SamaritanCloud: Secure and Scalable Infrastructure for enabling Location-based Services

Abhishek Samanta Fangfei Zhou Ravi Sundaram
Northeastern University
Boston, MA 02115
Email: {samanta, youyou, koods}@ccs.neu.edu

***Abstract*—With the maturation of online social networks (OSNs), people have begun to form online communities and look for assistance at a particular place and time from people they only know virtually. However, seeking for such help on existing OSN infrastructures has some disadvantages including loss of privacy (in terms of both location as well as the nature of the help sought) and high response times. In this paper we propose SamaritanCloud, a scalable infrastructure that enables a group of mobile and geographically-dispersed personal computing devices to form a cloud for the purpose of privately sharing relevant locality-specific information. From a technical standpoint our main contribution is to show that only additive homorphic encryption is sufficient to compute nearness in a cryptographically secure and scalable way in a distributed setting. This allows us to harness the benefit of linear match time while guaranteeing the locational privacy of the clients. In terms of performance our system compares favorably with simpler publish/subscribe schemes that support only equality match. We demonstrate the practical feasibility of SamaritanCloud with an experimental evaluation.**

***Index Terms*—location-based service, mobile devices, privacy, homorphic encryption**

I. Introduction

A. Motivation

People often have the need for assistance from strangers in remote locations. Consider the following requests: please tell the marathon runner with bib #123 I will wait at the finish line; did I leave my keychain on campus? is there a brown puppy roaming in the playground? In this paper we propose, not just a new architecture, but, in fact, a new cloud based service - SamaritanCloud - the goal is to provide a way for people to connect with others (possibly strangers) in a remote location and obtain (physical) help from them. SamaritanCloud is deployed as cell phone application, users submit their requests to the cloud which coordinates users' requests and efficiently find possible candidates to respond to the request. Such a service will require efficient technical solutions to problems such as scalability, privacy, reputation etc to overcome the social barriers of soliciting help from strangers. We focus primarily on the technical aspects of scalability and privacy (matching people with strangers in a secure and private way) so that the need for help and the ability/desire to assist are disclosed safely.

Since the emergence of online social networks (OSNs), people have sought help from their social contacts. The most common method to seek for help on social network sites, e.g, Facebook, Twitter, or "strangers helping strangers" [1], is post - an user posts his/her question or request on his/her social network page or a relative group page and waits for response, which is very similar to subscribing to an email list and broadcast questions except exposing more privacy. This method is simple but suffers from three major drawbacks.

1) High/unpredictable response latency: Popular OSNs, e.g., Facebook, Twitter own worldwide users, who live in different locations and have different schedules. A post on a group page could not reach all the members in a short time and may be overwhelmed by other posts soon.

2) Limited range of request subjects: Groups on OSNs are typically centered around interests, occupations, genders, ages. So, it is hard to elicit response for time-restricted and location sensitive questions. Most online requests focus on recommendation, suggestions and people do not make off-line offer before they build trust on each other.

3) Privacy loss: Users requesting help on OSNs could end up exposing themselves to a large group, including friends they know in real life, as well as users, who are not willing/able to offer help. Unnecessary privacy leak may affect user's personal life and should be avoided.

B. Related Work

Geosocial networking [2]–[5] offers location based services (LBS) to users to interact relative to their locations. LBS typically is an information or entertainment application that is dependant on location of user. By these services users are offered possibilities to find other people, machines and location-sensitive resources. Some applications [6], [7] match users with locations of interest. These services are used for finding a physically proximal social contact or for crowd sourcing. Many of such public LBS provide no privacy while some offer limited protection on an opt-in or opt-out basis. Here we briefly categorize the different approaches to providing privacy, as well as the associated shortcomings.

1) Middleware: Geopriv [8] and LocServ [9] are policy-based privacy management approaches for secure location transfer that employ a trustworthy middleware mediating between location-based applications and location tracking servers. Geopriv [8] describes an approach to securely transferring location information and associated privacy data by creating "location objects" that encapsulate user location data and associated privacy requirements. Once "location generator" (e.g. user device) creates an "location object", it sends it to a middle server which forwards the object to "location recipient" based on different kinds of rules. LocServ [9] is a middleware service that lies between location-based applications and location tracking servers. It offers a general framework of components that allows users to apply general policies to control release of their location information. However, the practicality of such systems have yet to be determined, as the trustworthiness of middleware is hard to guarantee.

2) Dummies or Anonymity: Instead of sending out the exact location, a client sends multiple different locations to the server with only one of them being true [10]. The drawback of such schemes is that over the long run the server is able to figure out the client's location by comparing the intersection of uploaded locations. Some other schemes [11], [12] separate location information from other identifying information and report anonymous locations to the server. However, guaranteeing anonymous usage of location-based services requires that the precise location information transmitted by a client cannot be easily used to re-identify the subject.

3) Location hiding: Clients define sensitive locations where they do not want to be tracked, and the location-based application stops tracking when the user gets close to those areas [13].

4) Location perturbation: These schemes "blur" the exact location information to a spatial region [14]–[17] or send a proxy landmark instead [18] and hence, are not suitable for applications that require accurate locations.

The scheme presented in this paper uses *client-specific*, *personalized* and *global* blurs that are random elements in a finite field to guarantee perfect accuracy and cryptographic security; more on this in the sections to follow.

To complete our review of related work we briefly survey the literature on homomorphic encryption. Homomorphic encryption is a type of encryption scheme that provides ability to perform arithmatic operation on cipher text and get the encrypted result which is equivalent to the encryption of the result by applying the same arithmatic operation on the plaintext. The encryption systems of Goldwasser and Micali [19], and El Gamal [20] are known to support either addition or multiplication among encrypted cypher texts, but not both operation at the same time. In a breakthrough work, Gentry [21] constructed a fully homomorphic encryption scheme (FHE) capable of an arbitrary number of addition and multiplication on encrypted data. Fully homorphic encryption schemes are very powerful as it computes any function on an encrypted ciphertext. But, Lauter et al [22] showed that fully homomorphic encryption schemes till date are very resource consuming and are impractical to be used for most of practical purposes.

