Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PRIVACY-PRESERVING QUERIES FOR LBS: INDEPENDENT SECURED HASH FUNCTION

[1]ABDULLAH ALBELAIHY, [2] JONATHAN CAZALAS, [3]VIJEY THAYANANTHAN

[1]King Abdulaziz University, Department of Computer Science, Saudi Arabia

[2] Florida Southern College, Department of Computer Science, Lakeland, Florida, USA

[3] King Abdulaziz University, Department of Computer Science, Jeddah, Saudi Arabia

E-mail: [1]aalbelaihy{at}stu.kau.edu.sa, [2] jcazalas{at}flsouthern.edu, [3] vthayanathan@kau.edu.sa

ABSTRACT

While location-based services have become ubiquitous, seemingly permeating our personal and professional lives, their inherent nature poses security risks to users, who are forced to reveal their highly-sensitive location data in order to make effective use of the service. Towards this end, a litany of techniques have been proposed to provide efficient answers for privacy-preserving queries in LBS. Spatial bloom filters were initially proposed as an efficient data structure used to manage special and geographic information in an space-efficient manner. Unfortunately, bloom filters suffer from two deficiencies: they leak at most one bit of information per query, and the hash functions require careful design and security analysis in order to be orthogonal and independent. In fact, developing quality hash function is paramount. We propose a method to automatically generate good, independent hash functions, with the goal of reducing information leakage. This means that even if one of the hash function is broken, for any reason, nothing can be learned about any other hash function. The results show that our proposed Hash functions are less dependent and leaked than the compared approach, while still seeing a notable improvement in performance.

Keywords: *Privacy, Bloom filter, LBS, Mobile user, Hash function.*

1. INTRODUCTION

Location-based services have become ubiquitous, effectively penetrated all smartphones and GPS-enabled devices, providing tremendous value to customers. While LBSs have grown in popularity, they are not without flaws; specifically, the user of LBS must reveal his or her location data in order to take full advantage of the service, thereby potentially risking their privacy and security [1].

Mobile users' awareness and opportunities to communicate with and within their environment have increased due to increased familiarity with LBS [2]. Mobile users can send queries to the servers of LBSs if necessary [3]. Thus, services related to "point of interest" (POIs) can be obtained tacitly by mobile users. For example, mobile users have become capable of easily identifying the closest banks, restaurants and easily verifying data related to the prices of some nearby restaurant. In short, LBSs are regarded as extremely beneficial. Nonetheless, the services present risks to the protection of users' privacy, as the service providers provide information relating to the user's location. An attacker could deduce sensitive private information related to the service recipients through information gathering about the location of users relating to their LBS queries.

Both location privacy and query privacy are issues that are brought to light through LBS. For example, information about users living in a rural area can be disclosed in term of a large area response. In fact, this allows the preservation of user's location privacy. While in some cases, through an LBS server, a submitted query of a location-based service gets a user-based response [1],[4]. Therefore, the interest, as well as the location, was included in this query. Important information like user IDs and query radius is also contained in the information, and these components can be captured by an adversary.

Even though such type of submitted information is revealed, it could be hacked, since the LBS servers can be unreliable [5], [6]. The consequence of this is that with the help of LBSs, the server may identify the location of the mobile user. Moreover,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

malicious servers may also identify the queries that were being submitted to the LBS servers and eventually determine the activities of the mobile users [7],[8]. Hence, one can infer that using or integrating LBSs leads to the ability to follow or track the mobile users and also to the release of all personal information of the mobile users to any 3rd parties such as advertisers [9]. Subsequently, protecting the users is paramount and should be ensured.

Under normal conditions, LBS to a user depend on the query presented by the user, which inherently contains the user's location, query details, and other information, like the radius of the query and a user's ID, etc. Notwithstanding, the information provided can be eavesdropped and misused by a malicious LBS server, resulting in the private or secret information of the user being revealed to a third party, such as advertisers. Therefore, more attention needs to be paid to protecting users' privacy.

Several techniques have been proposed in order to provide an optimal solution for privacy-preserving queries in LBS. The aim is to be able to make queries to a location-based service while providing guarantees concerning privacy and efficiency. As expected, the issue raises several questions about security in general. One of them is any information leakage in a secure system. If one bit is leaked per query, and thousands of queries are made, then it may be possible for the attacker to learn some information that should remain secret. Recent research has taken advantage on bloom filters to address this problem. Unfortunately, bloom filters suffer from two deficiencies:

1. They leak at most one bit of information per query.
2. The hash functions Hk require careful design and security analysis so that they are orthogonal and independent.

Accordingly, our idea is to provide an "engine" that can be used to generate an arbitrary set of orthogonal and independent hash functions, such that the joint distributions of two events A and B is the probability, that if event A happens, then event B happens. If A and B are completely independent, then the joint distribution will be 0; while if A and B are completely dependent, then the joint distribution will be 1. The joint distribution is a way of measuring whether two events or functions, depend on each other. For the hash functions we would like to use in this paper, it is important that they be as independent as possible, with the joint distributions effectively reaching to 0, as shown in Equation 1 below:

$$P(Hi|Hj) - P(Hi) - P(Hj) \quad (1)$$

Equation 1 is shown to be bounded from above by an arbitrarily small value E, with negligible probability of finding x, y such that:

$$|| P(Hi(x)|Hj(y)) - P(Hi(x)) - P(Hj(y))||>E \quad (2)$$

Where P(Hi(x)) in this equation denotes the probability [10].

