# COMBINING SOCIAL AUTHENTICATION AND UNTRUSTED CLOUDS FOR PRIVATE LOCATION SHARING

by

**Andrew K. Adams**

B.S. in Information Science, University of Pittsburgh, 1989

M.S. in Information Science, University of Pittsburgh, 1991

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

**Master of Science**

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This thesis was presented

by

Andrew K. Adams

It was defended on

November 20, 2015

and approved by

Adam J. Lee, Associate Professor, University of Pittsburgh

Daniel Mossé, Professor, University of Pittsburgh

Taieb Znati, Professor, University of Pittsburgh

Thesis Advisor: Adam J. Lee, Associate Professor, University of Pittsburgh

**COMBINING SOCIAL AUTHENTICATION AND UNTRUSTED CLOUDS FOR PRIVATE LOCATION SHARING**

Andrew K. Adams, M.S.

University of Pittsburgh, 2015

With the advent of GPS-enabled smartphones, location-sharing services (LSSs) have emerged that share data collected through those mobile devices. However, research has shown that many users are uncomfortable with LSS operators managing their location histories, and that the ease with which contextual data can be shared with unintended audiences can lead to regrets that sometimes outweigh the benefits of these systems. In an effort to address these issues, we have developed SLS: a secure location sharing system that combines location-limited channels, multi-channel key establishment, and untrusted cloud storage to hide user locations from LSS operators while also limiting unintended audience sharing. In addition to describing the key agreement and location-sharing protocols used by the architecture, we discuss an iOS implementation of SLS that enables location sharing at tunable granularity through an intuitive policy interface on the user's mobile device.

**Keywords:** Key Management; Location Tracking; Presence Systems; Privacy; Security.

**TABLE OF CONTENTS**

## LIST OF FIGURES

# 1.0 INTRODUCTION

Over the last decade, location- and presence-sharing systems have received considerable attention from both researchers [6, 8, 21, 22, 28, 29, 35] and in practice [12, 15, 18, 20]. The recent explosion in mobile computing and social networking has led to deployment of a wide range of location-sharing systems (LSSs), both stand-alone in nature (e.g., Find My Friends [14], FourSquare [15], or Glympse [18]) as well as integrated with other social networking platforms (e.g., Facebook places [12], Google Plus [20], Twitter, or Yelp). These types of systems allow a user to share her geographic location with her social contacts either as a first-class data object or as support for other content (e.g., attaching one's location to restaurant review). This sharing can be done in a near seamless manner, particularly when the LSS is embedded within a larger social platform.

Despite their popularity, LSSs are not without their own security and privacy problems. By their very design, these systems have the implicit shortcoming that sharing one's location with social contacts *requires* sharing this location with the LSS operator as well. This can lead to undesirable profiling of users by third parties, or increase users' exposure risk in the event of an LSS compromise. In addition, it is has been shown that social networks in general [36] and LSSs in particular [29] can sometimes lead to situations in which users experience regrets after (over)sharing information. This is often the result of the so-called *unintended audience* problem, in which data is shared with individuals other than those with whom the subject intended to share. This may manifest as a result of a location being automatically attached to content posted on a social network, accidental sharing of a location with a user's entire set of contacts instead of a restricted subset, or posting a location that contradicts other statements made by the user [29].

The latter problem is symptomatic of both LSS and access control complexity. For instance, it is well-known that users' social networks have many more contacts than they interact with on a day-to-day basis: a 2011 poll of 1,954 British citizens found that the average person had 476

Facebook friends, but only 152 contacts in their cellular phone [38]. Furthermore, research studies have shown that users frequently make mistakes when authoring even basic access control policies in commodity systems [5,11,30]. As such, it is clear that accidents and misconfigurations can lead to over-sharing in large social networks. On the other hand, the problem of required sharing with LSS operators is one of economic incentives: the ability to study user habits and carry out targeted advertising provides revenue for operators of the systems.

An interesting observation, however, is that current generation smartphones are capable of helping mitigate both of the above types of concerns. Given the 3G/4G connectivity of these devices and the open APIs to cloud storage-as-a-service (SaaS) providers like Amazon S3 [2] and Google Drive [19], it is possible for mobile applications to explicitly manage a user's published location history. Furthermore, smartphones store rich information about a users' close contacts (e.g., email addresses, phone numbers) and have access to multiple channels of communication (e.g., WiFi, 3G/4G, Bluetooth, SMS). As a result, it is possible to develop robust key exchange protocols—e.g., based on multiple distinct avenues of communication and historical context, or by leveraging location-limited channels—that allow location data to be selectively encrypted prior to upload, thereby preventing snooping attack by the SaaS provider and limiting incidences of over-sharing.

In this thesis, we describe *Secure Location Sharing* (SLS), a decentralized LSS that leverages the above observations to limit the over-exposure of user location data without relying on trusted infrastructure. Specifically, SLS allows users to set up secure location sharing with selected contacts by pairing devices in one of two ways. Users who happen to be physically co-located can use location-limited visual channels to pair devices (similar to [24]). Users who are located apart from one another can instead leverage multiple communication channels (e.g., email and SMS) along with contextual question/answer protocols to help prevent man-in-the-middle (MitM) attacks during device pairing. Cryptographic keys established during this pairing process are then used to aid in securely sharing a user's location at a tunable granularity (e.g., GPS coordinate, city-level, etc.) via untrusted SaaS services. In exploring SLS, we make the following contributions:

1. We demonstrate the first decentralized LSS that is capable of providing flexible and secure location sharing over untrusted infrastructure. Unlike existing approaches to securing social networks (e.g., X-pire! [3]), our work does not involve abuse of existing social network APIs,

but rather builds secure and flexible sharing into the real-life social networks managed by users' smartphones.

2. We propose an alternate economic model for LSSs, in which the users providing their location to others pay for the storage used to host their data.[1] This removes the economic incentives driving traditional LSS providers to view user location histories, and further reduces the risk of accidental over-exposure due to LSS compromise.
3. SLS limits the unintended audience sharing problem by requiring explicit device pairing between providers and consumers of location sharing. By leveraging multiple channels and/or location limited pairing protocols, this setup procedure is robust against even very strong adversaries with control over large portions of the network environment.
4. We develop an iOS application as a proof-of-concept implementation of SLS. This demonstrates both the efficiency of my approach, as well as the simplicity of interfaces needed to manage the secure device pairing aspects of SLS.

The rest of this thesis is organized as follows. In Chapter 2 we discuss related work, and briefly explain the problems associated with canonical location-sharing or presence system. We discuss the goals, properties and principals of our system in Chapter 3. In Chapter 4 and 5, we present our framework and implementation for secure location sharing. Chapter 6 re-examines the design, evaluates the performance and security of the system, and explores directions for future work. We present our conclusions in Chapter 7.

[1] Note, however, that some SaaS providers provide lower-tier service that is sufficient for SLS at little to no cost to the user (e.g., http://aws.amazon.com/free/).

# 2.0 RELATED WORK

## 2.1 PRIOR WORK

Google Plus [20], FourSquare [15], Facebook Places [12], and Glympse [18] are examples of LSSs that operate by having users (i.e., providers) upload their location data to the service, such that others (i.e., consumers) can access the location data. Current strategies for addressing privacy issues in LSSs are typically based on obfuscating the location data or anonymizing the provider; the efficiency of these techniques are discussed in, e.g., [33, 34]. In [31], the authors address oversharing in LSSs by providing users with interactive feedback about the number of users accessing their location, and the frequency of these accesses. This thesis deviates from prior work by (i) preventing the LSS from viewing a user's location data, and (ii) by ensuring that a user has full control over *whom* she chooses to share her location data with and *how* she intends for her location to be consumed.

The protection and secrecy of a user's data contained in the cloud is the focus of DataLocker [9], which is a collection of tools that enable a provider to encrypt data prior to uploading the data to the cloud. Our model does this precisely with location data, however, it is not tied to any specific cloud entity. Moreover, we do not generically encrypt location data: policy dictates the precision with which data is presented to the consumers, and how it is protected in the cloud. Instead of protecting data, X-Pire! [3] attempts to *decay* data (ostensibly images, but the technique could be applied to location data) by associating a key to an image; when the key expires the X-pire! aware server refuses to serve the data. Similarly, Vanish [16] decays data by altering links to the data stored within a DHT. Although our model does not address the decaying of data, it could benefit from techniques like these in the future.