C. Our Contributions

To overcome the above mentioned drawbacks of LBS and OSNs, we propose SamaritanCloud, a location-based cyber-physical network allowing people to reach friends or strangers in desired locations for help. SamaritanCloud is easily deployed as a location-based cell phone application; the cloud coordinates client requests and efficiently finds candidates who are able/willing to offer help, even possibly physical help, in real-world life-threatening emergencies.

From a technical standpoint our primary contribution is to show how partially homomorphic encryption can be adapted to a distributed setting so as to guarantee cryptographic security without sacrificing efficiency gains. The resulting protocol finds near-neighbors in constant-time thus providing efficient and private match of help-seekers and help-givers.

SamaritanCloud preserves efficiency and privacy while calculating near neighbor, by employing different blurring techniques and partially homomorphic encryption.

II. System Design

A. System Model

SamaritanCloud consists of m servers and it serves n clients. The cloud is also composed of high speed connectors connecting these m servers with each other. Each of the servers and clients have unique identification which is called an id. Each server manages atleast one client and stores client infomations on its local disk. Servers also store a hashing function (h), that maps a client id to a server id. Each client has a mobile device with enough computation power to efficiently encrypt-decrypt using additive homomorphic functions. Clients are assumed to share a common key with which they exchange private messages that are denied to the cloud. SamaritanCloud is efficient in simple arithmatic computations (viz. addition, subtraction) and also broadcasting to clients. On the other hand, a client can efficiently talk to SamaritanCloud or a small group of other clients. SamaritanCloud is used to authenticate clients and to initiate distributed near-neighbor computation. When a client, sends a request to the SamaritanCloud, the request is only forwarded to the server managing that client. The managing server of a client is determined by hashing the client id to server id using h. Once the querying client knows the set of near neighbors, it sends the help request to them. Each client is associated with an username, password and profile. A client uses username and password for authentication. A profile is a collection of variables and their respective values (or range of values). Variables are drawn from a metric space and for the purposes of this paper are assumed to take real numbers. To obtain better scalability, SamaritanCloud delegates all computationally extensive operations, viz. distance computations, encryption-decryption process, to the clients.

B. Attack Model

There are u adversaries (unknown to SamaritanCloud) present in the network. These adversaries form a group and launch attacks to retrieve profile information of other good clients. Adversaries follow SamaritanCloud protocol to avoid detection and also exchange informations among themeselves to launch an attack. The adversaries try to extract partial or full profile information from blurred profile data of a client.

C. Definitions and Preliminaries

We present some definitions we will need later on

1) Distance: The distance between two profiles p_1, p_2, represented as d-dimensional vectors, is defined as:

$$\|p_1 - p_2\| = (\Sigma_i(|p_{i1} - p_{i2}|)^2)^{(1/2)},$$

where p_{i1} and p_{i2} are the values of the i^{th} dimension of p_1 and p_2, respectively.

2) Near-neighbor: Given a specific client c^q, a set of clients C, and a distance value $dist$, $c_i \in C, c^q \notin C$, is said to be a near-neighbor of c^q if the distance between p^q and p_i is at most $dist$, i.e. if

$$\|p^q - p_i\| \leq dist,$$

where p^q and p_i are profile data of c^q and c_i, respectively.

3) Blur: Blur is a random number which is used to blur data. e.g. data p is blurred with a random blur r as follows,

$$\beta_r^{\pm}(p) = (p \pm r)\ modP,$$

where $r \in [0, P)$ and P is a large public prime number.

4) Profile: Profile (p_i) of a client c_i, is defined as a collection of informations of c_i, viz, x and y coordinates of its current location, maximum amount of weight c_i can help with, area where c_i can help etc. A profile of a client is represented as a d-dimensional vector (in d-dimensional Euclidean space).

5) Tolerance-value: Tolerence value is the maximum distance between profile of a client c_j and profile requested by c^q, within which c_j receives help request from c^q.

TABLE I
TABLE OF NOTATION

Notation	Explanation
c_i	i^{th} client
c^q	A client which needs help
C	Set of clients
p_i	Profile data of c_i, represented as d-dimensional vector
p_{ij}	i^{th} dimension of p_j
p^q	Profile data requested by c^q, represented as d-dimensional vector
p_i^q	i^{th} dimension of p^q
$\beta_r(p)$	Profile data p blurred with r
$\xi_k(p)$	Encryption of data p using key k
$\xi_k^h(p)$	Encryption of data p using key k, using additive homomorphic encryption
$\delta_k^h(c)$	Decryption of cipher c using key k
sk	Secret key only known to clients
${}^iT$	i^{th} entry of a tuple T

The different kinds of communications between the cloud and a client are classified as follows,

Authentication: A client uses the assigned username and password to authenticate itself to SamaritanCloud.

ProfileUpdate: The client-side application periodically sends update messages to the cloud to update profile data.

HelpRequest: When a client looks for help, he/she sends a request to the cloud, which initiates distributed near-neighbor computation.

```
1:  procedure MATCH_PUB_SUB($\xi_{\xi_k(p_q)}(r)$)
2:      $S_c \leftarrow \phi$
3:      for all $c_i \in C$ do
4:          $\xi_k(p_i) \leftarrow fetch\_encrypted\_profile(c_i)$
5:                                              ▷ Encrypted profile of $c_i$
6:          if $\xi_{\xi_k(p_i)}(r) = \xi_{\xi_k(p_q)}(r)$ then
7:              $S_c \leftarrow S_c \cup c_i$
8:          end if
9:      end for
10:     Send2Client($c^q, S_c$)                ▷ Sends $S_c$ to $c^q$
11:     return
12: end procedure
```

Fig. 1. Procedure invoked by the cloud in publish/subscribe scheme after receiving a request to determine set of clients whose profile match requested profile.

```
1: procedure REQUEST_PUB_SUB($p^q$)
2:     $r \leftarrow fetch\_rand\_cloud()$
3:   ▷ Fetches random number from cloud to encrypt clients data.
4:     Send2Cloud($r, \xi_{\xi_k(p^q)}(r)$)
5:                                          ▷ Sends arguments to cloud
6:     return
7: end procedure
```

Fig. 2. Procedure invoked by a client in publish/subscribe scheme to request for help

Notification: Once the requesting client finds a set of clients suitable for servicing a request, it forwards the request encrypted with shared key (sk) to the set.