In terms of hash function generation, there is a real opportunity to increase entropy and decrease mutual information between the hash functions, even if they are initially very weak. Thus, we will study ways in which we can insure that:

$$P(A(Hi)|A(Hj)) << P(Hi|Hj) \quad (3)$$

This type of approach is well known in the functional programming community, in that we are attempting to use some version of A to act on the functions Hi, as opposed to acting on the values of these functions. Given the output functions A(H(i)), we calculate correlations over the Bloom Filter keyspace (the number of bits b making up the array). These correlations will be used as statistical measure of success; if a function A increased entropy and decreases mutual information, then we can use A as a method for strengthening the irreversibility of the hash functions.

In general, our contributions are as follows:

- We develop a method to automatically generate an arbitrary set of orthogonal and independent hash functions, with the focus on increasing entropy and decreasing mutual information between the functions. In the event that one hash function leaks information for any reason, the rest of hash functions will not be affected. These two conditions make it harder for the adversary to obtain any meaningful information.
- Finding the better performance through the developed method as in the previous contribution. While much research has been done using bloom filters in a general cryptographic setting, we analyze its application to LBS privacy and how to limit information leakage when providing answers to privacy-preserving LBS queries.

The remainder of the paper is organized as follows. Section 2 reviews other related work.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Additional preliminaries are discussed in Section 3. Sections 4, 5 and 6 discuss our proposed solutions. Lastly, concluding remarks and future work is given in Section 7.

2. RELATED WORK

2.1 Privacy in general

To protect the privacy of users over LBS, there have been various solutions created recently [2], [11]. These studies [3],[12] have revealed that some of these solutions use k-anonymity, it has been observed that the k-anonymity has been presented in order to protect the privacy of the user's location. So, this technique hides the real location of the mobile user, to ensure location privacy to the mobile users. Integration of k-anonymity helps to conceal the real information of the mobile user into k – 1 other mobile users.

Meanwhile, algorithms are used to cloak the locations of the users. The algorithms are R-tree and Grid based. The R-tree algorithm, also called the "Clique Cloak," [13] has different requirements of k-anonymity per user [12]. This cloaking determines whether or not multiple users can share a spatial cloak [14].

2.2 Privacy with Bloom Filter

In [15], a Private Set Intersection (PSI) protocol was proposed based on oblivious Bloom intersection. It consists of linear complexity and relies mostly on efficient symmetric key operations. The authors declared that their method provides high scalability due to its capability in providing a basic protocol and an enhanced protocol that was proved in the semi-honest model and the malicious model respectively.

In [16], authors focused on Online Social Networks (mOSNs) whereas the location sharing service was introduced to mOSNs. They established their study by examining the current problems of location sharing through which they proposed BMobishare to provide a security-enhanced mechanism that ensures privacy location. The authors adopted Bloom Filter for the aim of securing sensitive data compared to other existing methods; unfortunately, an observation of using the bloom filter is that there is an increase in false positives, which we improve by reducing the information leakage by making the hash functions independent of each other's. Further, even if information is leaked, it will be neglected because each of the orthogonal characteristics of each hash function; any leaked information cannot be corelated to other hash functions and can be safely ignored. Unlike the other methods which neglect this aspect, this improvement alone would achieve an increased privacy level without having to incorporate additional encryption technologies.

While in [17], this paper takes into account the two-party computation model where clients desire to form a request to the server, but substantially not willing to reveal their questions to anybody. The solutions for the queries are based majorly on Oblivious Transfer, k - Anonymous Oblivious Transfer and Deterministic Encryption and Bloom Filers. From this point of view, the proposals for the two protocols include the k*l -- Anonymous Oblivious Transfer protocol with the blending of Symmetric Key Encryption (Block Cipher). It also includes k*l -- Anonymous Oblivious Transfer protocol that implies to the approach of the anonymous request problem where the server cannot articulate which record the querying client wants from the registers and neither also the client could obtain anything but the record relating to his request. Through observation, the protocol can achieve anonymity. It is also practical in scenarios in which privacy is most preferred to speed.

In [18], social networks or mobile social networks are becoming common in the world today because of the rapid growth of mobile devices and partly because the mobile users are allowed to communicate within a certain distance with their friends. Because of this feature, many mobile phone companies have come up with many interesting applications, and this creates a security concern for the freedom of their customers.

Therefore, authors proposed a customized privacy mechanism that will not only protect the privacy of user's profile but also it will establish a secure connection between matched users. Besides, the inventor is free to customize the request profile, as well as choosing the needed features and assigning each feature a specific value. Furthermore, the initiator can customize his or her privacy protection level depending on what they desire.

Collusion attacks among unmatched users are a sensitive threat that we guarantee to address because it has precisely not been worked on. Proposed protocol will, therefore, ensure that only matched users can communicate securely with the initiator while there will be little or no information that will be obtainable by the other participants.

To achieve this, they adopted a bloom filter that protects the privacy profile and hence decreasing computational overheads. Finally, conducted a security analysis and performance evaluation to justify the superiority of the protocol.

