VeriPlace: A Privacy-Aware Location Proof Architecture

Wanying Luo & Urs Hengartner
Cheriton School of Computer Science
University of Waterloo
{w8luo, uhengart}@cs.uwaterloo.ca

ABSTRACT

Recently, there has been a dramatic increase in the number of location-based services, with services like Foursquare or Yelp having hundreds of thousands of users. A user's location is a crucial factor for enabling these services. Many services rely on users to correctly report their location. However, if there is an incentive, users might lie about their location. A location proof architecture enables users to collect proofs for being at a location and services to validate these proofs. It is essential that this proof collection and validation does not violate user privacy. We introduce VeriPlace, a location proof architecture with user privacy as a key design component. In addition, VeriPlace can detect cheating users who collect proofs for places where they are not located. We also present an implementation and a performance evaluation of VeriPlace and its integration with Yelp.

Keywords

Privacy, Location-based Services, Mobile Applications

Categories and Subject Descriptors

C.2.0 [**Security and protection (e.g., firewalls)**]

General Terms

Design, Experimentation, Measurement, Security

1. INTRODUCTION

Location-based services have seen lots of interest recently, both from companies and from academia. For instance, Foursquare [10] provides a location-based social networking platform through which users can "check in" at places that they are visiting and learn their friends' location. As of May 2010, Foursquare is averaging 700,000 checkins per day [1]. Yelp [30] allows users to post and read reviews about places. Recently, Yelp enabled a "check in" feature on its iPhone application so that readers can learn how often a reviewer visited the reviewed location. Other popular services are Gowalla [14] and Google Latitude [13]. From an academic point of view, researchers have studied location-based access control [7], which grants people access to a resource based on their geographical location (and maybe other factors).

Often, users can benefit from being at a given location. For example, Foursquare has the user who checks in the most often at a location become the mayor of the location. In turn, the owner of the location (e.g., a bar) might provide a reward to this user (e.g., a free drink). If there is a benefit for being at a location, people get an incentive for claiming that they are at a location even though they are not. For example, a Foursquare user might check in wrongly as being at a pub to get a free drink. In location-based access control, a doctor might pretend to be in the ward where a celebrity patient is located to get access to the patient's records. Foursquare is using GPS (and other un-reported measures) to verify a user's manually entered location [2]. The problems with this approach are possible manipulations of a smartphone's GPS-based location reporting mechanism (e.g., by jailbreaking the phone) and the limited coverage of GPS indoors or between high-rise buildings.

Location proofs are an alternative approach to let location-based services verify a user's location. Formally, a *location proof* is an electronic 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 obtain location proofs and with which services validate proofs. A crucial challenge is to ensure that users' privacy does not get violated while collecting and using location proofs. Moreover, we need to be able to detect users that try to fool the architecture by acquiring location proofs for locations where they are not located. Finally, the architecture must be applicable to a variety of location-based services and widely deployable. We present VeriPlace, a location proof architecture that comprehensively addresses these challenges, and make the following contributions:

- the design of a location proof architecture that preserves user privacy and that can detect cheating. Our architecture harnesses appropriate cryptographic techniques to achieve robust system security and user privacy protection. Furthermore, our architecture takes a first step towards defences against wormhole attacks [15] that do not rely on dedicated hardware;
- an implementation of the proposed architecture and an experimental performance evaluation;
- an integration of our architecture with Yelp to demonstrate the practicality of our architecture.

2. DESIGN CHALLENGES

In this section, we discuss several challenges that we take into account in the design of VeriPlace.

2.1 Privacy

Privacy is of central importance to mobile users. Namely, we must prevent issuers and verifiers of location proofs from violating a user's privacy.

Issuers of a location proof have knowledge of a user's location at the time the proof is issued to the user. If proof issuers can also learn the user's identity, they will be able to track users by their location. Therefore, shielding user identity from proof issuers is essential. However, it is difficult for proof issuers to vouch for a user's location if they do not know who they are vouching for in the first place.

Location-based services act as verifiers of a location proof. Users have to present a location proof to a service to access the service. If the proof discloses more granular location information than strictly necessary, the service can potentially infer sensitive personal information about a user, in particular, if the user's location is sensitive (e.g., a hospital). Different services have different requirements on the granularity of the required location. To make a single location proof usable to any service, a naïve solution is to include the most granular location information in the proof. While this solves the applicability problem, it violates a user's privacy if a service does not require the most granular location information. If a user knows what service she plans to use a location proof for, she can ask proof issuers to include location information of the desired granularity in the proof. However, the problem becomes difficult when the user needs a location proof for future use (see below).

2.2 Security

Mobile users may lie about their location to use services that they are not qualified for. A challenge is to incorporate a cheating detection mechanism into the location proof architecture so as to (1) make cheating difficult for users, (2) enable services to spot dishonest users who submit incorrect location information nonetheless, and (3) detect cheating without compromising user privacy (e.g., having a single party monitor a user's location proofs is undesirable).

Curbing user cheating is a challenging problem that lacks practical solutions in previous work. For example, the only guaranteed way to defeat wormhole attacks is to rely on dedicated hardware and distance-bounding [4], which makes it difficult to widely deploy a location proof architecture based on these techniques (see below). In a wormhole attack [15], a malicious party records network traffic in a region of the wireless network and replays it in another region. Assume Alice, who is in New York, wants to access a location-based service that is available only to people in San Francisco. Also assume that access points (APs) in Starbucks locations hand out location proofs. To get access, Alice asks her friend Bob in San Francisco to proxy communication between her and an AP in a San Francisco Starbucks.

2.3 Flexibility

A location proof must satisfy the requirements of the service that it is handed to. Embedding service-specific data (e.g., the service's public key [29]) in a location proof reduces the applicability of the proof because users would have to request a proof for each service that they want to interact with. The situation worsens as the number of services increases. Therefore, a challenge is to produce general location proofs that are acceptable by various services.

When a user requests a location proof, it is not always the case that the user has a target service at hand to interact with. Sometimes the user obtains proofs to save them for future use. For example, these proactively gathered proofs come in handy when the user later gets wrongly accused by his/her spouse, the police, or a fellow citizen of having been at a location. For example, Parking Mobility [21] provides an iPhone application that lets citizens take pictures of illegally parked cars and have them ticketed. Whereas such an application could often be useful, it also brings the potential for misuse. Alternatively, a user might be at a location without realizing that a proof for having been at this location could be useful later when interacting with a location-based service. For example, a user consumes spoilt food at a restaurant and suffers from food poisoning. With the help of the location proof proactively gathered while being at the restaurant, the user can later give more credibility to her (negative) review of the restaurant submitted to Yelp. In another example, a user might want to take advantage of the "missed connections" feature of Craigslist and posts the description of a person that she wants to see again. Here, the user could ask the person for a location proof to keep away unsolicited requests.