Follow-up: SamaritanCloud provides a simple client evaluation scheme by sending follow-up notification to the candidates who committed to offer help. We leave a more complete evaluation/reputation scheme as future work.

Our scheme builds on prior work on confidential publish/subscribe schemes. In the subsections to follow we briefly summarize these techniques.

D. A confidential publish/subscribe scheme and its issues

In the content-based publish/subscribe model the interests of subscribers are stored in a forwarding server which matches and routes relevant notifications to interested subscribers [23]. Procedures in Figs. 1, 2 and 3 are the basic primitives shown in [23].

[0]Texts after right arrowheads (▷) in pseudocodes are comments

```
1: procedure PROFILEUPDATE_PUB_SUB($p_i$)
2:     Send2Cloud($\xi_k(p_i)$)       ▷ Sends argument to cloud
3:         ▷ $k$ is the encryption key, shared among all clients
4:     return
5: end procedure
```

Fig. 3. Procedure invoked by a client in publish/subscribe scheme to update its profile to cloud

Observe that, in the above scheme, the server can match profiles across clients and across time so long as the random number used by the querying client is same as the random number used by the server. But, for the security of the scheme, it is necessary to change the random number often. But this change incurs large overhead on the system. The extra overhead is explained with the following scenario.
A client c^q wants to send a request to the cloud. So, it invokes procedure REQUEST_PUB_SUB, Fig.2. But, after c^q fetches the random number (line2 Fig.2), cloud updates it. When c^q sends the pair of random number and the encrypted value (line4 Fig.2) back to the cloud, it is rejected as the random number in cloud is changed. To avoid such scenarios cloud has to keep extra bookkeeping to make sure that, all the request-processes are completed before the random number is changed. For a large system this incurs large overhead.
Also, the request process in publish/subscribe is very resource consuming as clients have to perform two encryptions before sending a request to the cloud.
More importantly, the above scheme, without fully-homorphic crypto-system only computes equality match and not the closeness of two attributes. But, Lauter et al [22], have shown that fully-homorphic encryptions are highly resource consuming and are not practical in real life scenario.
Therefore, the above publish/subscribe algorithm is inadequate for our purposes.

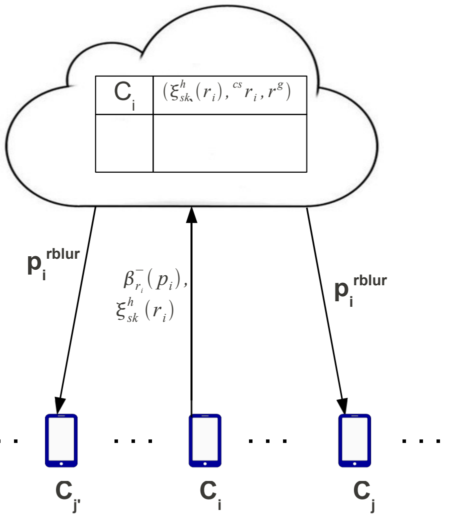
E. SamaritanCloud

In this section we propose the protocol of SamaritanCloud. Then we prove the correctness of our protocol. At the end we discuss about the security offered by SamaritanCloud.
Before delving into technical details, we first present short intuitive description of our protocol.

At the startup time, SamaritanCloud generates client specific blur for each client and a global blur, which are kept secret from clients.

When profile of a client changes, it blurs the changed profile with a random personalized blur and sends it to SamaritanCloud along with encrypted blur. Since, the profile data is blurred with a random blur and the random blur is encrypted, no profile information leaks from the blurred profile. SamaritanCloud reblurs the already blurred profile with client-specific and global blur of the client and distributes among k least loaded clients.

When a client wants help, he/she blurs its own profile with a personalized blur and sends to SamaritanCloud which in turn reblurs the already blurred profile and broadcasts to all clients. Along with these reblurred profiles, SamaritanCloud also broadcasts addition of client specific and encrypted personalized blurs. With these informations, a client computes the set of near-neighbors of requesting client. SamaritanCloud uses both client specific blur and global blur, because without these blurs under certain attack scenarios, an attacker can deterministically compute profile information of a requesting client.



C_i
$(\xi^h_{sk}(r_i), {}^{cs}r_i, r^g)$
p_i^{rblur}
$\beta^-_{r_i}(p_i)$, $\xi^h_{sk}(r_i)$
p_i^{rblur}
$C_{j'}$
C_i
C_j

Fig. 4. ProfileUpdate process in SamaritanCloud system

```
1: procedure INITIALIZE
2:     $r^g \leftarrow get\_rand()$                          ▷ Global blur
3:     Store($r^g$)
4:     ${}^{cs}r_i \leftarrow get\_rand(), \forall c_i \in C$        ▷ Client-specific blur
5:     Store($c_i, {}^{cs}r_i$), $\forall c_i \in C$
6:     ▷ Stores client-specific blur with corresponding client
7:     return
8: end procedure
```

Fig. 5. Procedure invoked at start of SamaritanCloud system

1) Protocol:

SamaritanCloud uses additive homomorphic crypto system to compute distance between profiles of two clients. It works in 3 phases as follows,

Initialization: In this phase, Fig.5, SamaritanCloud assignes client-specific blur to all the clients. A client-specific blur ${}^{cs}r_i$ is assigned to a client c_i. SamaritanCloud also generates a global blur r^g. Both client-specific and global blur are kept secret from clients. These are used by SamaritanCloud to blur client profiles.

