SHARP: Private Proximity Test and Secure Handshake with Cheat-Proof Location Tags

Yao Zheng[1], Ming Li[2], Wenjing Lou[1], and Y.Thomas Hou[1]

[1] Virginia Polytechnic Institute and State University
{zhengyao,wjlou,thou}@vt.edu
[2] Utah State University
{ming.li}@usu.edu

Abstract. A location proximity test service allows mobile users to determine whether they are in close proximity to each other, and has found numerous applications in mobile social networks. Unfortunately, existing solutions usually reveal much of users' private location information during proximity test. They are also vulnerable to location cheating where an attacker reports false locations to gain advantage. Moreover, the initial trust establishment among unfamiliar users in large scale mobile social networks has been a challenging task. In this paper, we propose a novel scheme that enables a user to perform (1) privacy-preserving proximity test without revealing her actual location to the server or other users not within the proximity, and (2) secure handshake that establishes secure communications among stranger users within the proximity who do not have pre-shared secret. The proposed scheme is based on a novel concept, i.e. location tags, and we put forward a location tag construction method using environmental signals that provides location unforgeability. Bloom filters are used to represent the location tags efficiently and a fuzzy extractor is exploited to extract shared secrets between matching location tags. Our solution also allows users to tune their desired location privacy level and range of proximity. We conduct extensive analysis and real experiments to demonstrate the feasibility, security, and efficiency of our scheme.

1 Introduction

The proliferation of smartphones has given rise to location-based service (LBS), which has drawn considerable research attention in recent years. The key enabler of LBS is the availability of user locations, which can be easily measured and reported by a smartphone today. With LBS, users report their locations in real-time to a location server, which allows users to ubiquitously query places of interest around them, or test if their friends are within certain physical proximity. Especially, the latter is called "proximity test" [17] and has found numerous mobile applications, for example, to locate nearby friends (e.g., in a mobile social network [8]), or in an emergency situation to find nearby medical personnel (e.g., in mobile healthcare [9, 10]), only to name a few. The former is representative

for proximity test between friends, while the latter is an example of proximity test between strangers, who may not share any secret a prior.

Similar to many LBS services, there are many security and privacy concerns associated with proximity test that may prevent its widespread adoption. One of the major security concerns is that the reported locations could be easily forged by malicious users in order to exploit the benefits of proximity test service. There are many incentives for users to not report their locations truthfully. For example, in [7], a location cheating attack has been discovered in which the attacker reports false locations to gain revenue by acquiring shopping coupons. In addition, a curious user may try to profile other users' locations by setting hers to any desired place. Thus, it is essential to provide *location unforgeability* in proximity match, so as to ensure the social welfare of LBS. On the other hand, the *location privacy* is also an important concern for common users. The primary reason is that the location servers are often operated by third-party service provider such as a cloud platform, which tends to be not fully trusted by people since the location data could be leaked to the server or outsider attackers [23]. Meanwhile, users also do not want to simply let all her friends or even strangers in the system know about her location and track her down.

To design a *privacy-preserving* proximity test scheme that is also *cheat-proof* involves several challenges. First and foremost, given the mobile and distributed nature of LBS users, how can we make sure that a user's reported location is truthful without involving a trusted authority? Some researchers suggest a distributed proof approach using presence evidences from peer devices [25]. However, the proof generation involves the use of digital signature which further requires a public-key infrastructure (PKI). This would require significant modifications to the existing security architecture. In addition, the traditional cryptography-based methods do not guard against stolen/compromised keys or credentials. Ideally, each device should extract unforgeable location tags relying on its own. Second, shared keys are usually required for preserving privacy during proximity test and secure communications between matched users. However, the initial trust establishment among users in a large-scale mobile social network remains a difficult task, simply because managing shared keys with everyone else is not scalable without a trusted authority. Most existing solutions to date have relied on a-priori shared secrets between each pair of users [17, 21], which severely limits their applicability and scalability. Finally, efficiency and usability need to be achieved simultaneously. To achieve strong privacy guarantee, previous protocols either rely on computational intensive cryptographic primitives, or do so at a cost of high communication overhead.

In this paper, we propose a novel proximity test scheme that is secure against location cheating, and also performs secure handshakes between matched users to secure their subsequent communication without relying on pre-shared secrets. We focus on a general one-to-many proximity match setting, that is, user Alice can find out from a group of users the one(s) that are within certain proximity to her with the help of a semi-trusted server. In order to defeat location cheating, we propose a novel form of location representation – *spatial-temporal location*

tag that is constructed from wireless signals captured in a device's surrounding environment, such as WiFi and cell tower signals. An attacker cannot forge a location tag if she is not at the corresponding location and time, due to the high freshness (entropy) and spatial variety of environmental signals. Our proposed privacy-preserving proximity test protocol is then based on the location tags. We exploit *fuzzy extractor* [3], a lightweight crypto primitive, to extract secret keys automatically between users within certain proximity, while ensuring that a user's location is revealed to neither the server nor users not within proximity. We also make use of *bloom filter* to efficiently represent users' location tags.

1.1 Our Contributions

The main contributions of this paper are as follows.

(1) We propose a novel form of user location representation – spatial-temporal location tag, to defeat location cheating attacks in LBS. We demonstrate our concept using collected real-world WiFi and cellular signal traces, and employ entropy analysis to show the feasibility of generating unforgeable location tags in practice.

(2) We propose a novel private proximity test scheme based on spatial-temporal location tags, which performs proximity test and establishes secure handshake between one user and many others at the same time. We uniquely combine bloom filter and fuzzy extractor to meet the stringent privacy and efficiency requirements. Our scheme also supports user-defined privacy level, and avoids the complexity of key management among users as it does not rely on prior-shared secrets.