Several papers present advanced key management protocols that make use of smart phone technology [7, 13, 23, 24]. McCune et al. describe the protocol, *Seeing-is-Believing (SiB)* [24], in which the camera in users' smart phones capture 2D barcodes—these 2D barcodes are used as commitments for exchanging public keys. Our key management protocol relies heavily on the concepts and ideas introduced in this work. SafeSlinger [13] is a protocol and framework designed to exchange public keys between smart phones; this is precisely one of the tasks that our key management protocol is designed to accomplish. Similarly, Open Key Chain [27] is an Android app used to exchange GPG keys, that uses QR codes to exchange commitment hashes. Our work diverges from both [13] and [27] by (i) using the *location-limited* channel between pairing smart phones to *fully* exchange asymmetric keys, and (ii) by leveraging what I refer to as a *file-store deposit* (a pointer to a dropbox in the cloud) to assist in symmetric key management. Accelerometer data from two smart phones is used in [23] and [7] to aid in authentication for secure pairing. Mayrhofer et al. [23] employ a strategy of shaking two phone simultaneously to generate a *movement limited* channel, while BUMP [7] uses the accelerometer *and* location data between to bumping smart phones. Again, our work differs from pairing protocols based on movement limited channels, by operating over location-limited and multichannel communication channels.

Multichannel security protocols, as surveyed in [37], are ways to mitigate against MitM attacks by using multiple communication channels, e.g., radio, visual and 1-bit on/off or toggle buttons, during authentication. The idea is that a malicious eavesdropper cannot eavesdrop on all channels. We use an instantiation of this idea in the variant of our pairing protocol based on historical, multiple open-lines of communication.

## 2.2 LIMITATIONS OF PRIOR WORK

As alluded to previously, LSS have significant privacy issues, and in fact the primary issue was exposed in [6]. In this study, it was shown that that users are uncomfortable with a service controlling access to their location data. Techniques have been introduced to mitigate users' privacy issue concerns, e.g., data can be diffused, or aggregated, but all of these reduce the utility of the data. This is especially troubling if the providers' intentions are for their data to be consumed at a high precision by a specific user, or one or more groups of users. A secondary issue that arose in the study is that many users are uncomfortable knowing that *anyone* can see their location information, i.e., once the location data is uploaded to the LSS the user forfeits control over the data.

Although not novel, the combination of symmetric and asymmetric cryptography can help address both of these concerns: i.e., providers encrypt their location data prior to uploading it to a LSS, and then must distribute the decryption key(s) to enable retrieval by authorized consumers. However, this key management process can be a heavy burden—the most common form of cryptography between Internet parties relies on a public-key infrastructure (PKI), which in turn relies on one or more trusted-third parties (e.g., Certificate Authorities or CAs). As of today, other than the "PGP PKI" (which utilizes a *web-of-trust*, as opposed to a true CA), or the "Web PKI" (which relies on a cartel of CAs being included in the predominant web browsers used today), no PKI exists that mobile devices can tap for the necessary key management that a privacy-enabled LSS would require.

Interestingly, McCune et al. and Farb et al. [13, 24] observed a decade ago that smart phones were almost ubiquitous and are exceptionally portable and, as such, are usually available during *vis-a-vis* interactions. This makes smartphones an ideal platform for bootstrapping the exchange of cryptographic keys through the location-limited channels that can exist between two parties. We further observe that current smart phones possess the technology to perform key management efficiently and fully over location-limited channels, but only lack the protocols and framework to achieve this. SafeSlinger [13] is architected to rely on Internet connectivity to/from a server to aid in the key exchange protocol (i.e., SafeSlinger only uses the location-limited channel between the pairing smart phones for initializing the key exchange, and then for confirmation). This has the obvious disadvantages of requiring (i) that the server be available at all times, and (ii) that the exchange occurs in an environment that possesses network connectivity. If a user is content to share private data with only principals within their social network, we argue that both of the above requirements are unnecessary to exchange asymmetric keys in a close, vis-a-vis setting that leverages location-limited channels, while using current smart phone technology.

## 3.0 SYSTEM DESIGN

The SLS system was designed with two main goals in mind: (i) enabling tunable and private location sharing with limited contacts, and (ii) limiting end-user location over exposure. We now overview the system architecture and describe the threat model within which we expect SLS to be used.

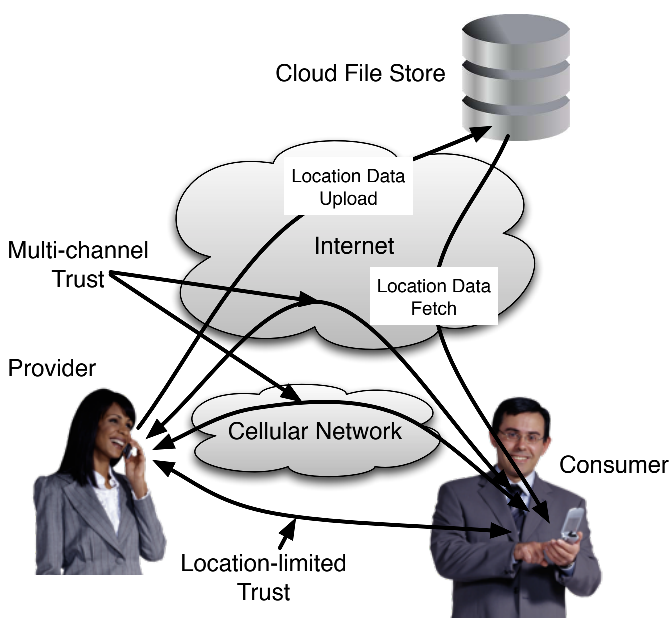


Figure 1: SLS Architecture Overview.

## 3.1 ARCHITECTURE OVERVIEW

Figure 1 presents a high-level view of the SLS system. Users in the system can be divided into two classes: *providers* and *consumers*. Providers share their location with others, while consumers retrieve the locations of others; a user can act as both a provider and a consumer. We assume that all users have smartphones, as well as (perhaps self-signed) asymmetric key pairs. Providers' smartphones must be able to detect their current location, e.g., via GPS or cellular/WiFi localization. Private location sharing is enabled by shared, symmetric keys. The sharing of these symmetric keys is facilitated by asymmetric keys exchanged during a device pairing protocol. SLS provides two pairing protocols to exchange asymmetric keys: one based upon in-person communication over location-limited channels, and another that leverages multi-channel communication for situations in which in-person exchange is not possible. To pair devices using location-limited channels, users' smartphones must have the ability to read and decode QR codes. To pair devices using multiple, historical open-lines of communication, users' smartphones must have both Internet access (e.g., via WiFi), as well as the ability to send and receive SMS messages over the cellular network. Although the multi-channel based pairing protocol requires that principals have previously communicated, there is no such restriction within the location-limited pairing protocol. Encrypted location data is shared through the use of a *SaaS service* (e.g., Amazon S3 or Google Drive) contracted by the provider. Note, although one can currently find SaaS services that are free, our model assumes an associated cost to use the service. We require that the SaaS service allow any user to download data posted to the provider's account (i.e., world readable option). There is no requirement that all providers must use the same SaaS service.

## 3.2 ADVERSARY MODEL

In this thesis, we make the following assumptions. We first assume that user smartphones are free of malware, as this would immediately make user locations available to the adversary through the smartphone API. We assume that all network communications are subject to read, replay, reorder, and modification by a Dolev-Yao style adversary [10]. [1] Finally, we assume that the SaaS providers employed are honest-but-curious in nature. That is, we assume that they will correctly execute the `GET` and `PUT` operations provided by their APIs, but may try to derive provider locations by inspecting the data that they host. In this work, we do not address DoS/DDoS attacks against SaaS providers as a means of thwarting location sharing.

[1] We alter this assumption to *at most one* communication channel when analyzing our pairing protocol based on multichannel communication.

## 4.0 SECURE LOCATION SHARING

We now describe the design of the Secure Location Sharing (SLS) framework. We first describe how multiple granularities of location data are encrypted and managed by the provider (Section 4.1). Then, we describe two protocols for pairing provider and consumer devices to enable secure retrieval of provider locations from SaaS services (Sections 4.2–4.3). Next, we discuss the policy controls available to providers within SLS (Section 4.4). Finally, we describe the iOS implementation of SLS (Chapter 5).