ProfileUpdate: This phase, Fig.4, is invoked by a client c_i when its profile data p_i is changed.
In this phase, c_i updates SamaritanCloud with blurred profile data. SamaritanCloud broadcasts blurred p_i among set of clients C. The data transfer is thus composed of two subphases as follows,

a) Update: In *Update* phase, Fig.6, c_i blurs its profile data p_i by blurring each dimension seperately as follows,

$$\beta^-_{r_i}(p_{ji}) = (p_{ji} - r_i) \mod P, \forall j \in [1, d] \quad (1)$$
$$\beta^-_{r_i}(p_i) = (\beta^-_{r_i}(p_{1i}), \beta^-_{r_i}(p_{2i}), ..., \beta^-_{r_i}(p_{di}))$$

where, p_{ji} is j^{th} dimension of p_i, and r_i is the personalized blur used by c_i.
c_i also encrypts its personalized blur, using the additive-homomorphic encryption $\xi^h_{sk}(r_i)$, and sends the pair $(\xi^h_{sk}(r_i), \beta^-_{r_i}(p_i))$ to the cloud.

```
procedure UPDATE($p_i$)
    $r \leftarrow get\_rand()$    ▷ Personalized random blur
    $(p_{1i}, ..., p_{ji}, ..., p_{di}) \leftarrow p_i$
    $\beta_r^-(p_{ji}) \leftarrow (p_{ji} - r) \mod P, \forall j \in [1, d]$
    $\beta_r^-(p_i) \leftarrow (\beta_r^-(p_{1i}), ..., \beta_r^-(p_{ji}), ..., \beta_r^-(p_{di}))$
    $\xi_{sk}^h(r) \leftarrow (r \cdot sk) \mod P$
    Send2Cloud $(\xi_{sk}^h(r), \beta_r^-(p_i))$
                                    ▷ Sends the pair to SamaritanCloud
    return
end procedure
```

Fig. 6. Procedure invoked by a client to update its blurred profile to SamaritanCloud

b) Redistribution: SamaritanCloud invokes this phase, Fig.7, when it receives an update request from a client c_i. Client-specific blur ${}^{cs}r_i$, and the global blur r^g are used to blur already blurred profile of c_i. We call this process reblurring.

$$\begin{aligned} p_{ji}^{rblur} &= (\beta_{r_i}^-(p_{ji}) -^{cs} r_i + r^g) \mod P, \forall j \in [1, d] \\ &= (p_{ji} - r_i -^{cs} r_i + r^g) \mod P \qquad (2) \\ p_i^{rblur} &= (p_{1i}^{rblur}, p_{2i}^{rblur}, ..., p_{di}^{rblur}) \end{aligned}$$

SamaritanCloud distributes reblurred profile values computed in equation (2) along with the address of c_i among k least-loaded clients and saves the encrypted blur $\xi_{sk}^h(r_i)$.

HelpRequest: When a client (c^q) is in need of help, it invokes HelpRequest phase Fig.10. In this phase, c^q sends blurred requested-profile along with a tolerance value to SamaritanCloud. On reception of a request, SamaritanCloud broadcasts the blurred requested-profile to all clients and delegates the near-neighbor computation to clients. Clients compute distance between the requested profile and saved blurred-profiles and send back the result directly to c^q. Thus, this phase is composed of 3 subphases as follows,

c) Request: The requesting client c^q blurs the requested-profile p^q with a randomly generated personalized blur (r^q), Fig.8, by blurring each dimension of p^q seperately.

$$\begin{aligned} \beta_{r^q}^+(p_i^q) &= (p_i^q + r^q) \mod P, \forall i \in [1, d] \qquad (3) \\ \beta_{r^q}^+(p^q) &= (\beta_{r^q}^+(p_1^q), \beta_{r^q}^+(p_2^q), ..., \beta_{r^q}^+(p_d^q)) \end{aligned}$$

c^q encrypts its personalized blur and tolerance value (tol) and sends the tuple $(\xi_{sk}^h(r^q), \xi_{sk}^h(tol), \beta_{r^q}^+(p^q))$ to SamaritanCloud.

d) Redistribution: SamaritanCloud invokes this phase on reception of a request from c^q.

Without loss of generality, let us assume that a client c_i has blurred profile information of t other clients $c_j, \forall j \in [1, t]$. SamaritanCloud builds a set (S_i^q) of t 6-tuples (T_j^q) and sends it to c_i, Fig.9.
1st and 2nd entries of T_j^q are the address of c^q and c_j, respectively.

$$\begin{aligned} {}^1T_j^q &= c^q \\ {}^2T_j^q &= c_j \end{aligned}$$

3rd entry is built by reblurring the blurred profile of c^q, with the clients-specific blur assigned to c^q and the global blur, as

```
procedure REDIST_UP($(\xi_{sk}^h(r_i), \beta_r^-(p_i))$)
    ${}^{cs}r_i \leftarrow fetch\_cs\_blur(c_i)$
                        ▷ Fetches client-specific blur for $c_i$
    $r^g \leftarrow fetch\_global\_blur()$    ▷ Fetches global blur
    $(\beta_r^-(p_{ji}), ..., \beta_r^-(p_{ji})), ..., \beta_r^-(p_{di})) \leftarrow \beta_r^-(p_i))$
    $p_{ji}^{rblur} \leftarrow (\beta_r^-(p_i) - {}^{cs}r_i + r^g) \mod P, \forall j \in [1, d]$
    $p_i^{rblur} \leftarrow (p_{1i}^{rblur}, ..., p_{ji}^{rblur}, ..., p_{di}^{rblur})$
    $C_t \leftarrow get\_random\_clients(t)$
                        ▷ Randomly selectes set of $t$ clients
    Send2Clients$_{C_t}(c_i, p_i^{rblur})$
                        ▷ Sends pair of arguments to $C_t$
    Store$(c_i, \xi_{sk}^h(r_i))$
                        ▷ Stores encrypted personalized blur
    return
end procedure
```

Fig. 7. Procedure invoked by SamaritanCloud to redistribute reblurred profile of c_i

```
procedure REQUEST($p^q$)
    $r^q \leftarrow get\_rand()$
                        ▷ Randomly generates personalized blur
    $(p_1^q, ..., p_j^q, ..., p_d^q) \leftarrow p^q$
    $\beta_{r^q}^+(p_j^q) \leftarrow (p_j^q + r^q) \mod P, \forall j \in [1, d]$
    $\beta_{r^q}^+(p^q) \leftarrow (\beta_{r^q}^+(p_1^q), ..., \beta_{r^q}^+(p_j^q), ..., \beta_{r^q}^+(p_d^q))$
    $\xi_{sk}^h(r^q) \leftarrow (r^q \cdot sk) \mod P$
    $\xi_{sk}^h(tol) \leftarrow (tol \cdot sk) \mod P$
                        ▷ Encrypts tolerance value
    Send2Cloud$(\xi_{sk}^h(r^q), \xi_{sk}^h(tol), \beta_{r^q}^+(p^q))$
                        ▷ Sends arguments to SamaritanCloud
    return
end procedure
```