(3) We carry out both thorough security analysis and performance evaluation. We first quantitatively characterize the security level of our protocol using entropy analysis. Then, using a proof-of-concept implementation, we study the system functionality and overhead, and show its superiority over existing protocols in efficiency. To the best of our knowledge, this is the first work that systematically studies unforgeable location tag and its use in location-based services.

1.2 Related Work

For privacy in location-based services, most previous works have been focusing on privacy in location queries, i.e., a model in which users report their "encrypted" location data to a central database server to perform range or k-nearest-neighbor (kNN)) queries [15, 22, 6, 2]. Note that in this model the database stored in the server is assumed to be public. In contrast, the recently emerged *proximity test* is a different model where location-based matching is done only between users, while the users' locations are private information. In this paper we focus on proximity test.

Proximity Test: Proximity test is a special form of location sharing [23], where the information being shared is whether or not two users are within a certain range or in the same geographic region. The main privacy concern in proximity test is that user's actual location may be involuntarily revealed to

either the server or other users. To this end, a privacy-preserving proximity test solution is proposed in [21], using a grid-based encryption algorithm. In [14], Mascetti et. al. proposed proximity detection schemes based on service provider (SP) filtering, in which privacy protection is achieved by user-chosen location representation that controls its granularity. However, their protocol leaks coarse-grained location information to the server. In [17], Narayanan et. al. proposed a suite of private proximity test protocols. The possibility of constructing location tags from environmental signals was noted; however, their protocols either require pre-shared secret key between users, or is not scalable and efficient enough to handle one-to-many proximity test as studied in our paper. Another proximity test scheme was proposed in [20], where users can also control their privacy levels via leveled publishing. The protocol is based on keyed hashing which suffers from dictionary attack. In [11], Lin et. al. proposed a proximity test scheme by applying shingling technique [1] on GSM cellular message. However, they did not thoroughly analyze its security. In this paper, we carry out a systematic study of unforgeable location tags and its use in proximity test, and formally analyze the security using entropy theory.

Private Matching: Our proposed scheme constructs location tags and takes the location tags as the inputs to private matching scheme to realize proximity test. Different location tag construction methods will yield different types of location tags with different data structure representation, which in turn demand different secure matching algorithms. Secure inner product computation has been proposed to compute the number of matching keywords between two binary-valued vector inputs, where each bit in the vector represents the presence or absence of a keyword [24]. Secure multi-party computation (SMC) techniques have also been used in private matching. For example, in [5], Freeman et. al. proposed a private set intersection protocol using homomorphic encryption, where the inputs to be matched are two sets of elements. In this paper, we are matching two location tags, which are environmental signals represented using bloom filter and further coded using BCH coding. The method used to realize the private matching is also very different from previous known private matching methods. Essentially our matching method is based on polynomial reconstruction. Compared to previous private matching algorithm, our scheme is more efficient because it does not involve any public key cryptography operations.

2 System Model and Design Challenges

2.1 System Overview

Our system model consists of two types of entities: a *server* and a large number of *users*. Users are subscribers of the proximity test service provided by the server. For convenience, we use *Alice* to refer to the user who initiates the proximity test, and *Bob*/*Charlie*/*David et al* to refer to the testees upon Alice's request. The centralized online server that provides the service is only responsible for assisting participants relay messages. The selection of the testee group is based on certain criteria specified in each test request. At the end of the proximity test, the testee(s) that are within the proximity of Alice will establish a secret key

with Alice, while the testees faraway will learn nothing about Alice's location. The server also remains oblivious to the result of the proximity test.

Our security goal is to prevent location forgery from all users and the privacy goal is to prevent unnecessary leakage of users' location information against both other users and the server.

2.2 Design Challenges and Goals

We noticed that proximity test between strangers is usually one-to-many. Consider the following example. A patient in an emergency situation may only wish to grant nearby emergency medical technicians (EMT) access to her personal health data on her phone. Since the patient can not specify which EMT she wants to test, she can only select a group of EMTs based on certain searching criteria, e.g. EMTs from organization A. Previous client-server based solution [17] becomes inefficient in such circumstances because the test has to be done one-to-one. To cope with this problem, we choose a broadcast system model since it allows non-interactive proximity testing [17] while using less bandwidth than client-server model. Some particular challenges and our design goals based on the broadcast system model are as follows.

Distance-Bounded Key Establishment: The main motivation of our study is to address the situation when Alice wishes to test the proximity with a group of users she has not met. Hence, if a proximity test yields a positive result (i.e., two users are close by), a secure handshake protocol shall follow, allowing Alice and Bob to establish a secure channel to communicate subsequently. If the proximity test needs to be carried out between each pair of users, it will be more communication-efficient if the handshake can be performed in non-interactive fashion.

Tunable Granularity Level: One main drawback of broadcast model is users' loss of granularity control of proximity testing, since Alice can not implement different granularities for different users in the broadcast messages. In order to achieve fine-grained privacy control, our design should allow users to negotiate a mutually agreed granularity level before proximity test.

Security: The main security goal for proximity test is to design *unforgeable location tags* so that the protocol is robust against *location cheating*. A location cheating happens when one party is able to convince the other party with an untruthful location. In our case, if Bob can trick Alice into believing that he is within her proximity while he actually is not, he has successfully launched a location cheating attack. Unforgeable location is extremely important for location based services. To the best of our knowledge, we are among the first to address the location unforgeability in proximity test.

Privacy: From the server perspective, the privacy goal of the protocol is to conceal users' location information from the server. Specifically, after the proximity test, the server can not infer users' locations. From the users' perspective, users should have location privacy against each other when they are far away. When they are nearby, user should learn nothing except the fact that they are close.