### 4.1 LOCATION SHARING

In SLS, a provider's smartphones is responsible for capturing her location data using, e.g., WiFi or cellular localization or GPS. Each location sample collected by SLS is represented as a four-tuple containing a location coordinate, an estimate of the provider's speed of travel, the providers bearing/heading, and a timestamp indicating when the sample was collected. In total, each location sample collected by SLS requires approximately 200 bytes to store. Given that a provider may wish to share her location at multiple granularities, the sample collected by SLS is generalized to each desired granularity or precision prior to upload: exact, building, neighborhood, city, county, or state precision. Where *exact* precision uses all available decimal places within the coordinate degrees; *building* uses only four of the available decimal places; *neighborhood* uses three decimal places; *city* uses two; *county* uses just the first decimal place, and *state* precision only uses the whole number in the coordinate degrees [17]. These exact or generalized provider locations are shared with consumers via an (untrusted) SaaS service contracted by the provider. As such, location data must be cryptographically protected prior to upload. To accomplish this, the provider generates one symmetric key for each granularity level at which her data is to be shared (i.e., exact, building, state, etc.), and then cipher-block-chaining (CBC) encrypts each sample prior to upload.

Encrypted location samples are thus unreadable to the SaaS service, with whom the user is under no obligation to share her location (unlike in a traditional LSS). We note, however, that there is economic incentive for the SaaS service to correctly house the data, regardless what the data's contents are, in that users are not bound to a particular SaaS service and can simply migrate their data should the SaaS service misbehave. Providers also have complete control over the amount of information shared: they may post only a single "current" location (e.g., by overwriting a single location sample), or instead maintain a history of location samples (e.g., by storing a sliding window of $n$ location samples). In SLS, we refer to these two operational modes as *update* and *history*, respectively. Finally, consumers are under no obligation to create or maintain accounts with each LSS that their providers use, as all data is pulled from SaaS file-stores by SLS using HTTP GET requests made to world-readable URLs.

Of course, the reliance of SLS on symmetric keys to protect provider location data raises two issues. First, it must be possible for providers and consumers to securely authenticate one another and exchange the cryptographic material needed to retrieve location data at the desired level of granularity. To this end, we present protocols for device pairing based on location-limited and multi-channel protocols in Sections 4.2 and 4.3, respectively. Second, it must be both possible and efficient for the provider to alter the list of consumers with whom she shares information and the granularity at which this information is shared, which are policy challenges that are addressed in Sections 4.2 and 4.4.

## 4.2 LOCATION-LIMITED PAIRING

In-person, interactions are an ideal setting for device pairing and key exchange. These intimate interactions between potential providers and consumers present the users engaging in the pairing protocol with an opportunity to physically identify the owner of the device with which they are attempting to establish a secure channel, as well as enable the use of location limited (e.g., visual [24] or Near Field Communication [26]) channels to exchange data, all the while presenting

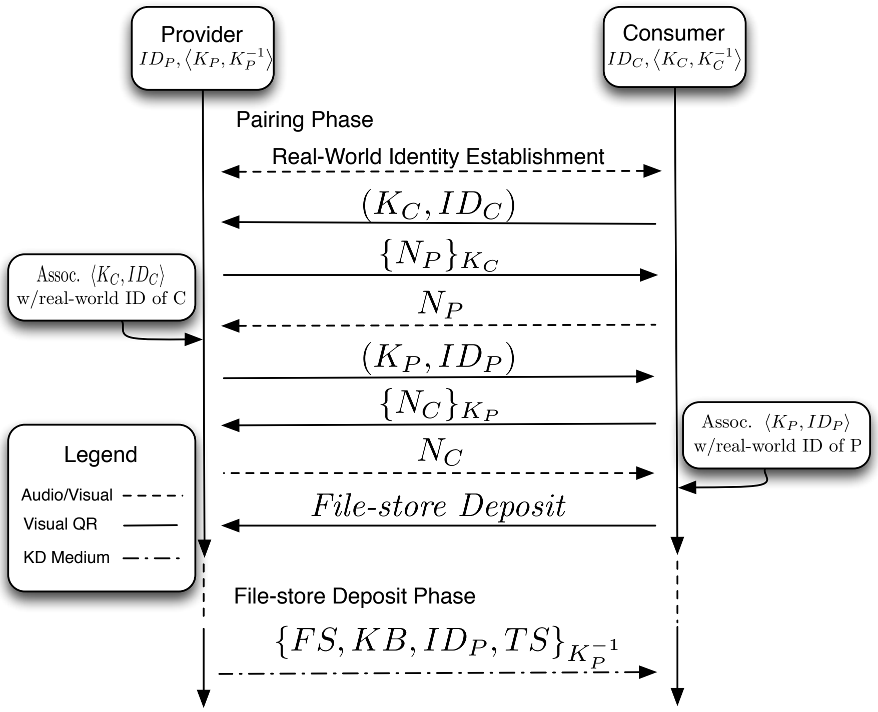


Figure 2: Communionable Trust Protocol.

a nigh impossible target for eavesdroppers intent on mounting MitM attacks—there simply is not sufficient area for current technologies to intercept, much less manipulate communications. The combination of human-to-human authentication and device-to-device communication that is difficult to intercept results in *demonstrative identification* [4] of the participants in location-limited device pairing protocols. SLS utilizes the traits of visual, location-limited channels, and extends the concepts presented in SiB [24] when pairing devices to aid in symmetric key management. During the pairing process, asymmetric, or public keys are exchanged, which are then used to wrap/unwrap the shared symmetric keys during transport.

We define *Communionable Trust* to be the confidence that an asymmetric key received through *intimate communication* is bounded to the principal's identity at the remote end of the intimate communication channel. Note, a trusted-third party is not necessary in this definition—the recipient has high confidence that the received key was given to them by the visually identified party.

Since our location-limited device pairing protocol is based on close, intimate communication channels, we refer to it as *Communionable Trust*, or simply *CT*. Figure 2 illustrates *CT*.

### 4.2.1 Pairing Phase

This protocol makes use of human-to-human audio and visual communication, as well as device-to-device visual communication using on-board cameras and Quick Response (QR) codes. The first step of this protocol is the real-world identification and authentication of the humans who wish to pair devices to facilitate location sharing via SLS. After the human participants have agreed to pair devices, the remainder of the protocol focuses on the exchange of public key information between provider and consumer, and exchanging metadata that enables the sharing of both symmetric keys and location data.

In Step 2 of the protocol, the consumer generates a QR code that contains her public key ($K_C$) as well as a device identity token ($ID_C$) used to associate her device with her real-world identity, as managed by the provider's smartphone. The inclusion of an *identity token* is necessary, because symmetric key sharing happens *out-of-band* from device pairing and a consumer, for example, may choose to use an identifier for a provider which is different than how the provider identifies themselves, e.g., *John Doe* vs. *Johnny Doe*. Thus, exchanging a hash token that represents a principal's identity ensures that all future communication will be associated with the correct identifier. Figure 3 shows a QR code containing a 2048-bit RSA public key and its associated identity token.[1] The provider scans this code with his phone, and recovers $K_C$ and $ID_C$. He then generates a random challenge nonce,$N_P$, encrypts $N_P$ using $K_C$, and generates a QR code containing the resulting ciphertext (Step 3). The consumer scans this QR code, decrypts the resulting ciphertext, and verbally communicates the nonce value to the provider (Step 4). After verifying this exchange, the provider associates $K_C$ and $ID_C$ with the consumer's contact information in his smartphone.

This process is then mirrored in Steps 5, 6, and 7 of the communionable trust protocol, which provides the consumer with the providers public key ($K_P$) and identity token ($ID_P$).

In the final step of the Pairing Phase, the consumer QR-encodes their *file-store deposit*—a description of an out-of-band channel (e.g., `type=sms, phone-number=4125551212`) over

[1]Version 40 QR codes can encode approximately 1500 bytes of data, which is more than sufficient for exchanging even 2048-bit public keys.