Without knowing what service a location proof will be used for, it is difficult for the user to judge what granularity of location information is appropriate. To make the proof useful both now and in the future, we need to include the most granular location information. However, this can violate a user's privacy, as discussed above.

2.4 Deployability

To be widely deployable, a location proof architecture must purely utilise the most common features of existing mobile devices or APs (assuming APs issue location proofs) and not require dedicated hardware. Requiring dedicated hardware is one of the factors that prevents previously proposed architectures (e.g., [29]) from being widely deployed. Although this requirement has benefits, such as defending against wormhole attacks, it is also a hurdle for regular users to use location proof technology.

The architecture must also be scalable as the number of users increases. Mobile devices and some proof issuers, such as APs, are weak compared to desktop computers in terms of processing power. Therefore, a location proof architecture should not require intensive computation on weak devices.

3. BENEFITS OF VERIPLACE

There are two primary benefits associated with VeriPlace: user privacy and cheating detection.

Protecting user privacy has been given paramount importance in our design. In particular, VeriPlace ensures that user identity is concealed from proof issuers and grants users fine-grained control over how much location information a location proof discloses to a service.

Our architecture is also capable of detecting cheating to a certain degree. Our cheating mechanism exploits the observation that a user cannot be at multiple locations at the same time. If a user gathers location proofs for her real location and also colludes with remote user(s) in a wormhole attack to obtain location proofs for other location(s) and if the time when these collusions happen is chronologically close, our architecture will detect the anomaly and notify the service when the dishonest user submits a location proof obtained by cheating.

As the popularity of location-based services rises, people will have to request location proofs increasingly often, and the effectiveness of our cheating detection mechanism will increase. Moreover, by guaranteeing people's anonymity from a proof issuer, VeriPlace gives users an incentive to gather location proofs frequently and on a proactive basis, in case a proof turns out to be useful later. Therefore, people's cheating attempts will more likely result in the acquisition of proofs for conflicting locations, which VeriPlace can detect. Therefore, VeriPlace can provide an effective solution against wormhole attacks.

VeriPlace does fail to detect cheating if cheating committed by a user happened in chronologically diverging periods. However, VeriPlace is not designed to offer perfect security, but to provide *more* security than what is currently available to existing location-based services, which are often *low-value*. The security mechanisms deployed by these services (if any at all) are weak and easy to circumvent. For example, by jailbreaking an iPhone, its GPS-based location reporting mechanism can be manipulated. Some services let a user manually enter her location. Services relying on source IP addresses for access control (e.g., a content-delivery service) can easily be fooled with a proxy. In these scenarios, VeriPlace provides additional security since VeriPlace is more difficult and more expensive to circumvent. For example, in a wormhole attack, an attacker must be physically next to an AP to relay packets, instead of simply running a proxy server. Nonetheless, for high-value location-based services, operators should always adopt additional security mechanisms, such as user authentication, and not rely only on VeriPlace, which can still serve as a complementary mechanism. Similarly, VeriPlace cannot resist targeted, carefully planned cheating, but it is a useful mechanism for gathering additional evidence to defend against/support wrongful/proper claims of having been at a location.

4. SYSTEM MODEL

VeriPlace targets services provided by third parties (such as applications in Apple's App store), not services offered by cellphone providers (maybe in collaboration with third parties). A cellphone provider has to know a user's location to route calls to the user, so trying to hide her location from the provider (and any collaborators) makes little sense.

VeriPlace lets an access point (AP) issue an *intermediate* location proof that certifies the user's presence nearby the AP. Later when the user wants to provide a service with a location proof, she must first present her intermediate proof to a TTPL (introduced next) to obtain a *final* proof.

VeriPlace requires three types of trusted entities that are run by different parties to avoid collusion. To protect users' privacy, each trusted entity knows either a user's identity or her location, but not both of them.

A *TTPL (Trusted Third Party for managing Location information)* is responsible for issuing final location proofs by creating a new proof that includes all the information contained in intermediate location proofs, with some information transformed to another representation. In particular, an intermediate proof contains the identity of an AP (i.e., $\mathrm{ID}_{\mathrm{AP}}$) and is digitally signed by the AP. A TTPL replaces the AP identity in an intermediate proof with the location of the AP (and takes some additional measures to protect location privacy, as explained later) and the AP's signature with its own signature to generate a final proof. Furthermore, a TTPL makes its mapping from AP identities to geographical locations publicly available. The disclosure of AP-to-location mapping serves two purposes: (1) Services can verify the accuracy of these mappings and decide which TTPLs to trust. (2) The CDA (introduced later in this section) can directly use a TTPL's mappings in its cheating detection procedure, thus dispensing with the overhead of maintaining an AP-to-location database itself.

A *TTPU (Trusted Third Party for managing User information)* is in charge of storing *encrypted* location information associated with users. More specifically, a TTPU stores triples of the form ($\mathrm{ID}_{\mathrm{user}}$, T, E), where T represents the time when user $\mathrm{ID}_{\mathrm{user}}$ requested a proof and E represents the encrypted identity of the AP that issued the proof. In short, each record shows when and where (to be more precise, encrypted "where") a particular user requested a location proof. When verifying a final location proof submitted by the user, a service submits $\mathrm{ID}_{\mathrm{user}}$ and T to the TTPU, which searches in its database for any records matching the same $\mathrm{ID}_{\mathrm{user}}$ and roughly the same T value (as specified by the service). The TTPU extracts the corresponding E values and submits them to the CDA for cheating detection.

A *CDA (Cheating Detection Authority)* carries out cheating detection. After receiving encrypted AP information from the TTPU, the CDA decrypts this information and checks whether any two APs 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.

We assume that a user's or an AP's identity (i.e., $\mathrm{ID}_{\mathrm{user}}$ and $\mathrm{ID}_{\mathrm{AP}}$, respectively) is a public key that is certified by a Certificate Authority, which can be identical to the TTPU. Users and APs will not give away the corresponding private key (e.g., users can be prevented from doing so by having this release also result in the release of personal information). Users store their public and private key on their personal mobile device (e.g., their smartphone). Note that 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 always be reliable. Saroiu and Wolman [23] propose several alternatives to achieve stronger identities, but none of them is foolproof. However, there are strong incentives for people not to give away their phone and to protect it, such as being reachable continuously and protecting personal information stored on the device. Also, as mentioned before, VeriPlace is designed as a security mechanism for low-value location-based services to provide more security than what is currently available to these services.