Efficiency: Existing private proximity test protocols [17, 21] operate on pairs of users. If Alice wants to test a group of n users, she has to run the protocol n times with each and every user in the group. This results in a bandwidth complexity of $O(n)$ and a computation complexity of $O(n)$ at Alice side. Our goal is to design an efficient protocol where Alice and each participant only submit their location tags once to the server. This leads to a communication complexity of $O(1)$ at user's side. This represents significant efficiency improvement comparing to the existing schemes.

3 Location Tags from Environmental Signals

Introduced in [17], a *location tag* can be regarded as a token of proof associated with a point in space and time. It is a collection of signals presented at a certain location at a certain time. From the functionality point of view, a good location tag should at least have the *reproducibility* property: If two measurements at the same space and time yield tags T_1 and T_2, then T_1 and T_2 match with high probability. On the other hand, from the security point of view, in order to be cheat-proof, a good location tag must have *unpredictability* property: An adversary not at a specific place and time is unable to produce a tag that matches the tag constructed at that location at that time. Note that this feature basically requires a location tag carries high entropy.

3.1 Sources of Location Tags

In our study, we have explored two possible sources of location tags: (1) using 802.11 frames in WiFi network. (2) using control messages in 4G LTE network. We consider 802.11 frame headers as a perfect location tag source with appropriate length and sufficient entropy. In our early design, we also tried using frame bodies as location tag sources. Though the resulting location tags pass the entropy and unpredictability evaluation, the low reproducibility quality rendered the location tag unusable. The shortcoming of WiFi-based location tag is its limited range. To provide wider coverage, we also study the possibility of generating location tags through cellular network traffic. The control messages, such as the temporary cell radio-network temporary identifier (TC-RNTI), are messages sent between LTE eNodeB (i.e., base station) and users' terminals for identification and resource allocation. They are usually locally assigned by the eNodeB and can be captured by all terminals. For example, the TC-RNTI is a 16-bit random number assigned to mobile terminal. Therefore, users who observe similar set of TC-RNTIs are likely under the same region.

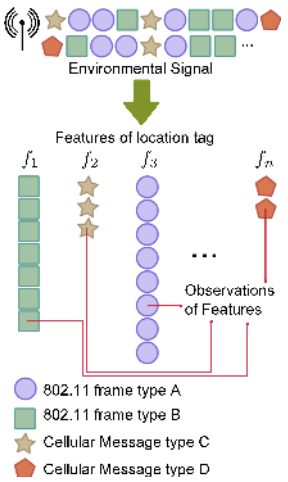


Fig. 1. Features, observations of location tag

For each location tag source, the amount of traffic necessary to generate a distinct and secure location tag is significant. Consequently, it consumes storage

space and computing power of mobile devices to store and process the data. To cope with this problem, we propose to divide the traffic into different groups based on the types of frames and store them using a space-efficient data structure. As shown in figure 1, each type of 802.11 frames or cellular messages forms a *feature* of the signal source. A location tag, therefore, can be represented by a collection of features $\{f_1, f_2, ..., f_z\}$ of the signal source. Each feature consists of many elements, or *observations*, which corresponding to data capture from one 802.11 frame or cellular message. For storage efficiency, we utilize a bloom filter to represent the many observations for each feature, which we will discuss in detail in section 4.

3.2 Entropy and Unpredictability

A good location tag should be time-variant and have sufficiently high entropy in order to satisfy the unpredictability requirement. The most straight forward way to measure the entropy of location tags is by measuring the length of the random values contained in the location tag. However, it is not difficult to see that not all sources we used are truly random. Hence, the traditional method tends to overestimate the entropy amount. In order to estimate the entropy more accurately, we use techniques from statistical language processing [13], namely n-gram Markov model, to evaluate the entropy contained in location tags.

The idea is that, if a feature of a location tag is a sequence of observations, the randomness of the feature can be interpreted as the probability that an adversary successfully predicts the next observation based on previous n observations. This probability can be modeled using an n-gram Markov model. For a feature consists of N observations $w_1, ..., w_N$, the probability that the adversary successfully predicts the entire sequence is

$$P(w_1, ..., w_N) = \prod_{i=1}^{N} P(w_i | w_{i-n+1}, ..., w_{i-1}) \tag{1}$$

where the conditional probabilities can be computed from the following formula

$$P(w_i | w_{i-n+1}, ..., w_{i-1}) = \frac{C(w_{i-n+1}, ..., w_{i-1}, w_i)}{\sum_{x \in V} C(w_{i-n+1}, ..., w_{i-1}, x)} \tag{2}$$

where $C(w_1, ..., w_n)$ represents the frequency of n-gram $w_1, ..., w_n$ in the initial sequence. In our experiment, the size of the captured observations for each feature is between 2000 to 5000 depending on the type of features. Due to the computation capability of our workstation, we use a trigram model to estimate the entropy of the sequence. According to the definition of *Shannon entropy*, the entropy of the feature is calculated as

$$\mathbf{H}(w_1, ..., w_N) = P(w_1, ..., w_N) \log P(w_1, ..., w_N) \tag{3}$$

We show the entropy of 802.11 frame headers in figure 2. The beacon frames contain the least amount of entropy since they are transmitted at a regular 1,024

microseconds (μs) intervals with consecutive sequence numbers. The probing request frames, on the other hand, contain the most amount of entropy since the algorithm used to scan for access points is not explicitly defined in 802.11 standard. The interval and format of probing frames are different depending on the device drivers and user's access pattern [4]. In figure 3, we show the entropy of LTE control messages. Among them, the TC-RNTI and UL-Grant messages contain the highest entropy since the eNodeB issues different TC-RNTI and UL-Grant for the same terminal during each random access session. Compared to that, the entropy in random access preamble and C-RNTI is significantly lower due to limited formats or timing variations. Heuristic results show that location tags with entropy higher than 64 *bits* is considered "unpredictable" [17]. Therefore, only by including multiple features we can construct location tags that are unpredictable to adversaries.