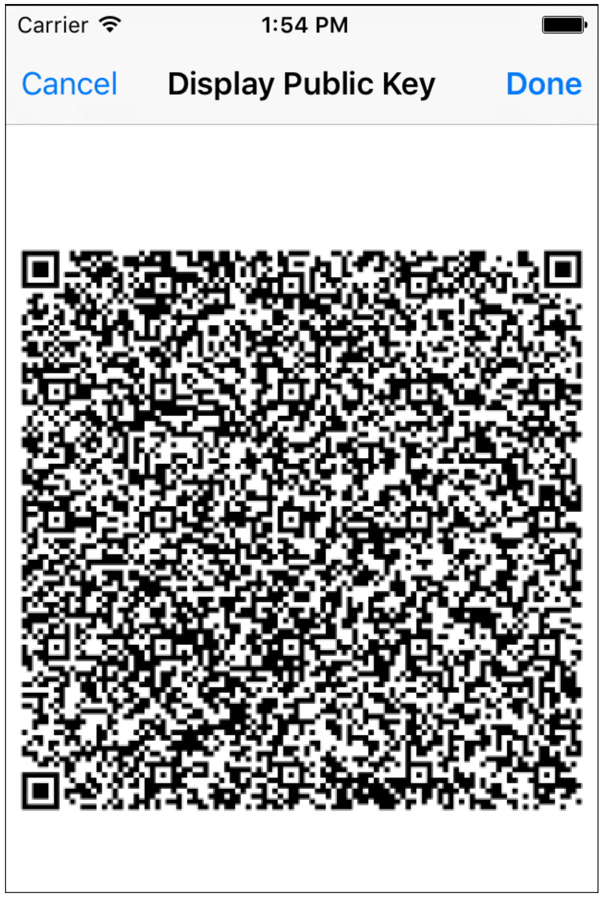


Figure 3: QR-encoded 2048-bit RSA public key (with associated identity token).

which the consumer wishes to be notified of the provider's SaaS file-store—and presents it for the provider to scan. This message is sent unencrypted due to the location-limited nature of the visual channel used by this protocol. The use of this consumer-specified "drop box" allows the provider to inform the consumer asynchronously (e.g., via SMS) if they change SaaS providers at a later date, and thus obviates the need to re-execute the CT protocol.

#### 4.2.2 File-store Deposit Phase

The final message of the communionable trust protocol handles the distribution of metadata that enables the consumer to retrieve location samples uploaded by the provider. Asymmetric keys only enable two parties to communicate securely, the SLS framework is a one-to-many infrastructure. That is, one provider can have many consumers, and one consumer may have many providers—to address this, SLS relies on shared or symmetric keys to map one or more consumers to a single provider. In short, a provider uses policy to dictate which groups, or what we refer to as *precision levels*, each of their consumers are assigned to (see Section 4.4).

The File-store Deposit Phase can occur anytime after the provider configures policy for the consumer, i.e., the consumer is assigned a precision level sometime after the pairing phase completes (see Figure 2). The message comprising the file-store deposit is sent over the channel identified in Step 8 of the communionable trust protocol, and is a four-tuple of values containing a URL for the file store at which the provider's location data will be hosted ($FS$), a URL at which the consumer can access her shared key bundle ($KB$), the provider's identity token ($ID_P$) [2], and a timestamp ($TS$) to prevent replay attacks. The entire message is then signed by the provider to ensure authenticity. After using $TS$ and $K_P$ (which is associated with $ID_P$ by the consumer) to validate the freshness and authenticity of this message, $FS$ and $KB$ provide the consumer with all of the information that is needed to securely access the provider's location data.

The key bundle URL, $KB$, provides the consumer with a pointer to an encrypted key bundle stored on the provider's SaaS service. This key bundle is a (*key*, *version*, *signature*) three-tuple that is encrypted using the consumer's public key ($K_C$). The *key* field of this tuple contains the current symmetric key corresponding to the precision level with which the consumer is permitted to access the provider's location, the *version* field indicates the version of this key, and the *signature* field is a hash of the key and version fields signed with the provider's private key ($K_P^{-1}$). Key versions are used to facilitate location retrieval as keys change in response to changes in provider access controls (see Section 4.4). The level of indirection added by the key bundle—as opposed to directly transferring keys as part of the CT protocol—eliminates the need for direct communication between the provider and consumer upon every policy change. After recovering their key bundle, the consumer can periodically retrieve provider locations (either the current location data or the history log, see Section 4.1) from the file store URL, $FS$, and decrypt this data. Note, the File-store Deposit Phase only occurs when a consumer is initially assigned policy, or if and when the

[2] The identity-token is included in the file-store information to provide the consumer with a way of quickly determining which provider sent the message.

provider changes their file-store (e.g., changing their cloud services).

## 4.3 MULTI-CHANNEL PAIRING

It is unreasonable to assume that users of SLS will always have the ability to physically co-locate during the device pairing process. As such, we also describe a pairing protocol that can be used by individuals who are not within close proximity. As such protocols can be vulnerable to MitM attacks, we make use of multiple *historical, open-lines of communication* associated with principals on their smart phones (e.g., email address, phone number, or instant messaging account). In this context, *historical* refers to pre-existing contacts within the smartphone's address book, and *open-lines of communication* implies that the principals have communicated with the preexisting contact over those multiple channels. This combination of properties gives providers (resp. consumers) higher assurance that the identity of the consumer (resp. provider) is correct, since (i) existing contact information is used to bootstrap the communication process and (ii) an active attacker would need to control multiple communication channels to subvert the protocol.

Wong et al. [37] present protocols that use multiple channels of communication to mitigate against MitM attacks during authentication or key exchange. Although the need for the multiple channels in [37] is more to combat the relative low bandwidth of the channels possessing *data origin authenticity*—the user of the receiving device knows for sure that the received data was sent by the intended source device—the premise of the idea is sound. That is, if two *distinct* communication channels are used in key exchange, the eavesdropper has a significantly harder task in controlling the communication.

Note, the historical, open-lines of communication that a user has listed in their address book do not possess the data origin authenticity property, unfortunately. Thus, a protocol based on historical, open-lines of communication must also have strong authentication assurances, i.e., *whom* the user is talking to is indeed *whom* the user thinks she is talking to. To address the lack of data origin authenticity, our protocol couples the notion of using multiple communication channels with "secret questions" to provide both parties with *reasonable* confidence that the principal at the remote end of the multichannel communication is indeed the owner of the public key exchanged over

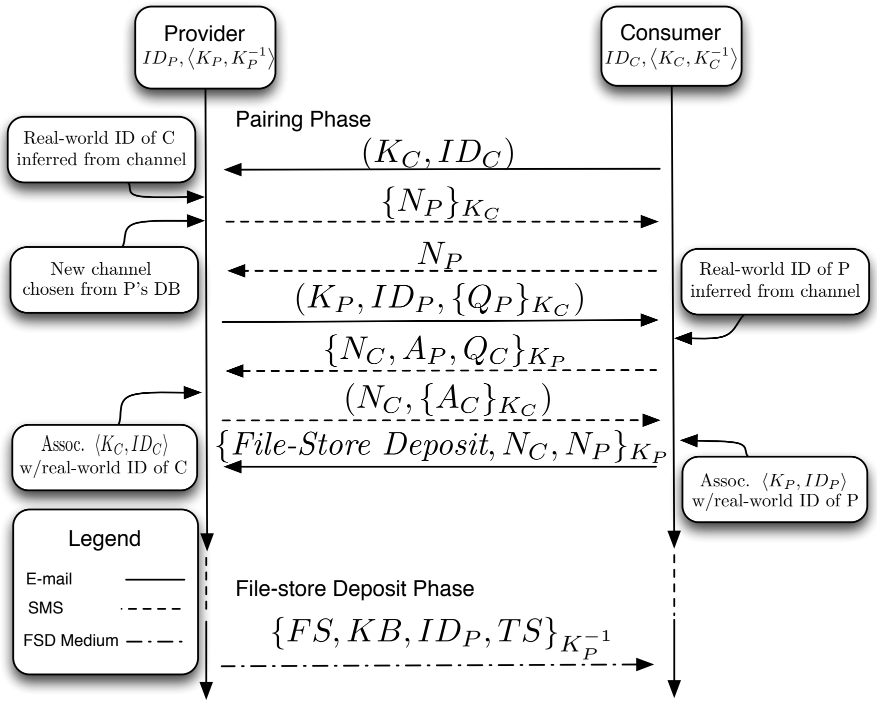


Figure 4: Historical Communication Channels Protocol.

one of those channels. The *Historical Communication Channels (HCC)* protocol is described in Figure 4.

HCC is initiated in the first step of the Pairing Phase by the consumer, who sends their public key ($K_C$) and device identity token ($ID_C$) used to associate her device with her real-world identity (which is established through previous contact as managed by the provider's smartphone). This message is sent to the provider over an existing communication channel (e.g., a known email address), signified using a solid line in Figure 4.

Upon receiving $K_C$ and $ID_C$, the provider generates a random challenge nonce $N_P$, encrypts $N_P$ using $K_C$, and sends $\{N_P\}_{K_C}$ to the consumer via a *different* historical, open-line of communication that the provider has previously associated with the consumer in their address book (e.g., via SMS), which is denoted by a dashed line in Figure 4.

The consumer decrypts the ciphertext and returns $N_P$ back to the provider over *HCC Sec-*

*ondary* (Step 3). At this point in the protocol, the provider is confident that the consumer has access to the private key associated with the public key received in Step 1.