Moreover, in [19], location-aware applications

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

are one of the biggest and crowned innovations brought by the smartphone era and are well of changing our daily lives throughout the generations. To say just but the truth, the human race is only starting to understand what privacy risks are associated with constant tracking of our localities. For the great acknowledgment location-based services in the future that provide for privacy and security, there is the need for new, privacy-friendly applications and protocols. A new compact data structure is carefully proposed based on Bloom filters that are designed to store location information.

The spatial Bloom filter (SBF), as it is known, is designed with privacy in mind, where it is proved through presenting two private positioning protocols basing the assessment of the old to the new primitive. These rules save the user's exact position privately and also allow the provider of the service to pick up the users' closeness and specificity points of interest, or exclusively predefined areas. In concurrence of time, the user remains much oblivious to the points and areas of interest.

These two proposed protocols are focused on different scenarios that are regarding a two-party setting where communication happens directly between the user and the service provider while, in a three-party setting, the service provider, outsources and the third-party contact with the user. A much more emphasized and detailed evaluation of the effectiveness and safety of the solution shows that privacy can and will be achieved with least computational and communication directly above the ranges of scrutiny. The possibility of spatial Bloom filters regarding overview, safety, and firmness mark them set for distribution, and this creates mode for confidentiality hence protecting location-aware applications.

In this paper [20], the availability of cheap positioning systems made it possible to embed them in smartphones and other small devices. This idea led to the development of a location-aware application that makes it possible for users to demand personalized services depending on their geographical location. There is a need to disclose only a small amount of a user's whereabouts at any given time because of the position of a user that is highly sensitive.

Some applications such as navigation system are based on the movement of the user and will, therefore, require constant tracking while others only require the knowledge of a user's position with an individual area of interest. In this paper, authors hence focus on the application that will only require the knowledge of a user's position with a particular area of interest to determine membership in one or more geographical sets. So they addressed the problem by using Bloom filters which are a compacted data structure for representing sets. In particular, in the paper used the spatial bloom filter (SBF) which are designed to manage spatial and geographical information. Moreover, spatial filters are well suited in enabling privacy in location-aware applications.

The author presented this by providing two multi-party protocols for privacy-preserving computation of location information which is based on well-known homomorphic properties of the public key encryption schemes. The rules keep private the exact position of a user while allowing the provider of the service to learn when the user is near specific points of interest.

With [21], From the study undertaken, a Bloom filter is a simple space-efficient randomized data structure for representing a group so that it backs up the connection requests. In recent years, Bloom filters have improved in acceptance in their database and networking applications. A Bloom filter entails two major phases that termed programming and membership request. From this, there is the introduction of a new method that integrates a hash table (HT) with Bloom filter to reduce the HT access time. This effectively means that when a Bloom filter for an external entry of a program, the next item thus is simultaneously stored in an HT.

Furthermore, to the connection request stage, if the application is proper, concurrently the address of the entry in the HT is created. Thus, the analysis is done for the average bucket size, maximum search length and a number of collisions for the projected method. Moreover, there is the comparison with the fast hash table (FHT) method. The executed approach in a software packet classifies it based on tuple space search within the H3 class of universal hashing functions. Conclusively, as compared to FHT, the method is capable of minimizing the average bucket size, maximum search length and a number of impacts when related to FHT.

On the other hand with [22], there is an academic study entailing interactive hashing. The paper begins by introducing the notion of interactive hashing as a cryptographic primitive while differentiating it from the specifics of the implementations it may present.

In this regard, this paper showed application-independent information ideal conditions that must be ideally satisfied by the interactive hashing.

Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Moreover, the paper provides in detail an analysis of a standard implementation of interactive hashing that meets all the conditions of our definition. From the analysis, get that it represents improvements in restricted contents than in previous attempts. In spite of its generality, the interactive hashing offers a simpler proof of security, which establishes security from a dishonest sender hence reducing his or her probability of cheating.

To prove that a sender who tries to manipulate the protocol in a way that makes it possible for output strings to have a uniquely desirable property would represent a fraction of all the strings from which the probability of both outputs will be from the set. Also, showed the power of interactive hashing as a cryptogrammic tool by looking into the protocols achieving oblivious transfer and which characteristically depends heavily on interactive hashing.

3. PRELIMINARIES

This section will emphasize the concepts, adversary model, and motivations that are incorporated into this paper. This section will also explain the methods used.

3.1 Basic Concept

A Bloom Filter (BF) is a probabilistic data structure that allows for a test for set membership without revealing more than a single bit of information about the set. If A is a putative element of a set S, then a Bloom Filter provides a probabilistic algorithm for determining if A is an element of S without any explicit recursion over the elements of S [23]. Furthermore, a Bloom filter has the property of never having false negatives, so that an assertion "A is not a member of S" is always an accurate assertion, assuming that the algorithm performing the test is benign. False positives are possible, however, so that an assertion "A is a member of S" is only probabilistically correct. Based on the number of bits used to implement the BF, an example will show in figure 1, B denotes Bloom filter; H denotes a hash function.