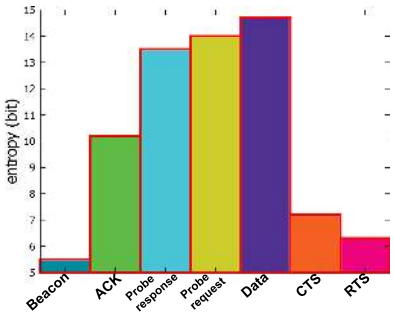


Fig. 2. 802.11 frame headers entropy

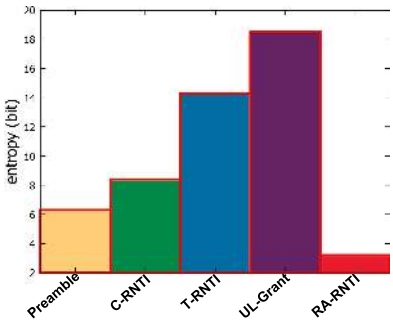


Fig. 3. LTE control messages entropy

4 SHARP: Private Proximity Test and Secure Handshake Protocol

Our private proximity test and secure handshake protocol, SHARP, is a two-step protocol designed for one-to-many proximity test between users that share no prior-secrets. During the first step, upon receiving the request from Alice, the server identifies a group of users designated by Alice and notifies users to construct their location tags simultaneously. Alice embeds a temporary session key K in her location tag and sends it to the group. During the second step, users in the group first try to extract K. Only those within a coarse-grained proximity of Alice can succeed, who then return a keyed hash of their current locations using grid map representation to Alice for fine-grained matching. We exploit bloom filter, a space-efficient randomized data structure, to compactly represent everyone's location tag while using fuzzy extractor technique to accomplish secure handshake. The protocol has the following main advantages: (1) it is far more scalable in that it effectively filters out users who are far away during key establishment without letting Alice interact with each and every one of them.

(2) it allows users to control granularity by negotiating the size of cells on the grid map.

4.1 Bloom Filter and Fuzzy Extractor

A Bloom filter is a space-efficient probabilistic data structure that is used to succinctly represent a set in order to support membership queries [16]. A bloom filter is a bit array of length m, and k independent hash functions $H_1(), H_2(), ..., H_k() : \{0,1\}^* \rightarrow \{0,1\}^{\log m}$ are used to insert and query the original data elements in the array by their hashed locations. In a bloom filter based membership test, false positives are possible, but false negatives are not. In our case, we represent each feature of a location tag with a bloom filter by adding all the observations into the bloom filter.

A fuzzy extractor [3] is a pair of randomized procedures, *generate* and *reproduce*, that allow one to extract some randomness value from an input and then successfully reproduce it from any inputs that is similar to the original input. In our case, the randomness value represents the temporary session key K, whereas the input represents the location tag. In other words, only a user with a similar location tag can reproduce K. In [3], Dodis et. al. proposed using *error correcting code* as a building block of fuzzy extractor. Particularly, we use the BCH error correcting code in our implementation. It has been shown that BCH code can be decoded in polynomial time w.r.t. the *weight* of the received corrupted codeword using syndrome decoding. The details of syndrome decoding can be found in [12, 3].

4.2 System Setup

As shown in figure 4, the system adopts a *grid reference* to represent locations, where grid indices represent areas covered by grid cells. Users share a list of coordinate-axis aligned grid system $G = \{g_0, ..., g_l\}$ of different levels. At each level l, the grid cell size, i.e., width and height, is fixed and equal to $L(l)$. Let $L(0) > L(1) > ... > L(l)$. Additionally, the system defines a security parameter κ, a cryptographic hash function $\mathcal{H}(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^s$, and a keyed cryptographic hash function $\mathcal{H}'(\cdot,\cdot) : \{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^s$ (can be an HMAC instantiated by SHA-256). Note that, $\mathcal{H}$, $\mathcal{H}'$ and G are known by all users and the server.

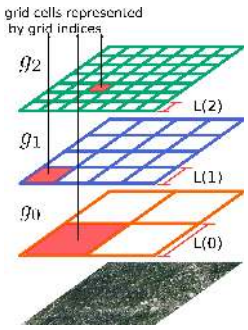


Fig. 4. Grid reference system

4.3 Proximity-based Filtering

The protocol starts by Alice sending her test request to the server, declaring the user group she wants to test. Upon receiving the request, the server broadcasts a synchronization signal to Alice and her intended testing group, and all users construct their location tags by collecting observations from a set of features

$\{f_1, f_2, ..., f_z\}$. For each f, Alice constructs a bloom filter bit array B_f with a given false positive rate p.