However, the provider does *not* yet have a high level of confidence that the consumer is indeed whom the provider believes they are (e.g., someone other than the consumer could have stolen the consumer's smart phone). Hence, similar to the use of *secrets* in OTR [1], the provider generates a *secret question* ($Q_P$) that, within reason, only she and the consumer should know the answer to; e.g., *"Who was the away team at the last hockey game that we attended together?"*. $Q_P$ is encrypted with $K_C$ and is sent to the consumer along with $K_P$ and $ID_P$ over the primary channel (Step 4).[3]

The consumer generates (i) the answer ($A_P$) to $Q_P$, (ii) her own random nonce $N_C$, and (iii) her own *secret question* ($Q_C$). All three are encrypted with $K_P$ and sent to the provider via *HCC Secondary* (Step 5). If, after decrypting the resulting ciphertext, $A_P$ is correct the provider sends $N_C$ and her answer ($A_C$) encrypted with $K_C$ via *HCC Secondary* (Step 6). After verifying $A_C$, the consumer encrypts their file-store deposit (*File-Store Deposit*), $N_P$ and $N_C$ with $K_P$ and sends them via *HCC Primary* (Step 7).

Finally, similar to the CT protocol, the provider assigns the consumer's precision level and the File-store Deposit Phase begins (see Section 4.2.2 and 4.4). If either (i) a principal receives a *secret question* via SLS without first initiating or receiving a public key from the same principal over a different channel, or (ii) the response to a participant's secret question is incorrect, the HCC protocol *must* terminate immediately.

## 4.4 POLICY CONTROL

As alluded to in the Section 4.2, after learning the consumer's public key and file-store deposit (either through CT or HCC), the provider must associate the consumer with the precision level at which they are authorized to view the provider's data. All consumers that are assigned the same precision level by a provider are considered to be in the same group and, thus, all have access to a

[3] Although secret questions are a questionable strategy employed by some services to allow principals to bypass potentially strong password authentication, employing them in this context—between two principals—fits the socially-based authentication model well.

single symmetric key protecting location disclosures made at this precision level. As a result, the symmetric key associated with a particular precision level may need to be updated as the group of users who have access to that precision level changes over time.

To provide the highest level of security for the provider's location data—i.e., preserving forward and backward secrecy—these shared symmetric keys should be changed whenever a consumer is *added* to a precision group or *removed* from a precision group. The former case ensures that new consumers cannot access old data, while the latter ensures that former consumers cannot access new data. Altering the symmetric key for a particular precision level requires creating a new symmetric key, encrypting key bundles for each user authorized at this precision level, and depositing the bundles on the provider's file store. After asynchronously retrieving these new key bundles, authorized consumers can again access the providers data. While shared symmetric key update is non-trivial, our evaluation (Section 6.1) shows that the overheads associated with this process in practice are minimal. We note that it is not necessary for the provider to re-pair their device with consumers via CT or HCC, as the asymmetric keys use for key management are *not* affected by a consumer's change in precision level.

We recognize that our LSS model prevents the enforcement of certain policies found in existing LSSs; e.g., policies that enable location sharing only when two parties are within a certain physical proximity, or policies that place access count limits on individual users. However, we observe that LSSs that can implement proximity-based policies are able to do so because they have access to the location data of all their users, and can thus determine the distance between two users. Since our goal is to prevent the LSS from acquiring this omniscience, this type of policy can not easily be enforced in SLS. Enforcing constraints on access frequency is also enabled via LSS intervention, which is contrary to our assumed sharing model. We do note that the ability to enforce these types of policies would be worthwhile additions to SLS. That said, we defer the exploration and development of techniques for achieving these goals to future work.

Finally, the provider also controls how often new location updates are available to all consumers, and as a metaphorical *panic button*, the provider can always disable future location updates to everyone; reenabling the location updates when the provider sees fit. This, and the granularity of precision-levels allows SLS providers to express effective policies tractably.

# 5.0 IMPLEMENTATION

SLS was implemented initially as an iOS 6 iPhone application and installed on an iPhone 4s. A second implementation was developed for iOS 7 and installed on an iPhone 5s. The location-sharing and CT pairing protocol were evaluated via the IPhone 4s, 5s and the iPhone simulator (modified to behave as if it could *scan* the iPhones' public keys).

## 5.1 PRECISION LEVELS

Our SLS implementation collects and stores provider locations as GPS coordinates, and provides six precision levels at which a these locations can be shared. The precision levels supported are *exact*, *building*, *neighborhood*, *town*, *county*, *state*, and *none*. Support for the building, neighborhood, town, county and state sharing levels are provided by masking lower-order bits in the exact GPS coordinates stored within SLS. For example, neighborhood precision equates to three decimal places of precision, hence, all extra decimals places are overwritten with zeros. The precision *none* implies that the consumer does *not* receive a symmetric key.

## 5.2 MANAGEMENT INTERFACE

The utility and usability of a security system's policy interface is crucial to its successful use: the ability to clearly indicate *who* can access an individual's data and *at what precision* is key. We approached this in SLS by presenting the provider with a clean, simple display that consists of a list of principals (i.e., smartphone-managed identities associated with each consumer), and the

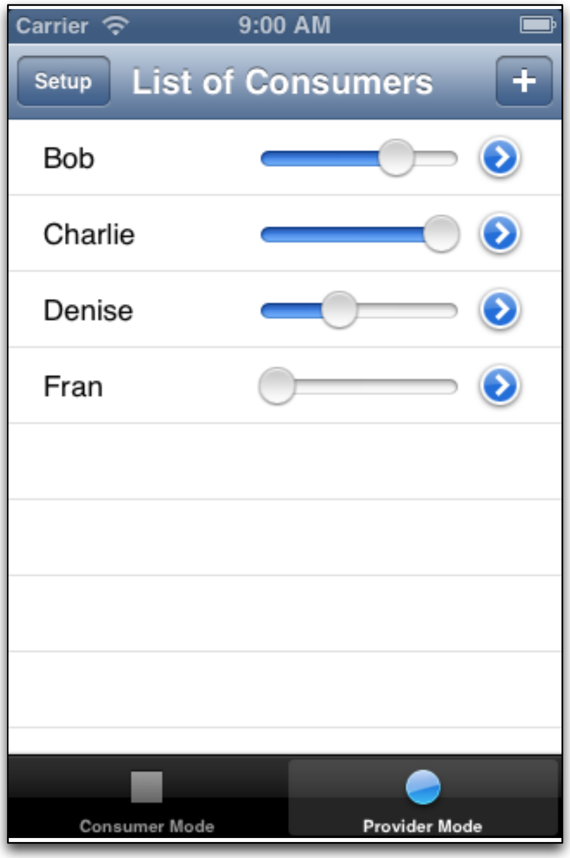


Figure 5: List of Alice's Consumers in SLS.

precision-level at which each consumer has been authorized (see Figure 5). The precision level can trivially be changed by the provider by adjusting a slider within this interface. Additionally, a *detail view* icon at the end of each row allows the provider to immediately review the consumer's information, resend the consumer's shared key bundle URL, or delete the consumer.

New consumers can easily be added when the provider taps the [+] button in the upper-right corner of the *List of Consumers* view (see Figure 5). This presents the provider (or the consumer, when they navigate into their respective *Add Provider* screen) with a choice of either the location-limited, CT-based key pairing protocol, or the HCC pairing protocol to exchange public keys and the consumer's file-store deposit.

Figure 6 shows several steps of the CT device pairing process, as executed by the provider and consumer. In the first step, the provider (Alice) has entered the consumer's identity (Bob), and Bob has done the same for Alice. When both tap the QR-code button, SLS will present both the provider and consumer with the task list associated with device pairing in CT (Step 2). The provider and consumer will iterate through the steps, synchronizing when QR images are printed and scanned, or during challenge/responses—these steps have been omitted from the Figure for clarity. In Step 7, the provider taps the button to scan the consumer's file-store deposit as a QR code, and Step 8 shows the consumer's screen after displaying their file-store deposit QR code to the provider.