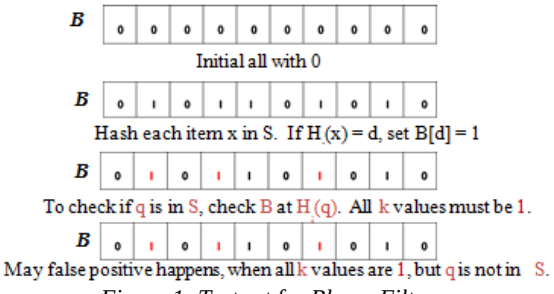

Figure 1: Test set for Bloom Filter

A (BF) is an array of m bits together with a set of k hash Hk functions. Initially, all bits are zero. It is required that the Hk be trapdoor functions, that is, given Hk(x) = y, it is not possible to recover the value of x from y except with negligible probability. It is also required that either:

- The range of all the functions Hk is the range of natural numbers [0, m-1].
- The hash computation is performed mod m, that is that the range of each Hk is a natural number, and the value y is computed as (Hk(x) mod m).

In order to add an element x to the BF, the set of values Hk(x) is computed for each k. For each value y = Hk(x) we then set m[y] = 1. Has collisions are irrelevant; if Ha(x) = Hb(x) = y, where 0 <= a < k and also 0 <= b < k, then we still set m[y] = 1. Thus, it is possible for an input x to be hashed into any number of different values of y, which may have any multiplicity in the range (1k).

In order to test if any element x is a member, each of the Hk(x) is computed. If any of the m[Hk(x)] = 0 then it is known with certainty that the element x is not a member. However, if all m[Hk(x)] = 1 then with probability p we can assert that x is a member of the set. A straightforward yet tedious computation shows that p is approximately equal to (1 – 10-(k/10)) [24].

In a LBS scenario, the LBS has a BF that has been populated with location information encoded as bit vectors. The user generates q (query, assertion) pairs. These queries are then transmitted to the LBS, which generates a bit vector of q values. If the i-th position of this vector is 0, then the pair (Qi, Ai) is absolutely inconsistent, while if the i-th position in this vector is 1, the pair (Qi, Ai) is probabilistically consistent [25]. If the LBS server is benign, then each evaluation will be accurate, within the limits of the BF algorithm. If the LBS server is malicious, then only some evaluations will be accurate. Since the user can generate (Q, A) pairs with a known state, the responses from the server can be tested against these known values.

Even a single false negative is an absolute indication that the server is malicious. A false positive is an indication that the server might be malicious; as with the other algorithms this false positive rate can be tuned to be as small as possible by using N iterations of q queries. Thus, except

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with negligible probability, a malicious LBS server can be detected with a probability of (1 – pN), where p was the legitimate false positive rate stated earlier. Note also that an eavesdropper cannot learn anything from the exchanges, except with negligible probability, due to the fact that it is computationally intractable to invert the Hk.

3.2 Adversary Model

Create a data transfer between an LBS client an LBS server and a secondary server (such as a social media portal) that obeys the following properties:

- The LBS server cannot determine more than 1 bit of information from any list of queries; the secondary server cannot determine any information for the list of queries
- A malicious LBS server can be detected by probability q and the hash functions Hk require careful design and security analysis so that they are orthogonal and independent. This means that even if Hi is broken, for some i, nothing can be learned about any other Hj (secured), j= i.

3.3 Motivation and Basic Idea

The hash functions should be **independent** to the greatest extent possible. (We will precisely define this word below.) A Bloom filter has the following operations: Add, Test, Change, and Unset. There is no delete operation, for reasons that will be described shortly.

Before specifying these operations and the associated standard Bloom filter protocol, it is necessary first to define two additional concepts: **backing store** and **set comprehension**. In many cases, the set member is the key in a key-value pair. As a result, one must store not only the members of the set but also their associated values. In this case, the value cannot be stored in the Bloom filter; it must be stored somewhere else, in a data structure known as the backing store. For small sets, this data structure may be kept in memory, while for larger sets, such as LBS datasets, the backing store is almost always some form of database. Set comprehension refers to the operation of choosing particular members of the set that belong to a specific subset. If S is a set and x is a potential member, we denote the membership test by the notation { x in S }. In the case of set comprehension, we not only test for set membership, but we also test for the truth value of an addition function f(). Since f() must be a boolean function, we often refer to this function as a **predicate**. Set comprehension is written as { x in S | f(x,S) }. The vertical bar | is read as "such that" so the set comprehension test may be read as "x is a member of the set S such that the predicate f(x,S), which denotes f applied to x and S, is true." As an example, consider P, the set of all primes. A membership test for an prime x would be { x in P }, while the membership test for a prime of the form 4n + 1 would be { x in P | (x-1)%4 == 0 }.