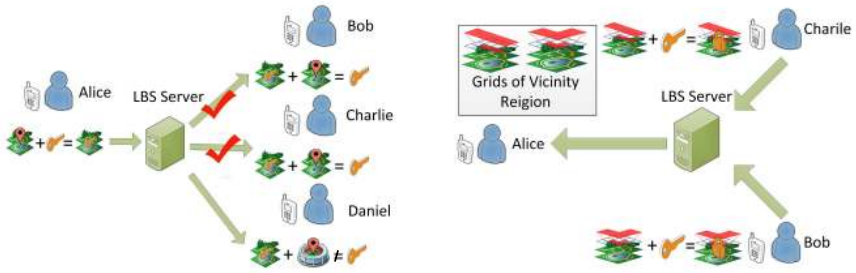
Alice adds each observation w of f into B_f by hashing it to k positions in the bit array using $H_1(), H_2(), ..., H_k()$. Alice then computes t "syndromes"[3] using the following equation, where t is the number of errors that the BCH code can correct:

$$S_i = \sum_{x \in B_f} x^i \quad i \in \{1, 3, ..., 2t-1\} \tag{4}$$

where x represents the index of those positions in B_f that are set to 1. The computations are done within a Galois Field. Assume the resulting syndrome set for each B_f is $\mathsf{syn}(B_f) = (S_1, S_3, ..., S_{2t-1})$. Alice generates a location tag T that can tolerate up to t errors in each feature:

$$T = \{\mathsf{syn}(B_{f_1}), ..., \mathsf{syn}(B_{f_z})\} \tag{5}$$

Next, in order to embed a secret session key in the location tag, Alice creates a fuzzy extractor by hashing all the location features by computing $B_0 = \mathcal{H}(B_{f_1}||B_{f_2}||\cdots||B_{f_z})$. Alice then generates a κ-bit random number (helper string) y, and computes $K = \mathcal{H}'(B_0, y)$ as the temporary session key. Alice can control testing granularity by choosing a subset $G_{Alice} \subset G$. Let $|B| = \{|B_{f_1}|, ..., |B_{f_z}|\}$, representing the length of all bloom filters. Together, Alice sends a message $m_a = \{\text{"Alice"}, T, |B|, y, G_{Alice}\}$ to the server. The server broadcasts m_a to the testing user group.



(a) Initial Proximity-based Filtering: Alice embeds a secret key in her location tag

(b) Fine-Grained Proximity Test: Alice learns users' locations by evaluating the grid indices

Fig. 5. Two steps of SHARP.

4.4 Fine-Grained Proximity Test

Upon receiving m_a, Bob tries to extract Alice's temporary session key using his observations of the features set. For each feature, Bob creates a bloom filter bit

[3] Intuitively, a syndrome is an error checking value of a codeword (here, B_f is considered as a codeword).

array B'_f of the same length as B_f and uses $\mathsf{syn}(B_f)$ to correct the difference between B_f and B'_f. Assume the syndromes of B'_f is $\mathsf{syn}(B'_f) = (S'_1, S'_3, ..., S'_{2t-1})$, Let $\sigma_i = S'_i - S_i$. The *error detecting vector* [3] of B'_f is:

$$\mathrm{syn}(B'_f) = (\sigma_1, \sigma_3, ..., \sigma_{2t-1}) \tag{6}$$

The corresponding *error correcting vector* $\mathrm{supp}(B'_f)$, which represents the difference of B_f and B'_f, can only be computed correctly from $\mathrm{syn}(B'_f)$ when $\mathrm{supp}(B'_f) < t$ [3].

$$\mathrm{supp}(B'_f) = B_f \ \Delta \ B'_f \triangleq \{x \in B_f \cup B'_f | x \notin B_f \cap B'_f\} \tag{7}$$

When Bob and Alice are nearby, the difference between B_f and B'_f is smaller than t. Bob succeeds in computing $\mathrm{syn}(B'_f)$ and obtains the original B_f through

$$B_f = B'_f \ \Delta \ \mathrm{supp}(B'_f) \tag{8}$$

Once Bob reconstructed all the B_fs, he can derive the original B_0 and recover $K = \mathcal{H}'(B_0, y)$. Bob can control the testing granularity by searching through G_{Alice} to find a reasonable granularity level and blind his grid index b with K, by computing $B = \mathcal{H}'(K, b||\text{“Bob”})$. If Bob does not agree on any of the granularity levels, he has two choices: (1) submit nothing indicating he does not allow Alice to carry out fine-grained proximity test. (2) submit multiple location index numbers to mask his actual location. Finally, Bob sends the message $m_b = \{\text{“Bob”}, B\}$ back to server.

The server forwards m_b to Alice. Alice can then searches through all the grid cells that she regarded as in her proximity; if she can find one b that is within one of her nearby cells and satisfies $\mathcal{H}'(K, b||\text{“Bob”}) = B$, then Alice learns that Bob is located in b. After that, Alice knows a list of users within her proximity range, and she can choose to securely communicate with one (or more) of them using the session key K.

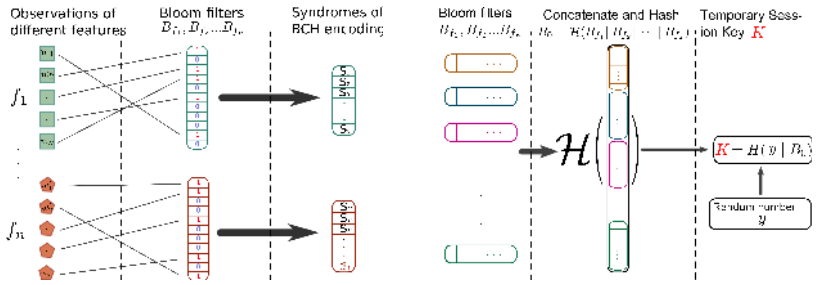
Note that, an attacker may try to send back multiple malformed responses to Alice to exhaust her resources. However, dealing with denial-of-service attack is out of the scope of this paper. We can use existing methods, for example, IPSec or TLS where the server can authenticate the users.

5 Security and Privacy Analysis

5.1 Entropy Loss and Location Unforgeability

SHARP provides unforgeable (unpredictable) location tags. In section 3 we evaluated the entropy contained in location tag sources. However, best practices mandate that we also consider the entropy loss during data processing. In this section, we derive the total entropy loss in our design. Consequently, we show that total amount of entropy loss is limited and the remaining guessing entropy of location tag remains high.