Fig. 8. Procedure invoked by a requesting client to compute set of near-neighbors

follows.

$$\begin{aligned} {}^3T_{lj}^q &= (\beta_{r^q}^+(p_l^q) + {}^{cs}r^q + r^g) \mod P, \forall j \in [1, t] \\ &= (p_l^q + r^q + {}^{cs}r^q + r^g) \mod P, \qquad (4) \end{aligned}$$

where ${}^3T_{lj}^q$ is l^{th} dimension of 3^{rd} entry of T_j^q, ${}^{cs}r^q$ is client specific blur of c^q and r^g is the global blur.
SamaritanCloud adds encrypted personalized random blur of c^q and c_j to build 4th entry of the tuple. This is where we use additive homorphic nature of encryption scheme.

$${}^4T_j^q = (\xi_{sk}^h(r^q) + \xi_{sk}^h(r_j)) \mod P \qquad (5)$$

5th entry is built by adding the client specific blur of c^q and c_j. Since, both the client-specific blurs are not known to clients, the individual value of either ${}^{cs}r^q$ or ${}^{cs}r_j$ is not leaked by this entry.

$${}^5T_j^q = ({}^{cs}r^q + {}^{cs}r_j) \mod P \qquad (6)$$

6th entry is the encrypted tolerance value sent by c^q. This tolerance value is used in the near-neighbor computation.

$${}^6T_j^q = \xi_{sk}^h(tol) \qquad (7)$$

```
1:  procedure REDIST_REQ($(\xi^h_{sk}(r^q), \xi^h_{sk}(tol), \beta^+_{r^q}(p^q))$)
2:      for all $c_i \in C$ do
3:          $r^g \leftarrow fetch\_global\_blur()$
4:                                                  ▷ Fetches global blur
5:          ${}^{cs}r^q \leftarrow fetch\_cs\_blur(c^q)$
6:                                  ▷ Fetches client-specific blur for $c_i$
7:          $p^{rblur}_j \leftarrow (\beta^+_{r^q}(p^q_j) + {}^{cs}r^q + r^g) \mod P, \forall j \in [1, d]$
8:          $p^{rblur} \leftarrow (p^{rblur}_1, ..., p^{rblur}_j, ..., p^{rblur}_d)$
9:          $C_i \leftarrow fetch\_clients\_in(c_i)$
10:         ▷ Set of clients whose blurred profile is stored by $c_i$
11:         $S^q_i \leftarrow \phi$
12:         for all $c_j \in C_i$ do
13:             ${}^1T^q_j \leftarrow c^q$
14:             ${}^2T^q_j \leftarrow c_j$
15:             ${}^3T^q_j \leftarrow p^{rblur}$
16:             $\xi^h_{sk}(r_j) \leftarrow fetch\_encrypted\_blur(c_j)$
17:                                 ▷ Encrypted personalized blur of $c_j$
18:             ${}^4T^q_j \leftarrow (\xi^h_{sk}(r^q) + \xi^h_{sk}(r_j)) \mod P$
19:             ${}^{cs}r_j \leftarrow fetch\_cs\_blur(c_j)$
20:                                          ▷ Client specific blur of $c_j$
21:             ${}^5T^q_j \leftarrow ({}^{cs}r^q + {}^{cs}r_j) \mod P$
22:             ${}^6T^q_j \leftarrow \xi^h_{sk}(tol)$
23:             $T^q_j \leftarrow ({}^1T^q_j, {}^2T^q_j, {}^3T^q_j, {}^4T^q_j, {}^5T^q_j, {}^6T^q_j)$
24:             $S^q_i \leftarrow S_i \cup T^q_j$
25:         end for
26:         Send2Client($c_i, S^q_i$)             ▷ Sends $S^q_i$ to client $c_i$
27:         return
28:     end for
29: end procedure
```

Fig. 9. Procedure invoked by SamaritanCloud to delegate nearness computation to clients

e) Distance-Computation: In this phase, Fig.11, clients compute distance between the blurred requested-profile and saved blurred profile data of other clients.
Let us assume that after redistribution phase, a client c_i receives a set S^q_i. Let us also assume that c_i has blurred profile informations of t clients, $c_j, \forall j \in [1, t]$. Let $T^i_j \in S^q_i$ be the tuple corresponding to c_j and p^{rblur}_j be the reblurred profile of c_j.
Since, c_i has the shared secret sk, it decrypts the encrypted blur and tolerance value in T^i_j. c_i sends clients whose reblurred profile satisfies the following condition to c^q.

$$(\sum_{l=1}^{d}({}^3T^i_{lj} - p^{rblur}_j - \delta^h_{sk}({}^4T^i_{lj}) -{}^5 T^i_{lj})^2)^{1/2} < \delta^h_{sk}({}^6T^i_{lj}), \quad (8)$$

where $\delta^h_{sk}(\cdot)$ decrypts given cipher data using key sk.

2) Correctness: After stating the newly proposed scheme, here we prove that, the scheme calculates distance correctly between requested profile data c^q and blurred profile of a candidate client c_j. The fact that blurring according to our proposed scheme preserves distance is captured by the following lemma.