5. THREAT AND TRUST MODEL

We consider the following malicious parties in VeriPlace:

- **Dishonest users**. A dishonest user 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 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 benefits like money. For example, a dishonest user may save the generated data when interacting with an AP and give these data to a malicious intruder, who can use these data in location proofs later on.

- **Defecting APs**. A malicious or misconfigured AP may collude with dishonest users and issue them location proofs with fake information. For example, a malicious AP in New York, which is only authorised to issue proofs representing places in New York, may issue location proofs representing places in Chicago.

- **Malicious services**. A malicious service tries to use location proofs obtained from its users to get unauthorised access to other services.

- **Passive and active eavesdroppers**. An eavesdropper records and maybe modifies communication between users, proof issuers, and services.

Neither users nor APs are trusted by services, because dishonest users lie about their location and defecting APs collude with these users. A TTPL trusts some APs to issue location proofs to users of the AP (and to no other users) and not to become compromised. As a consequence of its trust, the TTPL will be willing to convert intermediate proofs issued by a trusted AP into final location proofs. It is up to a TTPL to decide which APs to trust. We envision that this decision is typically based on an organisational basis. For example, a TTPL might trust all APs run by AT&T, but not necessarily APs run by an individual. Note that an intermediate location proof, as issued by an AP, simply says that a user is nearby the AP, but the proof does not contain any location information, so we do not have to defend against APs inserting wrong location information.

We assume that the three kinds of TTPs are run by different organisations and that users trust the TTPs used by them not to collude with each other. Otherwise, users' privacy could get violated. In addition, we have the following TTP-specific trust assumptions:

A TTPL is trusted by services to replace AP identity in intermediate location proofs with correct location information when producing final proofs. Services also trust TTPLs to encrypt the correct AP identity from an intermediate location proof in the final location proof. Users trust a TTPL not to collude with services. Otherwise TTPLs and services can collectively track users' location. Several TTPLs can exist, each of which could be trusted by different services or users. For example, companies that provide a location information infrastructure, such as SimpleGeo [26], or that maintain APs distributed over a large geographical area, such as AT&T, could become TTPLs. When a user wants to convert an intermediate location proof into a final location proof, she should consult a service first to get a list of TTPLs trusted by that service and then choose a TTPL that she also trusts.

A TTPU is trusted by services to assist in cheating detection in the way specified by our protocol. Services also trust a TTPU not to collude with users to make their cheating undetected. Users trust a TTPU not to collude with services or APs, otherwise their location privacy could get violated. Ideally, to get the best cheating detection results, each user uses exactly one TTPU (though different users could use different TTPUs). Recall that a TTPU keeps track of a user's location proofs (without learning the user's location). If the user used multiple TTPUs, none of them would have a complete picture of the user's location proofs. Therefore, a service should specify which TTPUs that it trusts (e.g., Facebook could be such a TTPU). If a service specifies that it trusts only one TTPU, the service gives a strong incentive to its users to use only this TTPU since location proofs issued in collaboration with other TTPUs would be useless. Of course, a user can always get multiple proofs for being at a given location, where each proof is acquired with the help of a different TTPU. (This would be similar to check in at a location with both Foursquare and Gowalla.)

A CDA is trusted by services to conduct cheating detection and to report results honestly, as specified by our protocol. Similar to TTPUs, there can be multiple CDAs, but a service should indicate to its users which CDAs that it trusts. Note that it is possible for a single location proof to cover multiple CDAs by including multiple ciphertexts (one per CDA, each containing the user's encrypted location) in the location proof. Services have the most interest in the existence of CDAs, so they should provide resources necessary for funding organisations that run CDAs. Since the functionality performed by a CDA is straightforward (decrypting ciphertexts and computing geographical distances), these organisations do not need to be complex. In the extreme, such an organisation simply consists of a secure co-processor, such as the IBM 4765 [16], that is directly run by a TTPU. The co-processor's security properties guarantee that nobody, including the TTPU, is able to observe the co-processor's operations, not even with physical attacks.

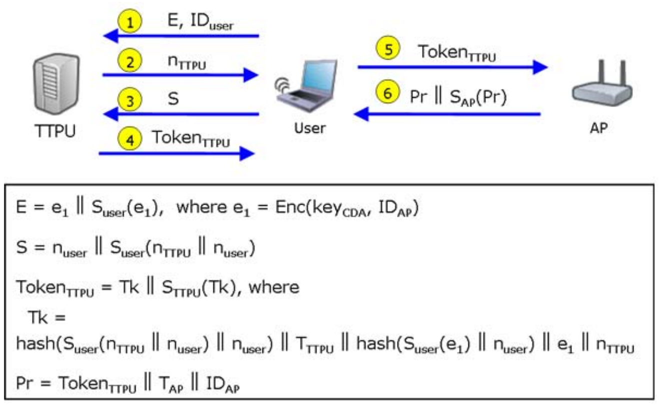
6. LOCATION PROOF ARCHITECTURE

We use $S_T(m)$ to represent a signature for message m generated with T's private key, and $\mathrm{Enc}(K, \text{data})$ to represent data encrypted with K's public key. The encryption scheme we will use here is probabilistic, that is, encrypting the same plaintext several times yields a different ciphertext each time. (Good cryptographic libraries, such as OpenSSL [20], provide this kind of encryption by default.) hash() denotes a cryptographic hash function, and $\|$ stands for concatenation. All nonces used by our protocols are of a fixed size. Finally, all communication is secured against passive and active eavesdroppers with the help of TLS/SSL.

6.1 Obtaining an Intermediate Location Proof

Before a user can request a final location proof from a TTPL, she must request an intermediate proof from an AP. The protocol to obtain an intermediate location proof is in figure 1. A user can proactively gather intermediate proofs or wait till she wants to access a particular location-based service. Users should change their device's MAC address to avoid becoming linkable to the AP. It might be possible to identify a device at the physical or link layer (e.g., [5]). However, these techniques are evaluated only with specific technologies, and countermeasures could be developed.

Briefly, in steps 1-4, the user retrieves a token from the TTPU that will be required for acquiring a location proof from the AP. The token is bound to the user and includes the (encrypted) identity of the AP. In steps 5-6, the user presents the token to the AP, which issues a location proof bound to the token and therefore to the user.