Assuming the location source contains h_0 bits of entropy. In our protocol, the entropy loss happens in two places: (1) when we pack the location tag sources



(a) Create Location Tag (b) Embed Ephemeral Key

Fig. 6. Construct location tag using fuzzy extractor and bloom filter

into bloom filters. (2) when we generate the fuzzy extractor from bloom filters. Note that when packing a set of elements into a bloom filter, the entropy loss is related to the probability rate of false positive [16]. Consider a bloom filter of length m is used to represent a set of n_b elements. From [16], the probability of a false positive is

$$p = (1 - (1 - \frac{1}{m})^{kn_b})^k \approx (1 - e^{-kn_b/m})^k = (1 - v)^k \tag{9}$$

where $v = 1 - p^{1/k}$ is probability of a bit being set to 1 in the bloom filter. Hence the entropy loss during bloom filter construction is

$$h_1 = (1 + v \log v + (1 - v) \log(1 - v))h_0 \tag{10}$$

By taking the derivative of the formula, p has a global minimum value $(1/2)^k = (0.6185)^{m/n})$ when $k = (\ln 2) \cdot (m/n)$. However, we shall explicitly note that in our design, balancing among m, n, k to achieve minimum p is not our main focus.

The second entropy loss happens during fuzzy extractor construction. In general, [18] shows that the entropy loss of a fuzzy extractor is upper-bounded by its entropy loss on the uniform distribution of inputs. In particular, the input of the fuzzy extractor in our design is the bloom filter bit array of length m. Assuming we apply BCH code with code length n_B to the bit array. Since the BCH code family is optimal for $t \ll n_B$ by the Hamming bound [12]. the entropy loss of syndrome fuzzy extractor with a BCH code is

$$h_2 = \frac{t(h_0 - h_1)}{n_B} \log(n_B + 1) \tag{11}$$

The overall entropy loss of our design is thus

$$h = h_1 + h_2 \tag{12}$$

Particularly, in Table 1, we show the entropy loss using a location tag source of entropy equal to 64 bits. In our evaluation, the average length of the bloom filter

is $m = 2^{10} = 1024(bits)$. The total entropy loss is around 7.8 bits. Therefore, the remaining guessing entropy remains high to secure the protocol against location cheating attack.

false positive rate	error tolerance	entropy loss
0.01	10/1024	3.7 *bits*
0.01	30/1024	7.1 *bits*
0.1	10/1024	4.3 *bits*
0.1	30/1024	8.7 *bits*

Table 1. Entropy Loss

5.2 Location Privacy

When Alice and Bob are far apart, the set difference between their location tags, A and B, will be greater than t. This means Bob can not correct all the errors using BCH syndrome, and therefore his view of Alice's location tag is indistinguishable from random. Next, when multiple users $b_1, ..., b_n$ outside of Alice's proximity range collude, denoting their location tags by $B_1, ...B_n$, we have

$$A \ \Delta \ B_i > t \quad 1 \leq i \leq n \tag{13}$$

Assuming B_is are pairwise disjoint. It is easy to see that the symmetric difference between the joint location tags $B = B_1 \bigcup B_2, ..., \bigcup B_n$ and A is still greater than t. Hence, Alice has privacy when multiple unmatched users collude. The server can not learn Alice's location or secret session key, since it is infeasible to record the environmental signal of all locations at all times.

We should note that unlike previous work [17] in our protocol, when Alice and Bob are nearby, Alice still has location privacy against Bob. The reasons is: Bob only gains knowledge of Alice's rough whereabouts during the key establishment step. In the second step, Bob does not receive any message from Alice, therefore can not know Alice's exact location even if the matching result is positive. Bob, too, can protect his location privacy against Alice by hiding his actual location within multiple grid indices. Compared with protocol using expensive PTSI operation, we achieve the same privacy level with less computational cost.

6 Experimental Evaluation

6.1 Experiment Setup

To test our design, we carry out a proof-of-concept experiment on the 802.11g WLAN network on campus. We use Dell inspiron 5100 with a 32-bit, 533MHz Pentium 4 CPU to log the WLAN traffic at varying distances. Using the logged data, we evaluate the performance of SHARP from the following aspects: (1) we measured the success probability of extracting temporary key using location tags. (2) we evaluated how the success rate is affected by the size of the bloom

filter, clock synchronization error, and user mobility. (3) we measured the CPU time required to generate the location tag and to extract the temporary key. The detailed experiment configuration is as follows.

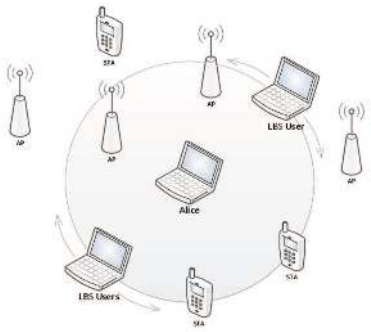


Fig. 7. Experiment configuration

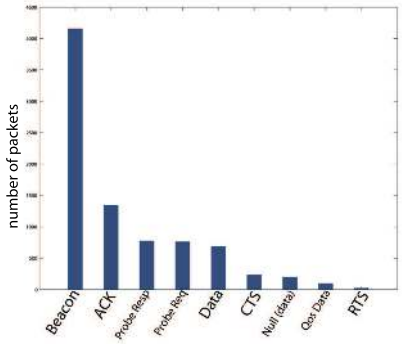


Fig. 8. Traffic Summary

1. We deployed three client laptops at different locations running Wireshark in the promiscuous mode. All three laptops are loosely synchronized before the test.