Let h_1 to h_k denote the k hash functions. In what follows, we will use x to denote the potential set member, with the hash functions only acting on x. If there is an associated value, we will refer to it as v. We assume the general case where v is always present, with the understanding that if we are only using a Bloom filter with no backing store, the parameter v is omitted. To add a member <x,v> to a Bloom filter we first compute the k values of the hash functions operating on x, that is $h_1(x)$ to $h_k(x)$. Since each of these k values is a positive number less than 2^m, it can be written as a binary number with at most m bits. For each of the $h_i(x)$ values, we note the bit position of each 1 bit in the corresponding binary expansion and set that same bit in the Bloom filter. If v is present, we also add the <x,v> pair in the backing store. For future convenience, we write $M = 2^m$ and $K = k^2$. In order to test if x is a member of the set, and potentially recover its backing store value, we again calculate the k hash-function values $h_i(x)$ and then perform the following algorithm:

```
FOREACH hash function value H = hi(x)
  FOREACH 1 bit in H
     IF that bit is not set in the Bloom filter
     THEN
RETURN
not_a_member
     ENDIF
  ENDFOR
ENDFOR
RETURN maybe_a_member
```

Note that the algorithm never has a false negative, but it may have false positives. That is if the algorithm determines that x is not a member, this is always correct. However, if the algorithm determines that x might be a member, then this is a probabilistic statement. False positives come about because the binary representations of the hash values of x and x' might overlap in some of the bits set in the Bloom filter. After many elements have been added it is possible that the all the bits conforming to the hash-values for x may be set in the Bloom filter, even if x is not a member. If there

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

is a backing store, the ambiguity may be resolved by querying the value that corresponds to x. This type of exchange [21] is shown in figure 2 below.

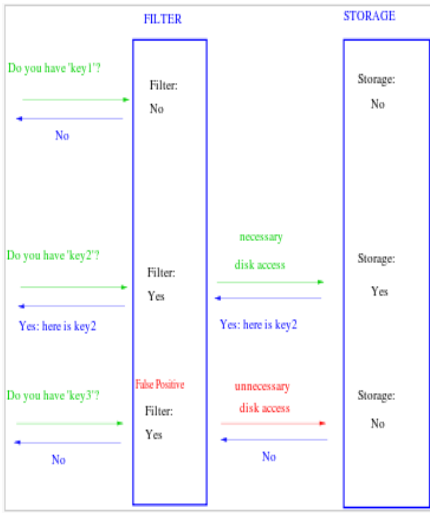

Figure 2: Standard Bloom filter query protocol

The Change operation, which changes a key-value pair <x,v> to a new pair <x,v'>, is simply a Test operation followed by a store operation to the backing store. The Change operation will fail if x is not already a member of the set, so it may be necessary to perform an Add operation, as shown by the following algorithm:

```
b = Change(x,v')
    IF  b is false  THEN
        Add(x,v')
ENDIF
```

Finally, the Unset operation is the same as Change(x, null). There is no way to delete a value x from the Bloom filter, because each 1 bit may have been the result of hash values that have that bit set for some other member x'. In the case of LBS providers, it should be relatively rare to need to delete a set element, and it may be sufficient to nullify the corresponding value in the backing store simply. Note, however, that for each additional null value in the backing store the cost of a look-up becomes very slightly slower. Thus, if many unset operations have been performed, the LBS provider may need to delete all null values, and then regenerate the Bloom filter based on the keys that are still in the backing store after the removal of the null values.

4. DATA ANONYMIZATION: INDEPENDENT SECURED HASH FUNCTION

Choosing good hash functions for a Bloom filter is a critical aspect of creating a secure implementation. Since the hash functions map the bits of a member x into the bits that are set in the Bloom filter, it is important to keep these hash functions independent so that information leakage by a side channel cannot successfully occur (except with negligible probability).

We will now formally define this notation of independence of hash function using the notion of conditional probability. If E is an event, then we refer to <E> as the probability that E has occurred. This has the standard definition of probability as the ratio of the number of times that E has occurred, divided by the number of samples that have been taken.

Given two events E and F we refer to <E|F> as the conditional probability that E has occurred, given that F has already occurred. We can define this as the ratio of the number of occurrences of E following an occurrence of F, divided by the total number of samples. If E and F are not events but functions, then we can extend this definition to the conditional probability of a function f(x) given that the value of a function g(x) is already known. We write this as above, <f|g>, which we define as the number of bits of f(x) that can be detected given that all bits of g(x) is known. If S is a discrete sample space (set), then we define the conditional probability <f|g> over S as the summation of #nbits(<f(x)|g(x)>/|S|, where the summation is taken over all elements x that are in S.

If h_1 to h_k is the hash functions used in a Bloom filter then we desire that

$<h_i|h_j> = 0$ for all i and j such that i != j
$<h_i|h_i> = 1$ for all i

If these conditions are satisfied, then the set of hash functions is said to be **orthogonal** or completely independent. We introduce the orthogonality measure e(H,k), where H denotes the entire set of k hash functions, as the sum of all the $<h_i|h_j>$, for all indices i and j, divided by the scale factor k. For an orthogonal set of hash functions H we have immediately that e(H,k) = 1, while for non-orthogonal hash functions we will have e(H,k) > 1. The goal of the orthogonalization procedure is to introduce a functional P (a function that operates on functions, rather than numerical values), with the following two properties:

Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$e(P(H),k) < e(H, k) \quad (4)$$

for any value w in the range (0,1], there exists an integer y such that $e(P^y(H),k) < 1 + w$.

Note that the latter condition is equivalent to the statement that $e(P^y(H),k)$ converges to 1 as y tends toward infinity, for any sample set, so long as $e(H,k) < 2$.

