

ZeroSquare: A Privacy-Friendly Location Hub for Geosocial Applications

MoST 2013

Sarah Pidcock & Urs Hengartner

Geosocial Applications

foursquare [LOG IN](#)

facebook [Sign Up](#)

Email or Phone Password [Log In](#)

Keep me logged in [Forgot your password?](#)

Google Latitude
See where your friends are right now. [▶ Watch video](#)

Sharing
Tagging
Timeline
Graph Search
Location >
Apps
Ads
Credits

Share
Let people know.

Add a location

- Remember favorite locations
- Let friends know you are nearby
- Share what you are doing and where you are

AT&T LTE
Best N
SoHo, New
W Broadway
Noasser St
Green St
Leah Co
Happening Nearby
The Dutch is tr
9.5 American
Circolo is new
7.1 Italian 0.2
Recommendations
Lupa Ost
9.4 Italian
Spring S
8.8 Vege
Friends

San Pablo Richmond Pleasant Hill Concord
Walnut Creek Clayton
Berkeley Alameda Danville
Oakland San Ramon
Alameda San Leandro Dublin
San Francisco Bay Hayward Pleasanton
San Francisco
Hayward
Mountain View, CA
Max Braun
San Mateo Redwood City
Half Moon Bay
Mills
View Sunnyvale
Cupertino San
Saratoga Campbell
Los Gatos

BlackBerry
San Francisco
Green City
San Francisco
Friends list

9:47 PM
Friends
Google Latitude
Andrew Gilbert
San Francisco, CA
Friends list
Lower Heights
San Francisco
Google

Geosocial Apps Raise Privacy Concerns

Many geosocial apps that want to know our location and maybe other personal information

Each of them needs to be trusted to properly secure this information

This will become worse if existing services become location hubs and allow access to third-party apps

Foursquare Connect

foursquare

Featured Connected Apps

Connect these apps to your Foursquare account to get more out of your check-ins, like finding the healthiest dish to order or interesting people nearby.

[SEARCH](#)

**Location Editor**

Edit foursquare venues on the go

[CONNECT APP](#)

**Squareadar**

Find out when far away friends are finally nearby

[CONNECT APP](#)

**Foodspotting**

See what dishes foodspotters, friends and experts recommend wherever you go

[CONNECT APP](#)

**Hear I Am**

Meet similar music lovers on the go

[CONNECT APP](#)

**SoundTracking**

Share the soundtrack to your life

[CONNECT APP](#)

**Untappd**

It's time to start drinking socially. Discover new beers, bars and friends.

[CONNECT APP](#)

4

Research Challenge

Design of a **privacy-friendly location hub** that supports **many types** of geosocial applications

What does privacy-friendly mean?

Less privacy than application-specific privacy-enhancing protocols for geosocial applications

More privacy than today's widely deployed geosocial applications

Design Goals

Privacy based on **separation of information**

Support of **various** application use cases

Decoupling of data storage and social networking functionality

No significant **client-side** computation

Separation of Information

Each component of the location hub can learn **either a user's identity or her location, but not both**

Each component is run by a **different organization** (or kept in a different data silo if all run by the same organization)

ZeroSquare: Architecture

Geosocial apps require information about users and about locations

Therefore, we have **two databases**:

User Database with information about users

Location Database with information about locations

Optionally, for scalability or privacy reasons, a geosocial app comes with an additional component running in the cloud ("**cloudlet**")

User Database, Location Database and cloudlets are **honest-but-curious** and do not collude

ZeroSquare: Architecture

User Database

Stores information about users
Information is encrypted if sensitive

Location Database

Stores information about locations (e.g.,
reviews of a place or users at a location)
Information is encrypted if sensitive

Cloudlets

Retrieve information about users and locations
Need to be whitelisted by a user

Smart-
phones

Interact with user and location databases and
cloudlet to provide a service

Encryption of Sensitive Information

Anyone can retrieve information stored in User Database or Location Database

A user storing sensitive information in either database should encrypt it with a **key specific to the user and the type of information**

The user **hands out decryption keys** to friends (and cloudlets of apps that he/she uses)

Details of key management are out of scope

Decryption of Information stored in Location Database

Assume a ciphertext associated with a location (e.g., an encrypted review of a restaurant) stored in Location Database needs to be decrypted

How do we find the right decryption key?