2. We configured one of the laptop to act as Alice. Before each capture, Alice sends out the synchronization signal to state the starting time of next capture. The other two laptops were configured as testees who participate in the proximity test. They receive the synchronization signal and schedule the next capture. Each capture is carried out for ten seconds. Captures are repeated for multiple times at the radius of $0meter$, $5meter$, $10meter$, $15meter$, $20meter$, $25meter$, $30meter$ and $35meter$.

3. After each capture, the program running on Alice reads frames from the capture (*.pcap*) file and sorts them according to the frame type of the packets. It generates the location tags based on captured packets and sends the size of the bloom filters to the other two laptops. Upon receiving these parameters, the other two laptops generate their location tags.

In figure 8, we show the histogram of various frame types from traffic analysis. During the test, we saw an average of 8432 packets on channel one. Half of them are 802.11 beacon packets. The rest of the packets are ACK, Probe response, Probe request, etc. In table 2, we show that an average of 105 different MAC addresses was detected during the test. 95 of them are 802.11 client stations whereas 10 of them are 802.11 access points. According to [19], our testing environment can be considered as a typical WLAN networks scenario in metropolitan areas.

6.2 Location Tag Reproducibility

Figure 9(a) shows users' success probability of extracting Alice's temporary session key at various distances. Interestingly, there is a clear cut-off distance in the graph. Within 30 meters, the difference between location tags is fairly small

Wireless Stations			Access Points		
avg	max	min	avg	max	min
95	121	73	10	13	9

Table 2. Number of 802.11 station and access points detected during the test.

which indicates Bob can successfully reproduce Alice's location tag. Beyond 30 meters, with quickly increased probability Bob won't be able to reproduce Alice's location tag due to the larger difference between location tags. In other words, the location tags we tested are either nearly disjoint or nearly identical. Thus, an efficient test that can accept near-identical sets and reject near-disjoint set is sufficient for our purpose. In [11], Lin *et al.* showed similar result using paging channel messages in GSM cellular networks. Hence, with all these findings, we argue that BCH error correcting coding approach with small t is superior to private cardinality threshold set matching approach [5] for our purpose in term of practicality and usability.

Bloom Filter and Reproducibility In figure 9(b), we show how the size of the bloom filter affects location tag reproducibility. It appears that when we increase the false positive rate of the bloom filter, the success probability at the far side increases. The reason is that increasing the false positive rate f is equivalent to reducing the length of the bit array. When the length of a bloom filter is small, the probability that each bit in the bit array being set to 1 increases. If the probability increases to 100%, the bloom filter contains no information entropy. The corresponding location tag becomes independent of location. Hence the difference between location tags is always 0 regardless of the distance. Clearly, there is a balance between the entropy loss versus the location tag reproducibility. When bloom filter is large, the entropy loss is small, yet it requires Bob to have stronger error correcting capability to reproduce the location tag. When the bloom filter is small, the location tag reproducibility is high, yet, the location tag itself become less distinct.

Clock Synchronization Error and Reproducibility We tested the protocol's performance against clock synchronization error. As shown in figure 9(c), when users did not start the location feature extraction process simultaneously, the average difference between location tags increased. Yet, the cut-off distance stays the same. Hence, our protocol only requires very loose time synchronization between users.

Mobility and Reproducibility We evaluated how the users' mobility affects the performance of the protocol. In the experiment, we let Bob randomly move slowly around Alice. Compared with the stationary case, Bob's chance of successfully extracting Alice's secret key slightly increases. The reason is when Bob is moving, he is able to see more access points and wireless stations compare to a stationary testee. However, the advantage Bob gains by moving is minimal since each capture window is only 10 seconds.

6.3 Storage and Communication Efficiency

The use of the bloom filter and BCH encoding during location feature extraction reduces the communication cost of the protocol. In this section, we show the location tag size and location generation time of our protocol.

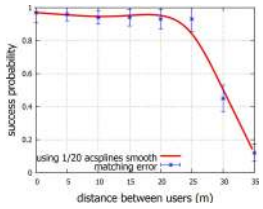
Location Tag Size We compared the size of the location tag of our design and the location tag generated through other polynomial based reconstruction [17]. We defined the fuzzy match threshold as follows. Assume a total number of n packets are captured. For our protocol, in order to generate a total of (n, t) fuzzy match, the number of t is distributed into each location tag source according to the total number of observations from that source. For example, if a location feature contains m packets, we create a $(m, \frac{tm}{n})$ fuzzy match. In [17], the location tag is generated by create a $n - t$ degree polynomial.

As shown in figure 9(e), if the size of each packet's hash value is k bit, the size of the location tag generated in [17] is approximately $2(n-t)k$, whereas the location tag generated with SHARP is approximately $t \ln(\frac{n \ln(p)}{m})$. SHARP clearly shows superior performance to polynomial based location tag construction. This is due to the usage of bloom filter and the fact that the location tag in SHARP only consists of the syndromes of the BCH code.

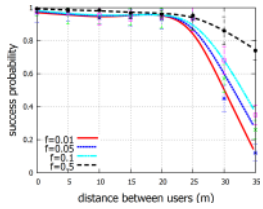
Location Tag Generation Time In Figure 9(f), we show the location tag generation time of our design. The main part of the generation time is contributed by: (1) Adding element to the bloom filter, and (2) calculating BCH syndromes. In (1), in order to add one element to the bloom filter, k hash functions are used. In our implementation, we use a 160 bit SHA-1 hash function which costs around $0.5ms$ to finish on the laptop. The total time of part (1) will grow linearly with the number of observations. In (2) the time consumption of BCH encoding is polynomial in $\log n$, where n is the size of the bloom filter [3]. Therefore, the time consumption of (1) dominates the overall location tag generation time.