The functional P is also referred to as a projection operator. Before we plunge into the mathematical details of the projection, it is important to consider an example that is much easier to visualize, namely the stereographic projection. The stereographic projection is a mapping between the surface of the sphere in 3-space, centered on (0,0,0) to the (x,y) plane (the plane defined by the equation z=0). This projection is shown in figure 3. Below [26]:

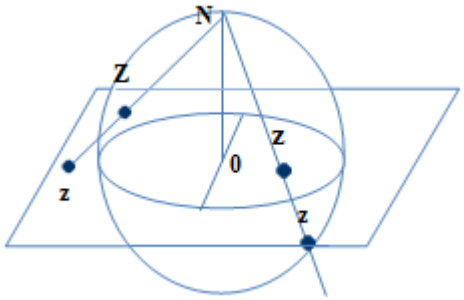

Figure 3: Stereographic Projection

Given a point Z on the surface of this sphere, a ray is shown in the North pole (the point (0,0,1)) through Z. When this ray is extended, it will eventually intersect the xy plane, at a point z as shown above. This mapping Z → z is the stereographic project from a 3-space object (the sphere) to a 2-space object (the xy plane). Note that the North pole is mapped to the point at infinity, so this projection's output is often referred to as the xy+ plane rather than the xy plane.

Recall the abbreviations $K = k^2$ and $M = 2^m$ and introduce the new abbreviation $J = K - k$. The projection operator is a projection [27], [28] from K-space to k-space that can be defined by a geometric formulation. First, we define the polynomial B(x) of a single variable x. The polynomial B(x) is formed by using a pseudo-random number generator (PRNG) to generate K coefficients, each in the range [1,M-1], which we will denote as c_i. The polynomial is then $\Sigma\ c_i x^i$ where the index i ranges from 0 to K. To define the projection in geometric terms; we use the PRNG to generate J angles in the range $(0, \pi/4)$. We use these angles to create a planar projection of B(x) from K-space to k-space. We refer to the resulting polynomial as P_1. Note that this polynomial has dimension k. We repeat the above procedure, using newly generated random angles to generate additional polynomials P_2, P_3 until we reach P_k.(similar to the procedure in [29]). We then define the projection operator P on the hash functions $H = \{ h_i \}$:

$$P(H) = P(\{ h_i \}) = \{ P_i(h_i) \} \quad (5)$$

$$\text{So that } P(H)(x) = \{ P_i(h_i(x)) \} \quad (6)$$

Since P is a projection function, it follows immediately after $e(P(H)) < e(H)$. As a result, we can create increasingly secure hash functions by simply applying P multiple times using different angles for each iteration. Note carefully that generation of the $P^y(H)$ is an offline operation. It needs to be computed only once when the Bloom filter is created. There is no impact on runtime performance.

5. INDEPENDENT SECURED HASH FUNCTION IMPROVEMENT

Suppose a browser needs to check if an input url is malicious or genuine. To check this, the browser will create a bloom filter with a known malicious url list (Add operation), which will result in a bit vector. The browser will then use this bit vector to test if a user input url is in the list of known malicious url and then act accordingly.

- Algorithm for Add: Add operation involves hashing the input data (e.g., url) to a bit vector and setting specific bit positions to 1 based on the outcome of the hash function.
- Algorithm for Test: Test involves hashing the input data and checking if all 1 position of hashed data is also set to 1 in bloom filter bit vector. The Hash function is most important part of the bloom filter.

Here is a brief method of a hash function:

1. Create some random constant value say c. The same constant has to be used for both Add operation and Test Operation.
2. Divide the input data into equal size blocks.
3. Multiply each block with the constant value and do some other operations. It's like creating a polynomial operation on the input data. Like say for input data

Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

xyz123.com, we are creating a runtime polynomial like(x*c8+y*c7+z*c6).

4. Apply hashing function operation based on the last/tail part.

5. Calculate two-part hash say h1 and h2.

Bloom filter involves multiple hash functions. We use the same hash function with some multiplication (to get nth hash function multiply by n) to derive the difference in the hash functions.

```
//Test Bloom Filter
        std::ifstream infile("test.txt");
        std::string value;
        while (infile.good())
                {
        getline(infile, value, '\n');
if (value.size() < 3) {
        continue;
                }
vector <string> record;
        splitString(value, record);
if (record.size() == 2) {
        const char* pdata = record[1].c_str();
int len = record[1].size();
bool b = bf.Test((const uint8_t*)pdata, len);
if (b == false) {
std::cout << "Not in the set :" <<record[1] <<
        std::endl;
                        }
if (b == true) {
std::cout << "May be in the set :" << record[1] <<
        std::endl;
                                }
                        }
                }
        }
        {
//Save Bloom Filter
ofstream outf("BF.dat",  td::ofstream::binary);
char acBuff[2048] = "";
int nSize = bf.m_bits.size();
        for (int i = 0; i < bf.m_bits.size(); ++i)
                {
        acBuff[i] = bf.m_bits[i];
                }
        outf.write(acBuff, nSize);
        outf.close();
        }
std::cout << "Final bloom vector" << std::endl;
for (int i = 0; i < bf.m_bits.size(); ++i)
        {
        std::cout << bf.m_bits[i];
        }
std::cout << std::endl;
return 0;
}
```

Result: With 128-bit bloom filter bit vector and three hash function we experimented. For 1000 randomly selected url, our algorithm detects 17 possibly in set and 983 definitely not in the set which means good secure and more efficiency.