$E = e_1 \| S_{user}(e_1)$, where $e_1 = Enc(key_{CDA}, ID_{AP})$

$S = n_{user} \| S_{user}(n_{TTPU} \| n_{user})$

$Token_{TTPU} = Tk \| S_{TTPU}(Tk)$, where

$Tk =$
$hash(S_{user}(n_{TTPU} \| n_{user}) \| n_{user}) \| T_{TTPU} \| hash(S_{user}(e_1) \| n_{user}) \| e_1 \| n_{TTPU}$

$Pr = Token_{TTPU} \| T_{AP} \| ID_{AP}$

Figure 1: Request an intermediate location proof

1. The user sends $\mathrm{ID}_{\mathrm{user}}$ and $e_1 \parallel S_{\mathrm{user}}(e_1)$ to the TTPU, where $e_1 = \mathrm{Enc}(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}})$ is $\mathrm{ID}_{\mathrm{AP}}$ encrypted with the CDA's public key and $\mathrm{ID}_{\mathrm{AP}}$ is the identity of the AP from which the user requests an intermediate location proof. Our use of a probabilistic encryption scheme makes it impossible for the TTPU to brute force $\mathrm{ID}_{\mathrm{AP}}$ by encrypting each possible $\mathrm{ID}'_{\mathrm{AP}}$. In section 6.4, we show how a dishonest user who encrypts a wrong value for $\mathrm{ID}_{\mathrm{AP}}$ will be detected.

2. The TTPU verifies the user's signature and sends the user a random nonce n_{TTPU} to ensure freshness of the token to be issued.

3. The user sends $n_{\mathrm{user}} \parallel S_{\mathrm{user}}(n_{\mathrm{TTPU}} \parallel n_{\mathrm{user}})$ to the TTPU, where n_{user} is a random nonce generated by the user that ensures that the content signed by the user is not controlled entirely by the TTPU.

4. The TTPU verifies the user's signature and stores $(\mathrm{ID}_{\mathrm{user}},\ T_{\mathrm{TTPU}},\ e_1)$ in its database, where T_{TTPU} is the time when the TTPU receives the request from the user. It sends the following token to the user:

$$\mathrm{Token}_{\mathrm{TTPU}} = Tk \parallel S_{\mathrm{TTPU}}(Tk)$$

where $Tk = \mathrm{hash}(S_{\mathrm{user}}(n_{\mathrm{TTPU}} \parallel n_{\mathrm{user}}) \parallel n_{\mathrm{user}}) \parallel T_{\mathrm{TTPU}} \parallel$

$\mathrm{hash}(S_{\mathrm{user}}(e_1) \parallel n_{\mathrm{user}}) \parallel e_1 \parallel n_{\mathrm{TTPU}}$. Note that the token includes hashes of the user's signatures, not the actual signatures, since a signature might reveal its creator's identity. A hash acts as a commitment by the user to her signature. Including random n_{user} in the hash, which is unknown to the AP, ensures that the AP cannot learn anything from the commitment, so the user will remain anonymous to the AP.

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

6. The AP returns an intermediate location proof:

$$Pr \parallel S_{\mathrm{AP}}(Pr)$$

where $Pr = \mathrm{Token}_{\mathrm{TTPU}} \parallel T_{\mathrm{AP}} \parallel \mathrm{ID}_{\mathrm{AP}}$ and T_{AP} is the time when the AP receives the request.

6.2 Obtaining a Final Location Proof

Figures 2(a) and 2(b) show the protocols to obtain a final location proof. A user can delay this conversion till she actually needs to interact with a location-based service (at the risk of limited availability if the TTPL happens to be unaccessible at this moment in time), or she can proactively convert (some of her) intermediate proofs into final proofs (in particular, when she knows that she will require a particular final proof).

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

2. The TTPL retrieves $\mathrm{ID}_{\mathrm{AP}}$ from the proof. If it trusts this AP and if the proof signature is valid, the TTPL examines whether the proof contains a TTPU token. If it does not, the request is rejected. Otherwise, the TTPL replaces $\mathrm{ID}_{\mathrm{AP}}$ with L_g, which is actual location information of granularity level g. Moreover, the TTPL includes $\mathrm{ID}_{\mathrm{AP}}$ encrypted with the CDA's public key in the proof for cheating detection. Finally, the TTPL sends to the user a final location proof:

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

where $Pr' = \mathrm{Token}_{\mathrm{TTPU}} \parallel T_{\mathrm{AP}} \parallel e_2 \parallel L_g$ and $e_2 = \mathrm{Enc}(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}})$.

The above protocol applies to the case where the user has a desired service and hence location granularity in mind. But as we discussed in section 2, the user is unlikely to know what granularity to ask for when she requests a proof 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. In particular, in the first step, the user now sends the wildcard location granularity $g = *$ to the TTPL. In the second step, the TTPL generates a root key key_5, which is used to symmetrically encrypt the most granular location information, or $\mathrm{L}_5 = \mathrm{E}(\mathrm{key}_5\ ,\ \text{location of granularity level } 5)$. For granularity levels $g = 4, \ldots, 1$, the TTPL computes

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

In this case, the final location proof looks as follows:

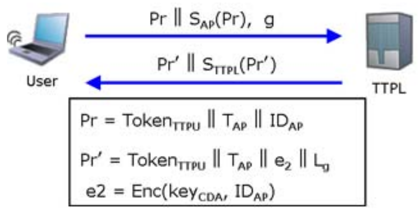
$$Pr' \parallel S_{\mathrm{TTPL}}(Pr')$$

where $Pr' = \mathrm{Token}_{\mathrm{TTPU}} \parallel T_{\mathrm{AP}} \parallel e_2 \parallel \mathrm{key}_1 \parallel \mathrm{L}_1 \parallel \ldots \parallel \mathrm{L}_5$ and $e_2 = \mathrm{Enc}(\mathrm{key}_{\mathrm{CDA}},\ \mathrm{ID}_{\mathrm{AP}})$. key_5 is also sent to the user.

Direct communication with the TTPL would reveal information about users, such as their IP address, so users should use tools such as Tor [28] for protecting their anonymity.

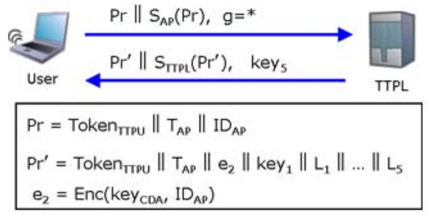
6.3 Verifying a Final Location Proof

The protocol to verify a final location proof is sketched in figure 3. The user submits n_{TTPU}, n_{user}, $S_{\mathrm{user}}(n_{\mathrm{TTPU}} \parallel n_{\mathrm{user}})$, $S_{\mathrm{user}}(e_1)$, $\mathrm{ID}_{\mathrm{user}}$, and $Pr' \parallel S_{\mathrm{TTPL}}(Pr')$ to the location-based service that she wants to access. The service 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 service verifies the location proof as follows:



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)

(a) Retroactive request



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)

(b) Proactive request

Figure 2: Request a final location proof

1. The service first verifies that the two signatures submitted by the user were issued by the user and correspond to the signatures committed to in $\text{Token}_{\text{TTPU}}$.