7 Conclusion

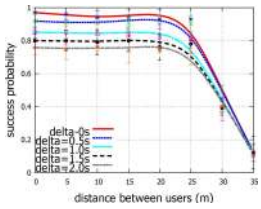
In this paper, we address the privacy and security issues of proximity test in location based services. We aim at letting users to find others who are within a certain geographical region or range with a help of a oblivious server, without pre-established secret keys while hiding user location information from the server. In order to prevent location cheating, we propose to use multiple types of real-time and location-dependent environmental signals to construct location tag. The location tag is the key to proximity matching, where fuzzy extractor is exploited to extract a secret key from two matching users. In addition, the location tag is organized in a bloom filter, such that users can choose their own matching sensitivity at ease via tuning the parameter of the bloom filter. Furthermore, we also improve the accuracy and fine-grainedness of the proximity test using geographical grid and keyed hashing. We allow user to control granularity by negotiating between different grid cell sizes. Through both theoretical analysis and experimental evaluation, we show that our location tag has enough freshness



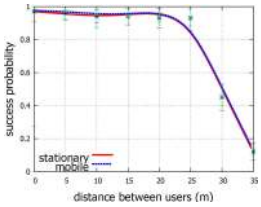
(a) Probability of successful key establishment versus distance



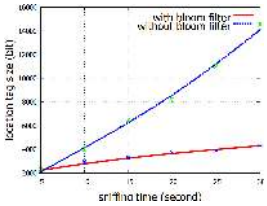
(b) Probability of successful key establishment versus false positive rate of bloom filters



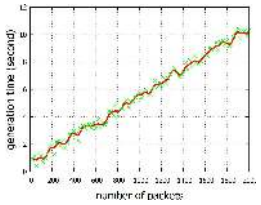
(c) Probability of successful key establishment versus clock synchronization error



(d) Probability of successful key establishment versus mobility



(e) Location tag size versus sniffing time



(f) Location tag generation time versus number of observations

and entropy to defend against location cheating. Our scheme is mostly non-interactive, does not require strict synchronization, and enjoys high scalability and efficiency.

Acknowledgments. This work was partially supported by US National Science Foundation under grants CNS-1155988, CNS-1156318, and CNS-0910531.

References

1. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.
2. W. Chang, J. Wu, and C. C. Tan. Enhancing mobile social network privacy. In *Proc. IEEE Global Communications Conference*, 2011.
3. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology-Eurocrypt 2004*, pages 523–540. Springer, 2004.
4. J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proc. 15th USENIX Security Symposium*, 2006.
5. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.
6. G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: Anonymizers are not necessary. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2008.

7. W. He, X. Liu, and M. Ren. Location cheating: A security challenge to location-based social network services. In *Proc. 31st IEEE International Conference on Distributed Computing Systems*, 2011.
8. M. Li, N. Cao, S. Yu, and W. Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *Proc. 30th IEEE International Conference on Computer Communications*, 2011.
9. M. Li, W. Lou, and K. Ren. Data security and privacy in wireless body area networks. *Journal of Wireless Communications*, 17(1), 2010.
10. X. Liang, R. Lu, L. Chen, X. Lin, and X. Shen. Pec: A privacy-preserving emergency call scheme for mobile healthcare social networks. *Journal of Communications and Networks*, 13(2), 2011.
11. Z. Lin, D. F. Kune, and N. Hoppe. Efficient private proximity testing with gsm location sketches. In *Proc. 32nd International Cryptology Conference*, 2012.
12. J. H. V. Lint. *Introduction to Coding Theory*, volume 86. Springer Verlag, 1999.
13. C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
14. S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia. Privacy-aware proximity based services. In *Proc. 10th IEEE International Conference on Mobile Data Management: Systems, Services and Middleware*, 2009.
15. J. Meyerowitz and R. R. Choudhury. Hiding stars with fireworks: Location privacy through camouflage. In *Proc. 15th ACM Annual International Conference on Mobile Computing and Networking*, 2009.
16. M. Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking (TON)*, 10(5):604–612, 2002.
17. A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *Proc. 18th Annual Network & Distributed System Security Symposium*, 2011.
18. L. Reyzin. Entropy loss is maximal for uniform inputs. Technical report, Boston University Computer Science Department, 2007.
19. K. V. Shrikhande, I. M. White, D. rudee Wonglumsom, S. M. Gemelos, M. S. Rogge, Y. Fukashiro, M. Avenarius, and L. G. Kazovsky. Hornet: A packet-over-wdm multiple access metropolitan area ring network. *Journal on Selected Areas in Communications*, 18(10):2004–2016, 2000.
20. L. Šikšnys, J. R. Thomsen, S. Šaltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *Proc. 11th IEEE International Conference on Mobile Data Management*, 2010.
21. L. Šikšnys, J. R. Thomsen, S. Šaltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. *Advances in Spatial and Temporal Databases*, pages 405–410, 2009.
22. N. Talukder and S. I. Ahamed. Preventing multi-query attack in location-based services. In *Proc. 3rd ACM Conference on Wireless Network Security*, 2010.
23. J. Y. Tsai, P. G. Kelley, L. F. Cranor, and N. Sadeh. Location-sharing technologies: Privacy risks and controls. *I/S: A Journal of Law & Policy for the Information Society*, 6:119–317, 2010.
24. W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *Proc. 35th ACM SIGMOD International Conference on Management of Data*, 2009.
25. Z. Zhu and G. Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *Proc. 30th IEEE International Conference on Computer Communications*, 2011.