6. Hash Functions Simulation Parameters

Following are the parameters used for simulation:

Table 1: SIMULATION PARAMETERS

S. No	Parameter	Value
1	Number of elements in bloom filter	1000
2	Bloom filter size	128 bits
3	Number of hash functions	3

A simulation was conducted using the above mentioned in table 1, parameters and the results were compared with the BMobishare hash functions.

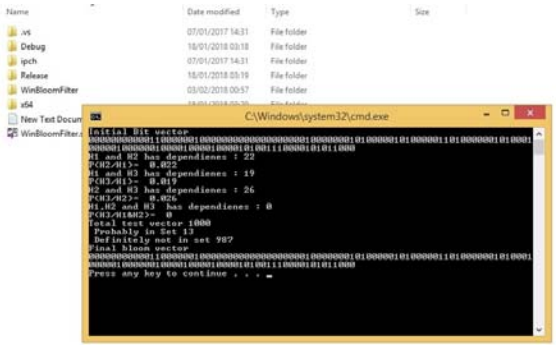

Figure 4: Proposed Hash dependency

The figure 4, shows the dependence of one hash function over other hash function using conditional probability.

First of all, it can be seen from the graph (refer to figure 5) that the chances of repetition of H2 value if H1 is known is 0.022 in our proposed hash function, while 0.033 is BMobishare solution. Similarly, Probability of occurring H3 again with the same H1 has the probability 0.019 in our case, and 0.025 in BMobishare case.

Also, the third hash function has a probability of occurring the same H3 along with the same H1 is 0.026 in our case and 0.41 in BMobishare. It can be an observer that H1, H2, and H3 when used together, are completely independent of each other. Therefore P(H2,H3/H1) =0.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

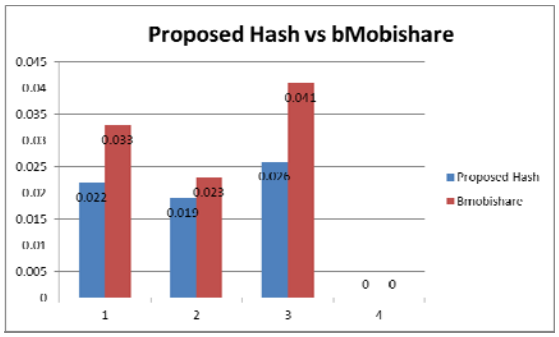

Figure 5: Hash Functions Comparison results

It is obvious from results that our proposed Hash functions are less dependent than BMobishare and therefore has better performance.

At the same time, one has to keep this fact is a mind that these hash function cannot be completely independent in the given scenario, because of the fact that, we have 1000 elements in our bloom filter and only 128 possible values of the hash functions. Therefore it is very obvious the hash values will surely be repeated 1000/128 time each. Also, hash functions are random functions within a range. Therefore we cannot make them fully independent. If we try to make them fully independent, their randomness will be destroyed, and they will be more predictable then.

Finally, the fundamental difference between our proposed approach and BMobishare is reducing the information leakage. Our evaluation is based on the two goals previously set forth: increase entropy and decrease mutual information, so that the hash functions are almost completely independent of one another. Figure 5 highlights the reduction rate of leaked information has reached P<H2 to H1>=0.033-0.022= 0.011, which means 33% of information leaked is reduced. P<H3 to H1>=0.023-0.019= 0.004, which means 17% of information leaked is reduced, and P<H3 to H2>=0.041-0.026= 0.015, which means is 37% of information leaked is reduced.

Therefore, our independent hash function slows down the information leakage from the BF from 1 bit on average per query (pure BF system), to 0.7/N bits per N queries., where N is arbitrarily large. While the compared approach used the standard bloom filter with two cryptography schemes, both public key encryption and symmetric key encryption, all of them could reduce the information leakage of 1 bit per query (pure BF system), to only 1 bit per N queries.

7. CONCLUSION AND FUTURE WORK

In this paper, we develop a method to automatically generate an arbitrary set of orthogonal and independent hash functions, with the focus on increasing entropy and decreasing mutual information between the functions. In the event that one hash function leaks information for any reason, the rest of hash functions will not be affected. These two conditions make it harder for the adversary to obtain any meaningful information. It was observed that choosing good hash functions is a critical part of creating a secured implementation. The importance of fully orthogonal hash functions cannot be overstated, so as to prevent information leakage from occurring. Close examination of the description of the hash functions shows that the construction of the functional P is, in fact, independent. Accordingly, we used the same projective construction of the hash functions for our proposed bloom filter. Our results demonstrate that the proposed hash functions are less dependent and leak when compared to previous approaches, ultimately resulting in better performance.

We have also attempted to address the security deficiencies that are commonly found in current LBS application environments. In particular, hash function address the security of backing store information, and security of the information that is visible to an eavesdropper, or to a malicious presence on the LBS provider machine. We believe this approach can be used to automatically transform the Bloom hash functions with an expected increase in security. This will be a significant advancement for privacy preservation in LBS systems and also more generally.