Lemma 1. *Given, T^q_j is the tuple corresponding to c_j and*

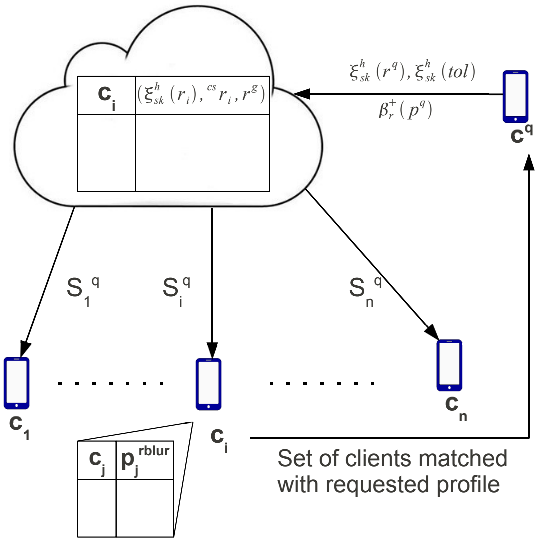

c_i
$(\xi^h_{sk}(r_i), {}^{cs}r_i, r^g)$
$\xi^h_{sk}(r^q), \xi^h_{sk}(tol)$
$\beta^+_r(p^q)$
c^q
S_1^q
S_i^q
S_n^q
c_1
c_i
c_n
c_j
p_j^{rblur}
Set of clients matched with requested profile

Fig. 10. Request process in SamaritanCloud system

p^{rblur}_j is the blurred profile of c_j

$$(\sum_{i=1}^{d}({}^3T^q_{ij} - p^{rblur}_j - \delta^h_{sk}({}^4T^q_{ij}) -{}^5 T^q_{ij})^2)^{1/2} = \|p^q - p_j\|$$

Proof: As described in ProfileUpdate protocol, blurred profile stored on SamaritanCloud is calculated using equation (2).

$$p^{rblur}_{ij} = (p_{ij} - r_j - {}^{cs}r_j + r^g) \mod P \quad (9)$$

Since, $\xi^h_{sk}(\cdot)$ is an additive homorphic encryption, from equation (5),

$$\begin{aligned} {}^4T^q_j &= (\xi^h_{sk}(r^q) + \xi^h_{sk}(r_j)) \mod P \\ &= \xi^h_{sk}(r^q + r_j) \mod P \end{aligned}$$

So,
$$\begin{aligned} \delta^h_{sk}({}^4T^q_j) &= \delta^h_{sk}(\xi^h_{sk}(r^q + r_j) \mod P) \\ &= (r^q + r_j) \mod P \end{aligned} \quad (10)$$

So, combining equations (4), (6), (9), and (10),

$$\begin{aligned} &({}^3T^q_{ij} - p^{rblur}_j - \delta^h_{sk}({}^4T^q_{ij}) -{}^5 T^q_{ij}) \\ &= ((p^q_i + r^q + {}^{cs}r^q + r^g) \mod P - \\ &\quad (p_{ij} - r_j - {}^{cs}r_j + r^g) \mod P - \\ &\quad (r^q + r_j) \mod P - ({}^{cs}r^q + {}^{cs}r_j) \mod P) \\ &= (p^q_i - p_{ij}) \mod P \end{aligned}$$

Thus, according to the definition,

$$\begin{aligned} &(\sum_{i=1}^{d}({}^3T^q_{ij} - p^{rblur}_j - \delta^h_{sk}({}^4T^q_{ij}) -{}^5 T^q_{ij})^2)^{1/2} \\ &= (\sum_{i=1}^{d}((p^q_i - p_{ij})^2)^{1/2}) \mod P \\ &= \|p^q - p_j\| \end{aligned}$$

■

```
procedure DISTANCE_COMP($S$)
    $S_c = \phi$        ▷ Set of clients matching requested profile
    for all $T \in S$ do
        $(c^q, c_j, {}^3T_j^q, {}^4T_j^q, {}^5T_j^q, {}^6T_j^q) \leftarrow T$
        $p_j^{rblur} \leftarrow fetch\_blur\_profile(c_j)$
                                        ▷ Blurred profile of $c_j$
        $\|p^q - p_j\| \leftarrow$
                $(\sum_{i=1}^{d}({}^3T_j^q - p_j^{rblur} - \delta_{sk}^h({}^4T_j^q) - {}^5T_j^q)^2)^{1/2}$
                        ▷ Distance between profiles of $c^q$ and $c_j$
        if $\|p^q - p_i\| < \delta_{sk}^h({}^6T_j^q)$ then
            $\xi_{sk}^h(c_j) \leftarrow (c_j \cdot sk)\ modP$
            $S_c \leftarrow S_c \cup \xi_{sk}^h(c_j)$
        end if
        Send2Client($c^q, S_c$)
                        ▷ Sends the set of near-neighbors to $c^q$
    end for
    return
end procedure
```

Fig. 11. Procedure invoked by clients to compute near-neighbor of requested profile

3) Security: A client c_i has access to reblurred profile data of set of other t clients $c_j, j \in [1, t]$. The client c_i does not know profile data of any of c_j clients, as all the profile data are blurred with random blurs unknown to c_i. Attacks on SamaritanCloud are averted by the use of different types of blurs. Here we state the effect of different blurs on the security of the overall SamaritanCloud system,

a) Personalized blur: Personalized blur is a random number used by every client to blur their original profile information. Since, this blur is randomly selected by clients, SamaritanCloud cannot gain knowledge from blurred profile about the profile data of a client.

b) Client-specific and global blur: Both client-specific and global blurs are randomly generated by SamaritanCloud. These blurs are used by the cloud to reblur profile of a client. Clients do not have access to these blurs. Global blur along with client-specific blur are used to protect profile data from distributed attacks by a group of adversaries.

We measure security of the proposed protocol with following metric,

Probability of Information Leak per Comparison (PILC): PILC is defined as the probability of an advesary successfully computing the profile data of a client by single comparison between reblurred profile of the client and that of an adversary.

Now, we state different kinds of possible attacks as follows,

External attack: In external attack, attackers know protocol of SamaritanCloud, but do not know the shared secret key(sk).

Every profile data is blurred with above mentioned blurs. Since, blurs are random numbers, it is impossible for external attackers, without the knowledge of blurs used, to extract profile information from the blurred profile data. Also, clients send personalized blurs to SamaritanCloud encrypted with a shared key. The shared key is known only to clients. Thus, blurred profile together with encrypted blurs avert external attacks.

Internal attack: For internal attack, the assumptions are much more strict than external attack. Here, attackers know both the protocol and the shared secret key(sk).