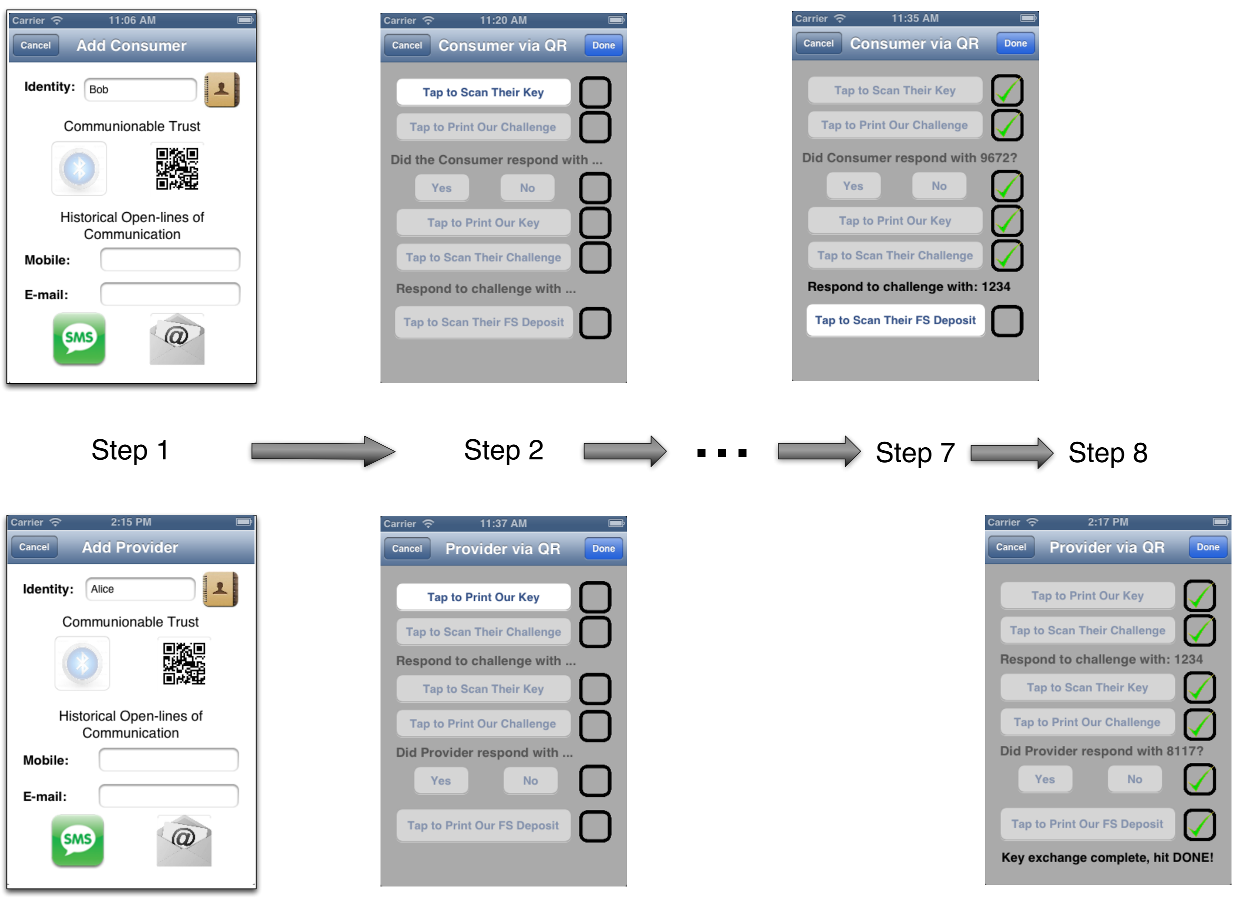


Figure 6: Screenshots of Communionable Trust protocol implemented in SLS. Steps 1, 2 and 7 are shown for the provider, and Steps 1, 2, and 8 are shown for the consumer.

### 5.3 SAAS SUPPORT

Our current implementation of SLS supports the use of Amazon's S3 and Google Drive services as cloud file-stores.[1] The period at which SLS updates GPS coordinates and then uploads the location data is configurable by the provider.[2] For simplicity, our implementation used the iOS Core Library API to serialize the location data that was gathered for the provider, with the resulting serialized location objects being $1KB$ in size. Although these objects are larger than those that could be produced by a custom serializer, the encryption and transmission overheads associated with these larger objects are not significant even on the older iPhone 4s.

The cloud file-store is also used to store encrypted key bundles for the consumers associated with a given producer. Whenever a consumer is added or a consumer's precision level is changed, the shared keys (for all precision-levels involved) are regenerated, new key bundles are generated and uploaded to the provider's file-store, and all future location samples are encrypted using the new key. Upon detecting a key version mismatch, the consumer's SLS application automatically looks to fetch a new key bundle from the provider's file-store. The new key can then be used to decrypt more recent location updates, which may be stored in either *update* or *history* mode. To support history mode, updates are stored in a log file referred to as the *history log*, which contains an entry for each location data update in the window.[3] The entries consist of the location data, time stamp and a signature over a hash of the two components. Thus, a consumer can check the history log to fetch any updates they may have missed, as their periodicity can be set differently than the providers.

[1] An example file-store URL using Amazon's S3 service is: `https://s3.amazonaws.com/id-precision-hash/locationdata.b64`, where `id-precision-hash` is a hash of the provider's identity token and the precision level assigned to this location data.

[2] iOS has two settings for location gathering, the first operates using a *distance filter* to determine when a new location update should occur, the second is a *power-saving* mode, in which a location update only occurs during a "significant" location change. Both modes are supported in our implementation.

[3] Unfortunately, the current Google Drive SDK for iOS does *not* support appending, so our implementation was changed to always upload a sliding window of history in the log.


Figure 7: Consumer's view of a provider walking up the street.

## 5.4 MAP VIEW

The consumer's initial view displays their providers' positions on a map (see Figure 7). If the provider is operating in history mode, in which a log of location data updates is being kept in their file-store, the consumer can view the provider's location as a path. Since the location data stored in the cloud also contains the bearing of the provider (in addition to the location coordinate), SLS can plot the provider's location using the entire history that it can obtain. This resulting view is a trail of footprints corresponding to the provider's path along the map. Tapping on a provider's footprints displays the provider's name and a *details disclosure* button, which leads to that provider's detailed information view.

## 6.0 DISCUSSION

We developed a system for enhancing user privacy during location sharing by adopting and extending previous work in device pairing, and tapping into the ubiquity and availability of cloud services. Our system was architected to be both scalable and secure, while also preserving providers' sharing policies and affording utility to consumers. We now discuss both the performance of our SLS implementation, and assurance provided by our pairing protocols.

### 6.1 PERFORMANCE EVALUATION

In evaluating the runtime overheads of SLS, we break our analysis into four phases: the device pairing phase, the file-store deposit phase, location data upload by providers, and location data download by consumers. To evaluate the communionable trust protocol (Section 4.2), we carried out 15 pairings using our iOS implementation of SLS and found the average time required for this process was $110$ seconds. [1] Although this pairing process takes longer than, e.g., Bluetooth device pairing, the overheads are reasonable given the human effort required by this protocol (i.e., scanning QR codes). We did not evaluate the time required by the historical communication channels protocol (Section 4.3) as this protocol was designed to be run asynchronously over multiple higher latency channels (e.g., email and SMS).

After devices are paired using either the CT or HCC protocol, in the file-store deposit phase the provider sends the consumer a digitally signed message that includes URLs that are to be used to retrieve the consumer's key bundle and the provider's location data. This data can only be sent after

[1] The simulator was faster at several things, e.g., typing in the user's identity, however, since the protocol was lock-step, the average times should still be within reason.

the provider has indicated the precision level with which the consumer may access his information, and is returned to the consumer over a communication channel identified in the final message of the pairing phase of the CT or HCC protocol (i.e., the file-store deposit message). The overhead associated with this phase is linear in the number of consumers, but is only a one time cost, as this only occurs once per consumer (or in the wake of the rare event of the provider changing cloud services). In our iOS implementation of SLS, the generation of the signature on this message took $98ms$ on average over 15 runs.

A consumer that is tracking $n$ providers must execute $O(n)$ operations, as they must retrieve $n$ encrypted location samples from up to $n$ different SaaS providers. The frequency with which this process occurs is a parameter that can be set per-provider within my implementation by the consumer—the linear cost could be applied once per day, or once every five seconds (if they so choose). We measured the average time required to execute the `HTTP GET` command required to obtain a provider's encrypted location sample and execute the symmetric key decryption of this sample. When using Amazon S3 as SaaS provider, this process took approximately $40ms$ per provider, where the RTT of the fetch was $35ms$. Thus, the consumer's performance is governed by the choice of cloud service each of its $n$ providers use and if the RTT was dominated by network latency, the consumer's current network path to those file-stores. [2]

On the other hand, the cost incurred by the provider during uploads is constant: a symmetrically encrypted GPS gathered location sample must be uploaded (e.g., `HTTP PUT`) at each of the $p$ precision levels used by the provider's policy. The provider can choose the periodicity that new GPS coordinates are produced, so the constant time cost can be applied once per day, or several times per minute (depending on how fast the smart phone is moving and how fast it can generate new GPS coordinates). In our iOS implementation, encrypting and uploading location samples at three precision levels over 15 runs took on average, $3.7s$, with a standard deviation of $1.8s$. Although we were unable to isolate the poorly behaving components, we suspect that the majority of time was spent blocking on our cloud provider's API.