2. If so, the service verifies whether T_{AP} (extracted from Pr') and T_{TTPU} (extracted from $\text{Token}_{\text{TTPU}}$) represent approximately the same time.

3. If so, the service sends ID_{user}, T_{TTPU}, e_1 and e_2 to the TTPU for cheating detection and notifies the TTPU which TTPL issued the final location proof.

4. With T_{TTPU} and ID_{user}, the TTPU is able to pull out all the records of the user with approximate T_{TTPU} value (as specified by the service) from its database. The TTPU then extracts from all these records the encrypted identities of APs that issued location proofs to the user around time T_{TTPU} and sends 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 final 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 reports cheating. If all the ciphertexts are decrypted to valid AP identities, the CDA compares whether $\text{ID}_{\text{AP}_1} = \text{ID}_{\text{AP}_2}$, where ID_{AP_1} is decrypted from e_1 and ID_{AP_2} is decrypted from e_2. If these two decrypted AP identities are not identical, the CDA informs that there is cheating. Otherwise, using the TTPL's AP-to-location database, the CDA examines whether any two ID_{AP}'s are far apart (as specified by the service), which indicates cheating, and informs the TTPU, which in turn tells the service.

If all the steps above are passed successfully and no cheating is detected, the service accepts the proof. If the proof is not proactively gathered, the service can use L_g in the proof directly. However, if the user requested the proof proactively, the user needs to decide about a desired granularity g and send the service 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 compute $\text{key}_3 = \text{hash}\left(\text{hash}\left(\text{key}_5\right)\right)$ and send key_3 to the service. On receiving the decryption key, the service must make sure the user submitted a proper key as follows: for $i = g, \ldots, 1$, $\text{key}_i = \text{hash}\left(\text{key}_{i+1}\right)$. If the computed key_1 matches the key_1 in the location proof, the service can be certain the decryption key submitted by the user is valid.

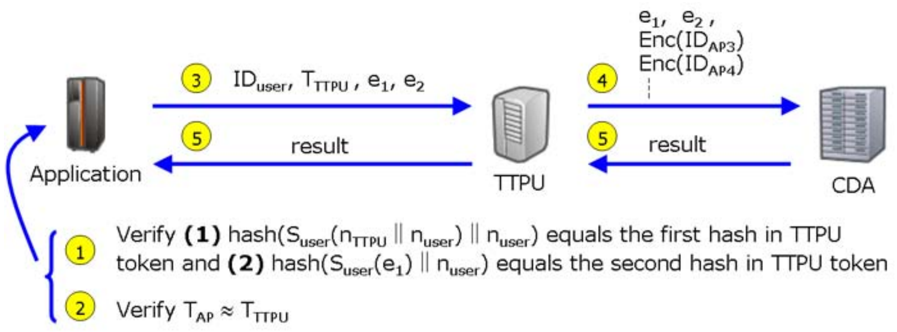
6.4 Security Analysis

VeriPlace defeats collusion between dishonest users and defecting APs. Assuming the information in the TTPU token is correct, if users and APs collude to produce intermediate location proofs with fake information, it will be detected during the verification phase, because the information that an AP places in an intermediate proof can be verified by comparing the corresponding fields in the TTPU token. For example, an incorrect T_{AP} or ID_{AP} will be detected in step 2 and 5 of the verification phase, respectively.

However, the TTPU token may contain incorrect information, since a dishonest user may cheat by submitting a wrong ID_{AP} when registering with the TTPU. For example, a dishonest user may submit $\text{Enc}\left(\text{key}_{\text{CDA}},\ \text{ID}_{\text{AP}_1}\right)$ to the TTPU to obtain a TTPU token, while she is actually attempting to request a location proof from another AP with identity ID_{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 ID_{AP_2} in the intermediate proof. In this case, the dishonest user has no way of preventing the TTPL from copying the correct ID_{AP_2} included by AP_2 to the final location proof, which allows the service to detect the cheating in step 5 of the verification phase. If AP_2 is defecting and colludes with the user and puts the incorrect identity ID_{AP_1} in the intermediate proof, it is not able to sign the proof with AP_1's private key, and the TTPL will refuse to create a final proof.

A malicious intruder cannot acquire a TTPU token and then give it to a dishonest user since each token contains a commitment to a signature, where this signature was created by the entity to whom the token was issued (the malicious intruder in our case). Due to the security properties of a cryptographic hash function, it is not possible for the dishonest user to create a signature that is covered by the same commitment. Similarly, since tokens are included in location proofs, a malicious service cannot re-use a location proof issued to somebody else for its own purposes.

A dishonest user might cheat by acquiring a TTPU token that lists a given AP while not actually being close to the AP and by getting a location proof from the AP only once she is nearby the AP. However, a token lists the time when the user is expected to send a proof request to an AP, and a location proof lists its issuing time. For a service to validate a proof, these two timestamps have to be close. Replay attacks take place when a dishonest user reuses the same TTPU token for requesting more than one location proof. If she reuses the same token with the same AP within the time-frame allowed by a service, this will not be detected, but it is not a problem. If she reuses the token with a different AP, the encrypted AP identity contained in the token will not be correct, which will be detected, as mentioned above.



3 ID_user, T_TTPU, e1, e2
5 result
4 e1, e2, Enc(ID_AP3) Enc(ID_AP4)
5 result
Application
TTPU
CDA
1 Verify (1) hash(S_user(n_TTPU || n_user) || n_user) equals the first hash in TTPU token and (2) hash(S_user(e1) || n_user) equals the second hash in TTPU token
2 Verify T_AP ≈ T_TTPU

Figure 3: Verification of a location proof

Services and the TTPU should be prevented from knowing ID_{AP} (i.e. the identity of the AP that issued location proofs to the user), since knowledge of ID_{AP} will allow them to deduce the user's accurate location by investigating the AP's geographical location. Instead, both the service and the TTPU are given the encrypted ID_{AP} when the user requests a token or presents a final location proof, respectively. Services and the TTPU may use a brute-force attack to try to learn ID_{AP}: they encrypt identities of all the APs in the world using the CDA's public key and compare ciphertexts However, our use of probabilistic encryption defends against this attack.

Communication secured with TLS/SSL protects against passive and active eavesdroppers.

7. IMPLEMENTATION

In this section, we elaborate on our wireless testbed and the implementation of the three trusted third parties. In our implementation, we choose RSA with a key length of 2048 bits and AES with a key length of 256 bits for all public-key and symmetric-key encryption, respectively. We use SHA-256 for all hashing operations.

