gmplib.org/`. 47

[15] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984. 10

[16] Gowalla. `http://gowalla.com/`. 2

[17] Baik Hoh, Marco Gruteser, Ryan Herring, Jeff Ban, Daniel Work, Juan-Carlos Herrera, Alexandre M. Bayen, Murali Annavaram, and Quinn Jacobson. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 15–28, New York, NY, USA, 2008. ACM. 21

[18] VeriSign Digital ID. `http://www.verisign.com/authentication/individual-authentication/digital-id/`. 7

[19] Don Johnson and Alfred Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, 1999. 47

[20] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, 2005. 13

[21] Vincent Lenders, Emmanouil Koukoumidis, Pei Zhang, and Margaret Martonosi. Location-based trust for mobile user-generated content: applications, challenges and implementations. In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 60–64, New York, NY, USA, 2008. ACM. 1, 2, 3, 56

[22] Memcached. `http://www.danga.com/memcached/`. 28

[23] OpenSSL. `http://www.openssl.org/`. 36

[24] Survey Information: Americans Care Deeply About Their Privacy. `http://www.cdt.org/privacy/guide/surveyinfo.php`. 6

[25] PyCrypto. `http://www.dlitz.net/software/pycrypto/`. 44

[26] Stefan Saroiu and Alec Wolman. Enabling new mobile applications with location proofs. In *HotMobile '09: Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, New York, NY, USA, 2009. ACM. 1, 2, 3, 9, 13, 57

[27] Naveen Sastry, Umesh Shankar, and David Wagner. Secure verification of location claims. In *WiSe '03: Proceedings of the 2nd ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2003. ACM. 55

[28] Anmol Sheth, Srinivasan Seshan, and David Wetherall. Geo-fencing: Confining Wi-Fi Coverage to Physical Boundaries. In Hideyuki Tokuda, Michael Beigl, Adrian Friday, A. J. Bernheim Brush, and Yoshito Tobe, editors, *Pervasive*, volume 5538 of *Lecture Notes in Computer Science*, pages 274–290. Springer, 2009. 57

[29] SOAP. `http://www.w3.org/TR/soap/`. 42

[30] SOAPpy. `http://pywebsvcs.sourceforge.net/`. 44

[31] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, Christina Pöpper, and Srdjan Čapkun. Attacks on Public WLAN-based Positioning. In *Proceedings of the ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2009. 57

[32] Tor. `http://www.torproject.org/`. 33

[33] Patrick Traynor, Joshua Schiffman, Thomas La Porta, Patrick McDaniel, Abhrajit Ghosh, and Farooq Anjum. Constructing Secure Localization Systems with Adjustable Granularity. Technical Report NAS-TR-0084-2007, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, 2007. 56

[34] UTM. `http://geology.isu.edu/geostac/Field_Exercise/topomaps/utm.htm`. 39

[35] Brent Waters and Edward Felten. Secure, Private Proofs of Location. Technical Report TR-667-03, Department of Computer Science, Princeton University, January 2003. 6, 55

[36] Yelp. http://www.yelp.com/. 2, 49