[2]It is conceivable that the RTT may have been more dependent on the file-store's API or infrastructure, then network latency.

## 6.2 SECURITY ANALYSIS

The SLS system was designed with two main goals in mind: (i) enabling tunable and private location sharing with *limited* contacts, and (ii) minimizing end-user location over exposure. This is achieved by storing encrypted (AES-256-CBC) location samples on SaaS servers contracted by the provider, and leveraging location-limited or multi-channel pairing protocols on smartphones to facilitate the key management required by this approach. We now informally analyze the security afforded by the protocols developed in this paper.

### 6.2.1 Location-limited Pairing Protocol

The communionable trust protocol described in Section 4.2 provides principals with high assurance regarding the secure handling of location data. In particular, the face-to-face nature of this protocol allows the human device owners to authenticate each other in the most natural sense. As a result, the public keys exchanged and validated using this protocol are intrinsically tied to the real-world participants in the protocol, since anyone attempting to launch a MitM attack would be quite conspicuous—we refer the reader to the security analysis in [24] for a thorough examination of attacks against this type of channel, as well as comparisons against other channels (e.g., audio, infrared, physical contact, etc.). For this reason, the principal's public key and identity exchanged via CT are made available to other applications outside of SLS. [3] Assuming that the consumer keeps her private key a secret, the public key obtained during this process enables the provider to safely transmit symmetric keys needed to recover her location to the consumer without exposing her location to unauthorized individuals (including the SaaS provider).

### 6.2.2 Multi-Channel Pairing Protocol

In settings where the provider and consumer are not physically located together, obtaining the same assurance level from the communionable trust protocol is difficult. The protocol in Section 4.3 attempts to overcome the lack of physical proximity in three ways. First, it leverages *historical* communication channels managed by each user's smartphone to increase assurance in the identity

[3] iOS provides for this by using a shared or public attribute group within its key-chain.

of the party being communicated with, thereby reducing the likelihood of accidental sharing with inappropriate parties. Second, the HCC protocol makes use of *multiple* communication channels to decrease the likelihood of a successful MitM attack against the protocol. In examining Figure 4, one can see that an adversary with access to *only* the e-mail channel has the ability to inject public keys into the protocol, but cannot complete the validation process for these keys. Likewise, an adversary with access to *only* the SMS channel can cause parties in the protocol to reject valid public keys, but cannot inject their own public keys.

While this protocol cannot protect against a MitM attack when the adversary has access to both channels used by the protocol, as long as the implementation ensures that both channels are indeed distinct (e.g., WiFi + SMS, as opposed to 3G/4G + SMS) the cost to mount such an attack would be prohibitive to most. A more realistic attack vector against cell phones would be a physical attack, i.e., the attacker steals the smartphone of the consumer (resp. provider). However, the third protection mechanism in HCC does allow it to protect against this attack. Specifically, each party is required to answer a contextual "secret question" (i.e., $Q_P$ and $Q_C$ in Figure 4) proposed by the other party prior to finalizing the pairing process and enabling location sharing. Unless the individual possessing the stolen smartphone has intimate knowledge of the relationship between the provider and consumer, they would be unable to answer this type of question and, thus, the protocol would fail.

Although this question-and-answer mechanism is useful, it is not perfect. First, HCC requires seven messages and the cognitive power to generate and answer two "secret questions". Upon closer examination, though, this apparent disadvantage is not that significant; the HCC phase only occurs once for a provider/consumer pair. Moreover, each message would arrive as an SLS URL, so the user would only need to click on it, as the app woud do the work and only when necessary, prompt the user to enter or answer a secret question. Imagine how long it takes to send a total of seven SMS messages in one conversation—not very long. However, the protocol requires that the implementation leverage distinct communication channels, the mechanisms used to ensure the channels are distinct could themselves be attacked, i.e., an attacker could try to trick the implementation to use multiple channels that he controls. For this reason, we encourage principals to *upgrade* their public keys exchanged via HCC with CT, whenever the opportunity presents itself. [4]

[4]SLS accommodates the weaker assurance in HCC by *flagging* public keys exchanged via HCC and limiting the

### 6.2.3 Shared Key Management

The shared or symmetric key management scheme employed by SLS provides three assurances to its users, including: the assurance that only a provider can upload new key material to the cloud file-store, the assurance that the location data can *not* be accessed by anyone without the appropriate shared key, and the assurance that exposure of the identities of the provider's consumers is mitigated within the cloud file-store. The first assurance is ensured by the act of encrypting the shared key bundle with the consumer's public key, while including a version and signature, i.e., an adversary can *not* alter the contents of the bundle as it is signed. The adversary could replay (or overwrite) an older shared key bundle, however, as the version of the key is included in the bundle the consumer will know that the current shared key bundle is incorrect.

Since the shared key bundles are encrypted with the consumers' public keys, and the uploaded location data is encrypted with the shared keys contained within those bundles, the provider is assured that only those consumers possessing the appropriate shared key will be able to view the provider's location data. Granted, a consumer could make the symmetric key accessible to non-authorized users, however, we note that this is the bane of *all* shared key security systems, and its mitigation is outside the scope of this thesis.

Finally, exposure of the consumers' identities is mitigated through the use of the file-store deposit. Specifically, the provider can use any unique token to identify a consumer's shared key bundle, as the location of the shared key bundle is delivered to the consumer out-of-band via the file-store deposit. Admittedly, the cloud service will know how many consumer a provider has, and possibly how many precision levels are in use by the provider (by examining the contents of the history log, if in use). Thus, in order for the cloud service to uniquely identify a consumer, they'll have to rely on properties of the HTTP connection during the fetch (e.g., IP address, HTTP message headers).

---

access to those keys by external applications.

## 6.3 BEYOND LOCATION SHARING

Interestingly, although our system was designed to share location data, there is no reason that the protocols and framework constituting SLS could not be adopted for other types of private data sharing (e.g., documents, pictures, music, or videos). That being said, we do not envision these SLS-like services replacing forums like Flickr or YouTube, which have proven to be de-facto file-stores for widely sharing information. However, these SLS-like services could be useful for providing secure and private hosting for information that is to be shared with a more limited audience.

## 6.4 FUTURE WORK

The most pertinent area of future work involves conducting a user study to assess the utility and ease-of-use of our SLS implementation. Although the focus of this thesis was on the correctness and feasibility of the system, ensuring that SLS is indeed usable is also quite important. A user study of our prototype iOS and a yet undeveloped Android implementation could help answer questions about the usability of the policy interface built into the application, as well as about the ability of users to manage their privacy using SLS's management interface. Insights from exit surveys conducted with participants in such a study could also help guide the design of more intuitive sharing interfaces, and protections that might help further limit unintended audience sharing (e.g., short-term sharing settings, etc.).

Another area that we intend to pursue is the development of more advanced and cooperative policy controls. As previously discussed in Section 4.4, adding some form of cooperative policy controls could prove invaluable. For example, consider a provider that only feels comfortable sharing her location data with consumers in the same region. We envision that this could be accomplished if the pair had mutual sharing configured between them (i.e., both acted as providers and consumers), and secure function evaluation or a protocol based on Narayanan et al. private proximity testing [25] was leveraged to decide when the distance between the two principals crossed some threshold (at which point SLS would operate normally). Although such advanced controls are likely possible, making these controls both intuitive for the user and efficient for the device to execute could prove to be challenging in practice.

Similarly, an intriguing extension of SLS's key management protocols would be to implement key escrow support for emergency response. Specifically, the symmetric key used in encrypting high-precision location data could not only be encrypted with the consumers' public keys, but also broken into *shares*, to be distributed to emergency respondents using a threshold scheme (e.g., [32]). For example, consider a 2-of-$n$ threshold scheme in which key shares are distributed to the local police force for the provider's municipality, the state police, and the security contractor used by the provider's employer. In this case, for instance, the provider's employer could cooperate with the local authorities in the event that the provider was missing long enough for the employer to file a missing persons report. Further developing these sorts of policy-based extensions to SLS could prove to be an interesting area of study.