To deploy VeriPlace, we adopt an existing Wi-Fi research testbed [3]. A total of 38 AP nodes, all connected to a central controller, are deployed on the second and third floors of the computer science building at our university. The testbed consists of two major components: 1) Wi-Fi APs, which consist of a VIA EPIA EN12000EG mainboard with a 1.2 GHz C7 nanoBGA2 processor. 2) A central controller, which is a desktop computer with a dual core 2.66 GHz processor and a Gigabit connection with the APs. The controller runs Ubuntu 7.04 with the 2.6.20-15 Linux kernel.

In our testbed, we have the controller issue location proofs on behalf of APs. In an organisation, APs are often managed by a central party, and it makes sense to also have this party issue location proofs. Since this entity is used for security-critical tasks (e.g., access control to the organisation's Wi-Fi), the organisation has lots of interest in protecting it against compromise. A user who wants to request a location proof from the central controller must first associate with one of the APs. After the association (and authentication) is complete, the AP sends an association message to the central controller. The user can then ask the controller for a location proof, and the AP will route protocol messages to and from the 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.

The TTPL and TTPU are implemented as Web servers, so that users can use their web browser to interact with them. For easier integration with location-based services and client applications accessing these services, we also implemented SOAP server versions of these two services. Both services also manage a MySQL database to store user information and AP's location and public keys, respectively. We implemented the CDA as a SOAP server and use TLS/SSL to secure communication.

All three trusted third parties are currently hosted on a server at our university to simplify experimentation. The 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. The three trusted third parties are implemented in Python. We use the SOAPpy library [27] to implement the SOAP services. We adopt the PyCrypto [22] toolkit to implement the cryptographic algorithms.

8. EXPERIMENTS

APs are typically weak devices that could potentially throttle the performance of VeriPlace. Therefore, we provide experimental performance results in this section. We first measure the throughput of the controller and then pinpoint the performance bottleneck.

We do not present TTPL and TTPU-related performance measurements, because the results of such measurements would have little value. In practice, neither a TTPU nor a TTPL have to be run on a single server. Instead, they could be deployed on multiple servers to ensure scalability (e.g., a TTPU assigns a user to one out of multiple servers based on ID_{user}). In general, the performance of the TTPL and TTPU is less of a concern, since we expect them to be more powerful compared to APs.

8.1 Controller Throughput

The controller is the master of all APs and responsible for issuing location proofs. Therefore, it is the potential performance bottleneck of VeriPlace. We conduct 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 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 them from the TTPU in real time. Another factor that could potentially skew the results 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.

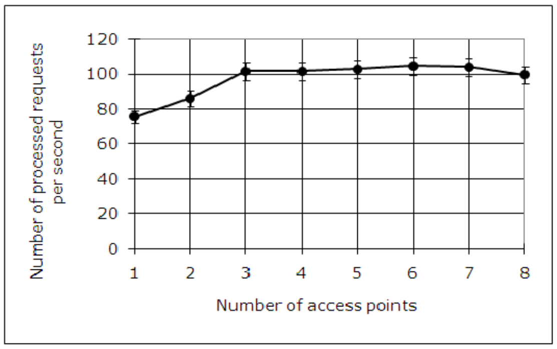


Figure 4: Controller throughput

We conduct eight rounds of experiments with an increasing number of APs. For every round, 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. As shown in the figure, the performance of the controller reaches its peak after we run three APs. Moreover, during the experiment we monitored the CPU usage, which was at 100%. Therefore, we can be certain that the network is not the bottleneck.

Our controller can issue about 100 proofs per second. Assuming users get a location proof once every half an hour, our setup can support up to 180,000 users, which is far above the number of people in our building (and on our campus).

8.2 Micro Benchmarks

Now that we have determined that the controller is the bottleneck, we need to find out which part of our protocol significantly affects performance. Generally speaking, creating cryptographic signatures is an expensive operation and is the most likely culprit. To confirm this expectation, we conduct the following experiment:

We divide the controller's processing of a request for a location proof 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 bit of the location proof and the last bit of the proof 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. Figure 5 summarises the experimental results. The most time-consuming part is signature creation, which is 23 times more expensive than the other two parts combined. In terms of the numerical value of the cost, the three parts take 0.327 ± 0.003 ms (for part 1), 0.387 ± 0.004 ms (for part 2) and 15.02 ± 0.09 ms (for part 3).

Since we rely on PyCrypto's implementation of RSA for creating signatures, the question arises whether our library choice is appropriate, given that interpreted languages like Python are often considered to be significantly slower than compiled languages such as C. We measured the PyCrypto implementation of RSA signing and compared it to the popular OpenSSL implementation [20]. Rather unexpectedly, we find that PyCrypto performs better (on the controller) or the same (on the server hosted at our university) as OpenSSL. The reason is that PyCrypto implements its speed-critical operations in C and relies on a high-performance library for fast arbitrary precision arithmetic.

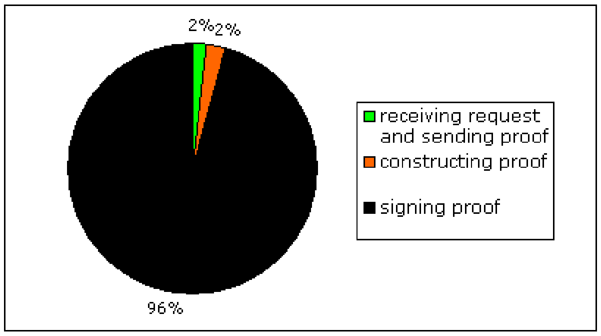


Figure 5: Cost of proof generation

9. REAL-WORLD SERVICES

As discussed in section 3, VeriPlace is suitable for two kinds of services: (1) services that are low-value and where executing a targeted attack would be more expensive; (2) services that currently have a weak location check and where VeriPlace gives better security. We built three real-world services based on VeriPlace that fit this profile: a service for instructors to take class attendance, a browser extension that adds location proofs to emails sent with Gmail, and a browser extension that adds location proofs to Yelp. We now present the third service. We first present the location proof daemon, which is a client-side program that runs on a user's mobile device (a Linux laptop in our implementation) and that automates the task of requesting location proofs.

9.1 Location Proof Daemon

The location proof daemon reads a local configuration file on startup, where parameters, such as the address of the TTPL and TTPU, desired location granularity, how often to request a location proof, etc. are specified. The daemon sends location proof requests periodically and saves location proofs to the mobile device. Moreover, the daemon allows the user to dynamically change the parameters through a frontend command shell. The user can also instruct the daemon to send a location proof request at once.