For future work, we plan to create a higher-order procedure in privacy-preserving framework for LBS, involving a combination of a Bloom filter with a resources. The resources here include memory (amount of data stored in the Bloom Filter bit arrays), computation time (time to generate k queries from 1 together time to execute all hash functions), and bandwidth (amount of data passing between sender and receiver per unit time).

REFERENCES:

[1] B. Niu, Z. Zhang, X. Li, and H. Li, “Privacy-area aware dummy generation algorithms for location-based services,” in Proc. Of IEEE ICC 2014.

[2] J. Krumm, “A survey of computational location privacy,” Personal Ubiquitous Comput., vol. 13, no. 6, pp. 391–399, Aug. 2009.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[3] Through spatial and temporal cloaking," in Proc. Of ACM MobiSys 2003.

[4] J. Meyerowitz and R. Roy Choudhury, "Hiding stars with fireworks: location privacy through camouflage," in Proc. Of ACM MobiCom 2009.

[5] J. Manweiler, R. Scudellari, and L. P. Cox, "Smile: Encounter-based trust for mobile social services," in Proc. Of ACM CCS 2009.

[6] W3C. (2011, Apr.) Platform for privacy preferences (p3p) project. [Online]. Available: http://www.w3.org/P3P/.

[7] H. Lu, C. S. Jensen, and M. L. Yiu, "Pad: privacy-area aware, dummy based location privacy in mobile services," in Proc. Of ACM MobiDE 2008.

[8] C.-Y. Chow, M. F. Mokbel, and X. Liu, "A peer-to-peer spatial cloaking algorithm for anonymous location-based service," in Proc. Of ACM GIS 2006.

[9] I. Bilogrevic, M. Jadliwala, K. Kalkan, J.-P. Hubaux, and I. Aad, "Privacy in mobile computing for location-sharing-based services," in Proc. Of ACM PETS 2011.

[10] B. Yao, F. Li, and X. Xiao, Secure nearest neighbor revisited. s.l. : In Proc. ICDE 2013.

[11] K. Shin, X. Ju, Z. Chen, and X. Hu, "Privacy protection for users of location-based services," Wireless Communications, IEEE, vol. 19, no. 1, pp. 30–39, 2012.

[12] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in Proc. Of IEEE INFOCOM 2014.

[13] H. Kido, Y. Yanagisawa, and T. Satoh, "An anonymous communication technique using dummies for location-based services," in Proc. Of IEEE ICPS 2005, 2005, pp. 88 – 97.

[14] A. Serjantov and G. Danezis, "Towards an information theoretic metric for anonymity," in Proc. Of ACM PETS 2003.

[15] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 789-800.

[16] N. Shen, J. Yang, K. Yuan, C. Fu and C. Jia, "An efficient and privacy-preserving location sharing mechanism", Computer Standards & Interfaces, vol. 44, pp. 102-109, 2015.

[17] V. Gupta, T. S. Vineeth, and V. Aggarwal, "Make Your Query Anonymous With Oblivious Transfer," in Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, 2015, pp. 345-349.

[18] H. Li, X. Cheng, K. Li, and Z. Tian, "Efficient Customized Privacy Preserving Friend Discovery in Mobile Social Networks," in Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on, 2015, pp. 225-234.

[19] L. Calderoni, P. Palmieri, and D. Maio, "Location privacy without mutual trust: The spatial Bloom filter," Computer Communications, vol. 68, pp. 4-16, 2015.

[20] Palmieri, L. Calderoni, D. Maio, Spatial bloom filters: enabling privacy in location-aware applications, in: Lecture Notes in Computer Science, vol. 8957, Springer, 2015, pp. 16-36.

[21] M. Ahmadi and R. Pourian, "A Bloom Filter with the Integrated Hash Table Using an Additional Hashing Function," Network Protocols and Algorithms, vol. 7, p. 24, 2015.

[22] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," Internet mathematics, vol. 1, pp. 485-509, 2004.

[23] C. Cachin, C. Crepeau, J. Marcil and G. Savvides, "Information-Theoretic Interactive Hashing and Oblivious Transfer to a Storage-Bounded Receiver", IEEE Trans. Inform. Theory, vol. 61, no. 10, pp. 5623-5635, 2015.

[24] R. L. Moy, L.-S. Chen, and L. J. Kao, "Discrete Random Variables and Probability Distributions," in Study Guide for Statistics for Business and Financial Economics, ed: Springer, 2015, pp. 67-81.

[25] L. Bunimovich, I. Cornfeld, R. Dobrushin, N. Maslova, Y. B. Pesin, A. Vershik, et al., Dynamical Systems II: Ergodic Theory with Applications to Dynamical Systems and Statistical Mechanics vol. 2: Springer Science & Business Media, 2013.

[26] B. Casselman, Feature column February 2014:Stereographic Projection, AMS, retrieved 2014-12-12.

[27] J. S. Kraft and L. C. Washington, An introduction to number theory with cryptography: CRC Press, 2016.

[28] N. D. J. S. L. Operators, "Part 1: General Theory," Interscience Publ., New York-Londou, 1958.

[29] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload attribution via hierarchical bloom filters," in Proceedings of the 11th ACM conference on Computer and communications security, 2004, pp. 31-41.