A mechanism for *transitive trust* would allow a user's PKI to grown along its social network—Safeslinger [13] introduced an interesting feature that resembles a type of a *"friend of a friend"* bootstrapping process for its key exchange. Although we would need to be very careful leveraging any sort of a *"friend of a friend"* attribute, the notion is compelling. Presumably, any mechanism to initialize our key management protocol through using this attribute would operate over HCC, so this utility would be held to the same controlled assurances as HCC.

Moreover, HCC may be further enhanced by pursuing a means to *protect* the answers to secret question. This could be realized by employing a zero-knowledge proof; Alexander and Goldberg [1] introduce the idea of using zero-knowledge proofs in the OTR protocol to see if both parties agree on a secret, without divulging what that secret is. Although it may prove to be only a marginal improvement, SLS's privacy assurances could benefit by adopting a similar approach.

Finally, as alluded to in Section 6.3, SLS's framework could be modularized to support sharing any form of private data. Specifically, the code for both pairing protocols should be extracted into a separate app designed to be used opportunistically by users. Furthermore, the file-deposit phase and the cloud file-store operations could easily be repackaged as a library. With the key-exchange pairing app devoted to building a PKI in the smartphone's key-chain, and the above mentioned library, any app could leverage the same framework SLS enjoys.

# 7.0 CONCLUSIONS

Location-sharing systems (LSSs) have high utility, but research has shown that (i) many users are wary of sharing detailed trace information with LSS operators and (ii) the large social networks with which LSSs are often integrated make it all too easy to accidentally share location data with unintended audiences. In this thesis, we make inroads to the above problems by developing SLS, a framework for private location sharing that combines the use of social authentication protocols based upon location-limited or multi-channel protocols, with user-contracted cloud SaaS providers to facilitate secure data storage and location sharing. In particular, our device pairing protocols leverage the smaller social networks managed by user smartphones to provide a high degree of assurance in the identities of the individuals with which sharing is to occur. The asymmetric keys exchanged during this process can then be used to distributed shared symmetric keys that protect a location provider's sensitive location information from unauthorized viewers, including the cloud service used to host the data. Our iOS prototype implementation of SLS shows that the overheads associated with this form of key management are reasonable.

Although the information sharing model developed in this thesis was developed to facilitate secure location sharing based upon an individual's limited, real-world social contacts, we believe that my techniques have application beyond location sharing. In particular, the combination of device pairing with high identity assurance and third-party storage that is used by SLS's framework can be used to facilitate the sharing of many types of information currently shared using existing social networks (e.g., photos, etc.) without requiring implicit trust in the operators of these social networks.

# BIBLIOGRAPHY


[1] C. Alexander and I. Goldberg.
Improved user authentication in off-the-record messaging.
In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 41–47, New York, NY, USA, 2007. ACM.

[2] Amazon s3.
https://s3.amazonaws.com/.

[3] J. Backes, M. Backes, M. Dürmuth, S. Gerling, and S. Lorenz.
X-pire! - a digital expiration date for images in social networks.
*CoRR*, abs/1112.2649, 2011.

[4] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong.
Talking To Strangers : Authentication in Ad-Hoc Wireless Networks.
In *ISOC Network and Distributed Systems Security Symposium (NDSS)*, 2002.

[5] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea.
Real life challenges in access-control management.
In *CHI 2009: Conference on Human Factors in Computing Systems*, pages 899–908, April 2009.

[6] J. T. Biehl, E. G. Rieffel, and A. J. Lee.
When privacy and utility are in harmony: towards better design of presence technologies.
*Personal and Ubiquitous Computing*, 17(3):503–518, 2013.

[7] Bump.
http://bu.mp.

[8] Y. Cai and T. Xu.
Design, analysis, and implementation of a large-scale real-time location-based information sharing system.
*Proceeding of the 6th international conference on Mobile systems applications and services MobiSys 08*, page 106, 2008.

[9] Datalocker.
http://www.appsense.com/labs/data-locker.

[10] D. Dolev and A. C. Yao.
On the security of public key protocols.
In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.

[11] S. Egelman, A. Oates, and S. Krishnamurthi.
Oops, i did it again: mitigating repeated access control errors on facebook.
In *CHI 2011: Conference on Human Factors in Computing Systems*, pages 2295–2304, 2011.

[12] Facebook places.
http://www.facebook.com/about/location?_fb_noscript=1.

[13] M. Farb, M. Burman, G. Singh, C. Jon, and M. A. Perrig.
Safeslinger: easy-to-use and secure public-key exchange, 2013.
http://www.cmu.edu/homepage/computing/2012/winter/safeslinger.shtml.

[14] Find my friends.
https://support.apple.com/kb/PH19425?viewlocale=en_US&locale=en_US.

[15] Foursquare.
https://foursquare.com/.

[16] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy.
Vanish: Increasing data privacy with self-destructing data.
In *Proc. of the 18th USENIX Security Symposium*, 2009.

[17] Geo hashing.
http://wiki.xkcd.com/geohashing/GPS_accuracy.

[18] Glympse.
http://www.glympse.com/.

[19] Google drive.
https://www.google.com/drive/learnmore/.

[20] Google plus.
https://www.google.com/+/learnmore/.

[21] P. Klasnja, S. Consolvo, T. Choudhury, R. Beckwith, and J. Hightower.
Exploring privacy concerns about personal sensing.
In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 176–183, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman.
I m the mayor of my house : Examining why people use foursquare - a social-driven location sharing application.

*Design*, 54(6):2409–2418, 2011.

[23] R. Mayrhofer and H. Gellersen.
Shake Well Before Use: Intuitive and Secure Pairing of Mobile Devices.
*IEEE Transactions on Mobile Computing*, 8(6):792–806, June 2009.

[24] J. McCune, A. Perrig, and M. Reiter.
Seeing-Is-Believing: Using Camera Phones for Human-Verifiable Authentication.
In *IEEE Symposium on Security and Privacy*, pages 110–124. IEEE, 2005.

[25] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh.
Location privacy via private proximity testing.
In *In NDSS*, 2011.

[26] D. Nelson, M. Qiao, and A. Carpenter.
Security of the near field communication protocol: An overview.
*J. Comput. Sci. Coll.*, 29(2):94–104, Dec. 2013.

[27] Open key chain.
https://www.openkeychain.org/about/.

[28] B. Palanisamy and L. Liu.
Mobimix: Protecting location privacy with mix-zones over road networks.
*Data Engineering, International Conference on*, 0:494–505, 2011.

[29] S. Patil, G. Norcie, A. Kapadia, and A. J. Lee.
Reasons, rewards, regrets: Privacy considerations in location sharing as an interactive practice.
In *Symposium on Usable Privacy and Security (SOUPS)*, July 2012.

[30] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, K. Bacon, K. How, and H. Strong.
Expandable grids for visualizing and authoring computer security policies.
In *CHI 2008: Conference on Human Factors in Computing Systems*, pages 1473–1482, 2008.

[31] R. Schlegel, A. Kapadia, and A. J. Lee.
Eyeing your exposure: quantifying and controlling information sharing for improved privacy.
In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, SOUPS '11, pages 14:1–14:14, New York, NY, USA, 2011. ACM.

[32] A. Shamir.
How to share a secret.
*Commun. ACM*, 22(11):612–613, 1979.

[33] R. Shokri, G. Theodorakopoulos, G. Danezis, J.-P. Hubaux, and J.-Y. Le Boudec.
Quantifying Location Privacy: The Case of Sporadic Location Exposure.
In *The 11th Privacy Enhancing Technologies Symposium (PETS)*, 2011.

[34] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux.

Quantifying Location Privacy.
In *2011 Ieee Symposium On Security And Privacy (Sp 2011)*, IEEE Symposium on Security and Privacy, pages 247–262. Ieee Computer Soc Press, Customer Service Center, Po Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, Ca 90720-1264 Usa, 2011.

[35] J. Y. Tsai, P. Kelley, P. Drielsma, L. F. Cranor, J. Hong, and N. Sadeh.
Who's viewed you?: the impact of feedback in a mobile location-sharing application.
In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 2003–2012, New York, NY, USA, 2009. ACM.

[36] Y. Wang, S. Komanduri, P. Leon, G. Norcie, A. Acquisti, and L. Cranor.
I regretted the minute i pressed share: A qualitative study of regrets on facebook.
In *Symposium on Usable Privacy and Security (SOUPS)*, 2011.

[37] F. L. Wong and F. Stajano.
Related Work in Multichannel Security Protocols.
*IEEE Pervasive Computing*, 6(4):31–39, 2007.

[38] www.MyVoucherCodes.co.uk.
Average brit has 476 facebook friends compared to 152 mobile phone contacts, 2011.