9.2 Location Proofs For Yelp Web Review

Yelp [30] is a social networking and user review website with over 25 million visits per month. Users can provide ratings and reviews for various places. In early 2010, Yelp released a collection of location features for its iPhone application that allows users' reviews to be supported by their location. Namely, a Yelp user can "check in" through Yelp's iPhone application. Alongside a user's review, Yelp will then display how often the user checked in at the reviewed location. This feature can increase the credibility of reviews.

VeriPlace is a good fit for making Yelp more secure. Currently, by jailbreaking an iPhone, an attacker could modify the Yelp application and have it provide wrong check-in locations. Instead, VeriPlace requires users to prove their physical presence at the claimed location. Since we cannot

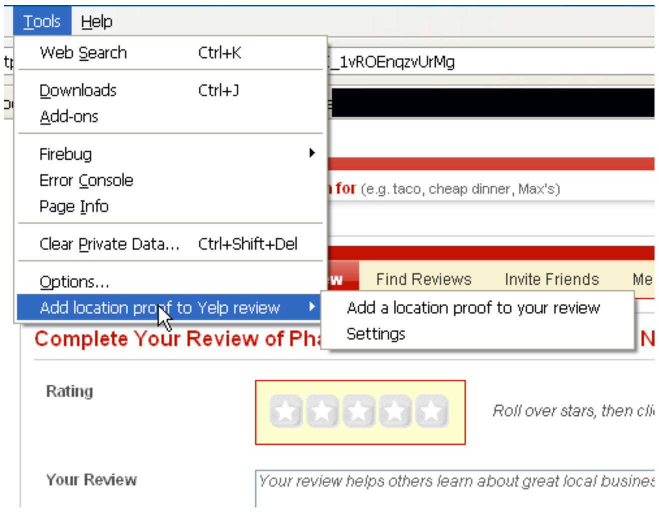
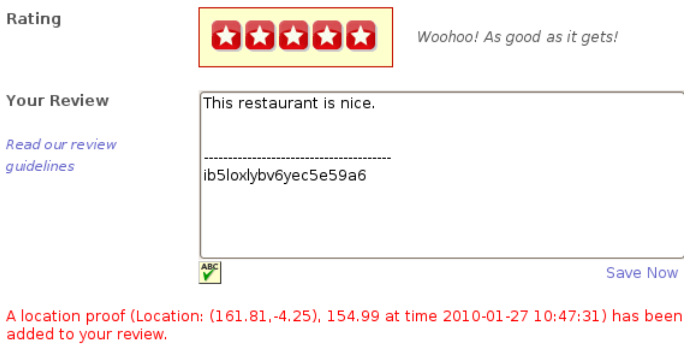

Tools
Help
Web Search Ctrl+K
Downloads Ctrl+J
Add-ons
Firebug
Error Console
Page Info
Clear Private Data... Ctrl+Shift+Del
Options...
Add location proof to Yelp review
Add a location proof to your review
Settings
Find Reviews
Invite Friends
Complete Your Review of Pha
Rating
Roll over stars, then cli
Your Review
Your review helps others learn about great local busines

Figure 6: Attach a location proof to a Yelp review



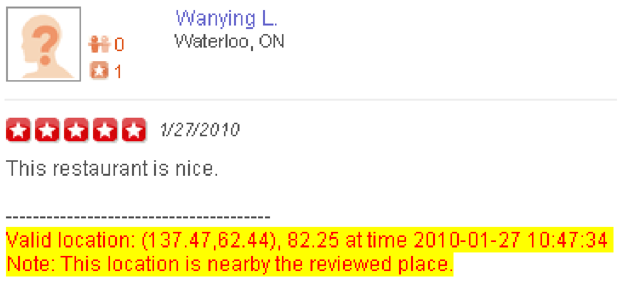
This restaurant is nice.

----

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.

Figure 7: Notify user of a location proof



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.

Figure 8: Yelp page with a location proof

modify the Yelp service directly, we developed a Firefox extension that lets reviewers attach a location proof to their review and that retrieves location proofs for the readers of a review. More specifically, a user composes a review on the Yelp website as usual. Before submitting the review, she attaches a location proof, retrieved from the location proof daemon, to her review using the extension as shown in figures 6 and 7. The extension generates a random review identifier and adds it to the review. Then it submits the location proof, the Yelp URL of the reviewed place, and the review identifier to a server currently hosted by us. The server acts as a service in our location proof architecture. It first verifies the location proof. Then, using the URL it retrieves the address of the reviewed place and determines its longitude/latitude using Google's geocoding service. If this location is indeed nearby the location extracted from the location proof, the server stores the extracted location information along with the corresponding review identifier in its database. Otherwise, the server stores the extracted location information with an appended string that warns that the location proof represents a non-nearby location (therefore, the user's review is questionable). The user then submits her review to Yelp. Later when a user tries to view a review, assuming that she has the extension installed, the extension will use the review identifier contained in the review to automatically retrieve from our server location information associated with the review and display it along with the review, as shown in figure 8.

10. RELATED WORK

There are several previous approaches for location proof architectures that rely on measuring the round trip time of an electromagnetic signal to bound the distance between two parties (e.g., [4, 6, 29], see [12] for a survey). A general problem with these approaches is that they are based on measurements of very high precision, which requires dedicated hardware and hinders deployability. For example, Waters and Felten [29] introduce a system that allows a device to obtain location proofs from a location manager (LM) and submit proofs to a verifier. A device requests location proofs by sending its device ID encrypted with the public key of the verifier. The LM then sends the device a nonce which the device immediately sends back. The LM measures the round-trip delay and sends the device a location proof containing the measured delay and the encrypted device ID. A drawback of this system is that a malicious intruder can craft fake device IDs and collect location proofs on behalf of dishonest users, which is not possible in VeriPlace. Moreover, the system requires the device to know verifiers in advance and does not support proactively gathered location proofs.

An approach that does not require dedicated hardware is to have APs or GPS satellites broadcast a token. Only devices that are close to the AP/satellite will see this token and can hand it over as a location proof to a service [8, 9, 25]. For example, Faria and Cheriton [9] design a location-based authentication architecture for wireless LANs. A centralised wireless appliance (WA) controls a group of APs and broadcasts a set of random nonces through its controlled APs. To prove its closeness, a client must capture and send all the received nonces back to the WA. Denning and MacDoran [8] present a location-based authentication system where a location signature sensor (LSS) creates location signatures from GPS signals that describe the physical location of the LSS at a particular time. A user carrying an LSS can hand a location signature to a service, which learns the user's location based on the location signature. Broadcast-based approaches lack a strong binding between the location proof and the user identity, so proofs can be easily given away. In comparison, VeriPlace ties location proofs to specific users.