According to our proposed system, a client has access to reblurred profile of t other clients. Since, for internal attacks, attackers behave as clients, an adversary a_i has access to blurred profile data of t other clients $c_i, i \in [1, t]$. Some of these t clients are adversaries. There are two types of internal attacks as follows,

- *Active attack:* An adversary sends a help request to SamaritanCloud. The intention of the help request is to assist other advesaries to compute random blurs and thereby assisting them to successfully retrieve profile information of a good client. We know that, an adversary a_i has profile information of a client c_i blurred with personalized blur, client-specific blur of c_i and the global blur, equation (2).
  When SamaritanCloud receives the fake help request, it broadcasts following three informations to all clients,
  - reblurred requested-profile data of the querying adversary (a^q) to all other clients. Thus an adversary (a_i) receives requested profile blurred with personalized blur, client-specific blur of the querying adversary and the global blur r^g, equation (4). Assuming a^q informs a_i about personalized blur (r^q), and requested profile (p^q), a_i computes $({}^{cs}r^q + r^g) \mod P$ successfully. Profile data of a client c_i is blurred with its own client-specific (${}^{cs}r_i$) and the global blur. So, a_i can successfully compute blurred profile $(p_{ji} - r_i) \mod P$ of a client c_i, if ${}^{cs}r^q = {}^{cs}r_i$, equation (2). Since, all the blurs are selected randomly from $[0, P)$,
  $$Pr[a_i\ knows\ (p_{ji} - r_i) \mod P] = \frac{1}{P}$$
  - the addition of client-specific blur of a^q and c_i as in equation (6). Since, both the blurs are random numbers unknown to clients, this addition does not reveal individual value of each blur.
  - encrypted value of the addition of the personalized blur of a^q and c_i as shown in equation (5). a_i knows the personlaized blur of a^q, as all the advesaries share information. Thus, a_i computes $r_i \mod P$ successfully.

  Thus,
  $$PILC = \frac{1}{P} \tag{11}$$
- *Passive Attack:* An adversary (a_i) can launch a passive attack if it has access to reblurred profile of another adversary a_j, equation (2). Let us assume that a_i knows the reblurred profile of a client c_i. Since, all adversaries share informations, a_i knows the blurred profile of a_j. If client-specific blur of a_j is same as that of c_i, a_i can compute blurred profile of c_i ($(p_{ji} - r_i) \mod P$), equation (2). Since, reblurred profile of a client is randomly distributed

among t clients with least number of reblurred profile of other clients,

$$Pr[a_i\ has\ reblurred\ profile\ of\ a_j] \\ = 1 - (1 - \frac{u-1}{n})^t \\ \approx \frac{(u-1)\cdot loglogn}{n \cdot logk}, [Since,\ t = \frac{loglogn}{logk}\ [24]]$$

Since, personalized and client-specific blurs are randomly selected from $[0, P)$,

$$Pr[a_j\ knows\ client\ specific\ blur\ of\ c_i] = \frac{1}{P}$$

$$Pr[a_i\ knows\ personalized\ blur\ of\ c_i] = \frac{1}{P}$$

So, $$PILC = \frac{(u-1)\cdot loglogn}{n \cdot logk} \cdot (\frac{1}{P})^2 \tag{12}$$

Since, P is chosen to be a large prime number, probability of a successful attack is small both in active (equation (11)) and passive (equation (12)) internal attacks.

III. Experimental Evaluation

A. Implementation

The implementation of SamaritanCloud includes two parts — client side application and the cloud. We implemented the client side application with a prototype mobile application running on iOS 5.0, allowing users to,

- log in or register with unique username and password.
- choose time interval to update profile; by setting a fixed update time interval, exact profile change time is not exposed to the cloud.
- input request location (either latitude and longitude coordinates or zipcode) and request content.
- get notification when the client is close to a requested location; respond with willingness to offer help.
- get notification if anyone offers help to the client's request.
- get follow-up questionnaire of (1) "did you offer the help that you committed to?" if the client was a candidate, (2) "was the request resolved?" if the client was the requester.

We implemented the server using the Python Twisted library. The major concerns of SamaritanCloud performance are client application battery consumption and server scalability. We examine those two concerns with experimental results.

B. Mobile application

We first examine SamaritanCloud on mobile devices. Our application registers itself as a background VoIP service allowing it to maintain a persistent connection with the server, periodically fetch location information, even if the application is running in background. Our application utilizes *Core Location Framework* to get physical location. Location coordinates can be obtained through standard location service, and significant-change location service. With standard location service, location data is gathered with either cell triangulation, Wi-Fi hotspots or GPS satellites, depending on required accuracy. As our application periodically updates location to the server, it saves battery life by enabling the standard location service as per the interval defined by clients. With significant-change location service, a location update is triggered by the OS only if a device moves from one cell tower to another, thus providing only coarse location update. There is an accuracy-power tradeoff – GPS location update is accurate but power inefficient while significant-change location service is less accurate and low on power usage. To test the worst possible impact of our application on battery life, we run the applications always in foreground with intensive update (once per minute) via both 3G and WiFi. In each update, the mobile application fetches location coordinates (lat, lon), computes $((lat + r) \mod P, (lon + r) \mod P)$ (P is a prime with 64 bits, r is randomly chosen between [0, P)), sends the result together with $\xi^h_{sk}(r)$ to the server. The encryption function used is as follows,

$$\xi^h_{sk}(r) = (r \cdot sk) \mod P \tag{13}$$

where, sk is a 512bit key. The results in Table II show that the impact on battery life is acceptable (in real world deployment the application would run in background with infrequent location updates).

TABLE II
Impact of SamaritanCloud iOS application on battery life.

network	standard (GPS)	standard (WiFi/cellular)	significant-change location service
3G	10h	12h 8mins	12h 10mins
WiFi	11h 14mins	16h 14mins	16h

1) Key distrubution: Each client-side application is provided with a long term 1024bit secret key (lk). This key is used to encrypt the secret key sk, equation (13). The shared secret key, sk, is changed every one hour. At the start of each hour, SamaritanCloud randomly selects a client. The selected client, randomly generates the shared secret key, encrypts it with lk as follows, and sends it to SamaritanCloud

$$\xi_{lk}(sk) = (sk \cdot lk) \mod P$$

SamaritanCloud broadcasts the encrypted shared secret, to all clients.

C. Server scalability

For server side performance we are primarily interested in understanding the rate of *requests* that a single server can handle, as this serves as the dominating factor controlling the number of online users that the server can support.