Adding the identity of the creator to the ciphertext would violate separation of information goal

Trying every obtained decryption key would violate efficiency goal

Tags

The creator of a ciphertext stored in the Location Database attaches a **tag** to the ciphertext

A tag must not reveal any information about the creator to the Location Database

A tag must **never be reused**

Tags are generated by the User Database

The User Database allows entities that were **whitelisted** by the creator to map the tag back to the creator's identity

Example – Friend Proximity Detection (Option 1)

1. Alice is Bob's friend and has the decryption key for his location information
2. Bob stores his current location in the User Database in encrypted form
3. Alice retrieves Bob's encrypted location from the User Database, decrypts it, and checks whether Bob is nearby

Inefficient for many friends and Alice learns more information than necessary

Example – Friend Proximity Detection (Option 2)

1. Bob retrieves a tag from the User Database and stores it at his current location in the Location Database
2. Alice retrieves all tags currently at her location from the Location Database
3. If whitelisted by Bob, the User Database will allow Alice to map Bob's tag back to his identity

Inefficient

Example – Friend Proximity Detection (Option 3)

1. Same as option 2, except that list of tags at Alice's location (but not the location!) is given to clouldlet associated with the friend proximity detection app
2. If Bob has whitelisted this app, the cloudlet can map his tag back to his identity
3. Cloudlet checks whether Bob is a friend of Alice
4. If so, cloudlet notifies Alice of Bob's proximity

ZeroSquare: API

Defined an **API** for both the User Database and the Location Database

Used this API to **build various geosocial applications**
Friend locator, friend proximity detection, social recommendations, advertising service

API provided by User Database

Accessing information:

setAttribute(identity, attribute, value, ttl)

// value can be encrypted

value(s) ← **getAttribute**(identity, attribute)

Managing tags:

tag ← **createTag**(identity)

whitelist(identity, application)

identity ← **getIdentity**(tag, application)

API provided by Location Database

setAttribute(location, attribute, value, tag, ttl)

// e.g., attribute = "review"/value = <encrypted review> or attribute = "present"/value = true

value(s)/tag(s) ← **getAttribute**(location, attribute)

token ← **registerCallback**(location, attribute, ttl, handler)

// call handler whenever setAttribute() is invoked for given location and attribute

Example – Interest matching

Alice signs up with Match, an interest matching app, to get notified when she is at the same location as somebody else with matching interests (e.g., Bob)

Signup:

- 1) Alice and Bob give Match the decryption keys for their interests, as stored in the User Database
- 2) Alice @ User Database: **whitelist**(Alice, Match)
Bob @ User Database: **whitelist**(Bob, Match)

Example – Interest matching (cont.)

Alice and Bob are checking in at a location:

- 1) Alice @ User Database: $\text{Tag}_A \leftarrow \text{createTag}(\text{Alice})$
- 2) Alice @ Location Database: **setAttribute**
($\langle \text{Alice's location} \rangle$, "present", Tag_A)
- 3) Alice sends her encrypted location to Match
- 4) Match @ Location Database:
registerCallback($\langle \text{Alice's encrypted location} \rangle$,
"present", handler @ Match)

Bob performs the same steps

Example – Interest matching (cont.)

Bob checks in at Alice's location, and her callback gets invoked

- 1) Location Database invokes Match's handler and passes over set of tags at the location
- 2) For each received tag Tag_k , Match @ User Database:
 - 1) $\text{Id}_k \leftarrow \text{getIdentity}(\text{Tag}_k, \text{Match})$
 - 2) $\text{interests} \leftarrow \text{getAttribute}(\text{Id}_k, \text{"interests"})$
 - 3) Notify Alice of Id_k (Bob) if interests match

Match does not learn Alice and Bob's location

Implementation and Evaluation

ZeroSquare has been implemented with Android and a Python client as end-user platforms

Evaluated the performance of an interest-matching application

End-to-end performance of combined sign-up, check-in, and notification is 500 ms for Python client and 1.4 s for Android client (see paper for details)

Related Work

Lots of related work on retrieving PoIs in a privacy-friendly manner, where location of PoI is static and non-sensitive
Does not apply to geosocial apps

Storing encrypted user and location information in two different databases: **Puttaswamy et al.** (HotMobile 2010)
Targets friends only, not efficient

Double indirection between objects and their plaintext attributes: **Trust No One** (MobiHeld 2010), **Koi** (NSDI 2012)
Susceptible to traffic analysis and trace analysis attacks

Future Work

Ensure that location updates seen by Location Database remain **unlinkable**

Avoid information leak where by polling User Database, attacker can observe that a user has moved (but not to where)

Conclusions

Privacy-friendly architecture for geosocial apps based on **separating information between two non-colluding databases**, one for user-specific information, the other one for location-specific information

Databases provide **simple API** that is sufficient for building many of today's geosocial apps

Application-specific cloudlets for improved scalability and privacy