Lenders et al. [18] describe a geotagging service that allows a content creator to obtain a location/time certificate for the content. Such a certificate proves the generation location and time of the content, but, as opposed to VeriPlace, it does not bind the content to its generator. In addition, their system relies on a single trusted location/time verification party, whereas VeriPlace distributes trust among multiple parties run by different organisations.

Kirkpatrick and Bertino [17] have dedicated location devices (LDs) issue location proofs based on Near-Field Communication. A user's device sends a proof to a Resource Manager, which forwards it to a Role Manager for validation. Separating between these two parties prevents them from learning a user's identity and location and the requested resource, respectively. This system is designed to provide only pseudonymity. Because the mapping between a user's identity and her device likely remains constant, LDs could infer it over time and track the user. VeriPlace provides anonymity, since APs learn neither a user's identity nor a pseudonym associated with her. Finally, the system does not deal with cheating users, as opposed to VeriPlace.

Saroiu and Wolman [23] also have APs issue location proofs. A user extracts a sequence number from a beacon broadcast by an AP and signs and returns the number to the AP, which issues a location proof. This design is problematic from a privacy point of view. First, it reveals a user's identity to the AP, which makes the proactive collection of location proofs impossible due to privacy concerns. In VeriPlace, privacy protection is fundamental, which enables proactively gathered location proofs. Second, their system always reveals a user's fine-grained location to a service, which can violate a user's location privacy. Finally, the system does not deal with cheating users, as opposed to VeriPlace.

Gilbert et al. [11] and Saroiu and Wolman [24] introduce mechanisms based on dedicated hardware (namely, a Trusted Platform Module (TPM)) to ensure the integrity of a sensing device, such as a smartphone. The assumption of dedicated hardware violates one of our assumptions; for example, consumer laptops (like netbooks) often do not come with a TPM. In addition, whereas their mechanisms ensure that a sensor cannot be manipulated, it does not help if the sensed signal (GPS in case of location) is manipulated. The approach also suffers from the disadvantages of broadcast-based solutions mentioned above.

We presented an early version of our architecture in a workshop paper [19]. The early version has no cheating detection capabilities, and there is no implementation and evaluation of the architecture.

11. CONCLUDING REMARKS

We have identified four challenges in designing a location proof architecture and addressed them in VeriPlace. In particular, we illustrated how cryptographic techniques can aid in preserving user privacy and protecting system security. Furthermore, our work takes a first step towards defending against wormhole attacks without relying on dedicated hardware. Developing additional software-based defences against wormhole attacks demands future research. Finally, we note that sometimes users lie about their location not to access a service that they are not entitled to, but to protect their privacy. Another topic of future research is studying whether and how this kind of lying should be supported in a location proof architecture.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work is supported by a Google Research Award and by the Natural Sciences and Engineering Research Council of Canada.

12. REFERENCES

[1] http://mashable.com/2010/05/28/foursquare-checkins/. Accessed June 2010.

[2] http://blog.foursquare.com/post/503822143/on-foursquare-cheating-and-claiming-mayorships-from. Accessed June 2010.

[3] N. Ahmed and U. Ismail. Designing a High Performance WLAN Testbed for Centralized Control. In *Proc. TridentCom 2009*, pages 1–6.

[4] S. Brands and D. Chaum. Distance-Bounding Protocols. In *Proc. EUROCRYPT '93*, pages 344–359.

[5] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless Device Identification with Radiometric Signatures. In *Proc. MobiCom 2008*, pages 116–127.

[6] S. Căpkun, L. Buttyán, and J.-P. Hubaux. SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks. In *Proc. SASN 2003*, pages 21–32.

[7] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. GEO-RBAC: A Spatially Aware RBAC. *TISSEC*, 10(1), 2007.

[8] D. E. Denning and P. F. MacDoran. Location-Based Authentication: Grounding Cyberspace for Better Security. pages 167–174, 1998.

[9] D. Faria and D. Cheriton. No Long-term Secrets: Location Based Security in Overprovisioned Wireless LANs. In *Proc. HotNets-III*, 2004.

[10] Foursquare. http://foursquare.com/.

[11] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall. Toward Trustworthy Mobile Sensing. In *Proc. HotMobile 2010*, pages 31–36.

[12] A. I. González-Tablas Ferreres, B. R. Álvarez, and A. R. Garnacho. Guaranteeing the Authenticity of Location Information. *IEEE Pervasive Computing*, pages 72–80, Jul-Sept 2008.

[13] Google Latitude. http://www.google.com/latitude/.

[14] Gowalla. http://gowalla.com/.

[15] Y. C. Hu, A. Perrig, and D. B. Johnson. Packet Leashes: a Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proc. INFOCOM 2003*, pages 1976–1986.

[16] IBM. IBM PCIe Cryptographic Coprocessor. http://www-03.ibm.com/security/cryptocards/pciecc/overview.shtml.

[17] M. S. Kirkpatrick and E. Bertino. Enforcing Spatial Constraints for Mobile RBAC Systems. In *Proc. SACMAT 2010*, pages 99–108.

[18] V. Lenders, E. Koukoumidis, P. Zhang, and M. Martonosi. Location-based Trust for Mobile User-generated Content: Applications, Challenges and Implementations. In *Proc. HotMobile '08*, pages 60–64.

[19] W. Luo and U. Hengartner. Proving Your Location Without Giving up Your Privacy. In *Proc. HotMobile 2010*, pages 7–12.

[20] OpenSSL. http://www.openssl.org/.

[21] Parking Mobility. http://www.parkingmobility.com/.

[22] PyCrypto. http://www.dlitz.net/software/pycrypto/.

[23] S. Saroiu and A. Wolman. Enabling New Mobile Applications with Location Proofs. In *Proc. HotMobile '09*, pages 1–6.

[24] S. Saroiu and A. Wolman. I Am a Sensor, and I Approve This Message. In *Proc. HotMobile 2010*, pages 37–42.

[25] A. Sheth, S. Seshan, and D. Wetherall. Geo-fencing: Confining Wi-Fi Coverage to Physical Boundaries. In *Proc. Pervasive 2009*, pages 274–290.

[26] SimpleGeo. http://simplegeo.com/.

[27] SOAPpy. http://pywebsvcs.sourceforge.net/.

[28] Tor. http://www.torproject.org/.

[29] B. Waters and E. Felten. Secure, Private Proofs of Location. Technical Report TR-667-03, Department of Computer Science, Princeton University, January 2003.

[30] Yelp. http://www.yelp.com/.