In $ProfileUpdate$ phase, a client sends its blurred profile and SamaritanCloud distributes the blurred profile among t clients having least number of profiles of other clients. Since, SamaritanCloud can distribute in parallel, the runtime of $ProfileUpdate$ is $O(1)$.

In $HelpRequest$ phase, the requesting client sends blurred requested profile along with encrypted tolerance value to the cloud. SamaritanCloud broadcasts these informations among all available clients. Clients compute the distance between requested profile and other stored profile data. Broadcast to clients take $O(1)$ time. Time taken to compute distance between a pair is $O(d)$, because a profile has d dimensions. Since, each client has k number of profile data of other clients, it takes $O(d \cdot k)$ time for a client to determine set of near-neighbors. Since, blurred profile data of a client is distributed among t clients having least number of profile data of other clients, $k = \frac{loglogn}{logt}$ [24].

For experimental purpose we consider plain-text mode, where

$ProfileUpdate$	$HelpRequest$
$O(1)$	$O(d \cdot \frac{loglogn}{logt})$

TABLE III
RUNTIME OF DIFFERENT PHASES OF SAMARITANCLOUD

no profile information is blurred and all communications are un-encrypted. We compared latency of a help request in plain-text mode with that of the proposed protocol.

Latency, Fig.12, is measured starting from the moment the requester sends out blurred attributes to the time he gets response from the server. We ran our server implementation on a machine with 2.5 GHz Intel Core i5 and 4GB memory. To minimize the affect of Internet speed, we simulated clients on another machine on the same LAN. The main takeaways

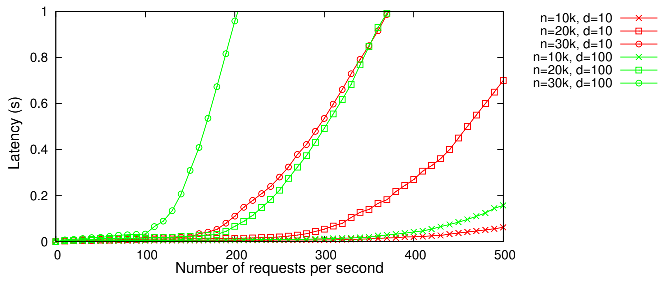


Fig. 12. Matching time measurement: plain-text(red) vs. encrypted message(green)

are that using plain-text messages has the benefit of fast query processing but leaks client privacy to the server; communication with encrypted messages preserves client privacy, while the query processing is expensive. When using plain-text messages, query latency nearly remains the same when request rate increases; server scalability depends on the number of live TCP connections (open files) allowed on the machine. In the normal encrypted mode, for n=20k query processing is not fast enough when request rate exceeds 200/second. To prevent congestion each client can send one valid request only every 5 minutes. With this request rate, the server can handle 30,000 live clients simultaneously.

IV. CONCLUSION

To the best of our knowledge, SamaritanCloud is the first of its kind in using location-based social networking system to enable people to *physically* help each other. In our system, when an user needs help at a particular location, the request is sent to the server and the users near the area of interest are looked up efficiently using distributed additive homomorphic encryption. To hinder external and internal security attacks SamaritanCloud uses randomly generated blurs. We have implemented a SamaritanCloud mobile application for iOS 5.0. The application fetches user's location with the help of one of cell triangulation, Wi-Fi hotspot or GPS. Because of the limited battery-life and processing power of smartphones, the mobile application allows users to manually select the frequency of location update and the level of security that they desire. Our SamaritanCloud system, opens up an entirely new approach to enable people to benefit from location-based social networks.

REFERENCES

[1] "Strangers helping strangers," http://www.facebook.com/SHStrangers.
[2] "Yelp," http://www.yelp.com/.
[3] "Facebook places," http://www.facebook.com/facebookplaces.
[4] "Gowalla," http://gowalla.com/.
[5] "Foursquare," https://foursquare.com/.
[6] D. Stackpole, "Dynamic geosocial networking," Patent US 2008/0 140 650 A1, 06 12, 2008.
[7] Q. Huang and Y. Liu, "On geo-social network services," *Geoinformatics*, 2009.
[8] A. Cooper and T. Hardie, "Geopriv: creating building blocks for managing location privacy on the internet," *IETF Journal*, 2002.
[9] G. Myles, A. Friday, and N. Davies, "Preserving privacy in environments with location-based applications," *Pervasive Computing, IEEE*, 2003.
[10] H. Kido, Y. Yanagisawa, and T. Satoh, "An anonymous communication technique using dummies for location-based services," in *Proceedings of IEEE International Conference on Pervasive Service*, 2005.
[11] A. R. Beresford and F. Stajano, "Location privacy in pervasive computing," *IEEE Pervasive Computing*, 2003.
[12] K. P. Tang, P. Keyani, J. Fogarty, and J. I. Hong, "Putting people in their place: An anonymous and privacy-sensitive approach to collecting sensed data in location-based applications," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 2006.
[13] M. Gruteser and X. Liu, "Protecting privacy in continuous location-tracking applications," *IEEE security and privacy*, 2004.
[14] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," *Pervasive*, 2005.
[15] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloacking," in *Proceedings of the International Conference on MobiSys*, 2003.
[16] B. Gedik and L. Liu, "A customizable k-anonymity model for protecting location privacy," in *Proceeding of the International Conference on Distributed Computing Systems*, 2005.
[17] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: a privacy-aware location-based database server," *IEEE 23rd International Conference on Data Engineering*, 2007.
[18] J. I. Hong and J. A. Landay, "An architecture for privacy-sensitive ubiquitous computing," in *Proceedings of the International Conference on Mobile Systems*, 2004.
[19] S. Goldwasser and S. Micali, "Probabilistic encryption & how to play mental poker keeping secret all partial information," in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, ser. STOC '82. ACM, 1982.
[20] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *Information Theory, IEEE Transactions on*, vol. 31, no. 4, pp. 469 – 472, jul 1985.
[21] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09. ACM, 2009.
[22] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, ser. CCSW '11. ACM, 2011.
[23] C. Raiciu and D. S. Rosenblum, "Enabling confidentiality in content-based publish/subscribe infrastructures," *Securecomm and Workshops*, 2006.
[24] R. S. Michael Mitzenmacher, Andrea W. Richa, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*. Kluwer, 2000, pp. 255–312.